# Software Resource Disaggregation for HPC with Serverless Computing

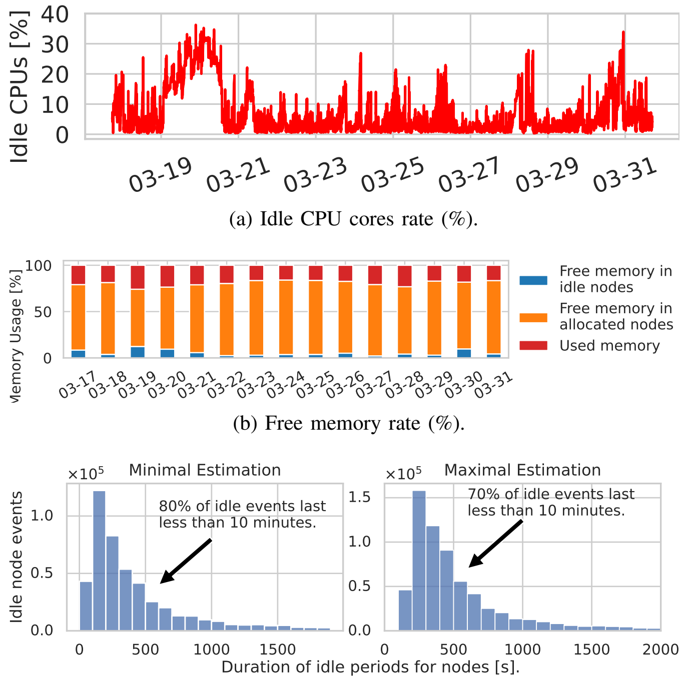
Marcin Copik*, Marcin Chrapek*, Larissa Schmid†, Alexandru Calotoiu*, Torsten Hoefler*
*Department of Computer Science, ETH Zürich, Zürich, Switzerland
†Karlsruhe Institute of Technology, Germany
*firstname.lastname@inf.ethz.ch †larissa.schmid@kit.edu

***Abstract*—Aggregated HPC resources have rigid allocation systems and programming models which struggle to adapt to diverse and changing workloads. Consequently, HPC systems fail to efficiently use the large pools of unused memory and increase the utilization of idle computing resources. Prior work attempted to increase the throughput and efficiency of supercomputing systems through workload co-location and resource disaggregation. However, these methods fall short of providing a solution that can be applied to existing systems without major hardware modifications and performance losses. In this paper, we improve the utilization of supercomputers by employing the new cloud paradigm of serverless computing. We show how serverless functions provide fine-grained access to the resources of batch-managed cluster nodes. We present an HPC-oriented Function-as-a-Service (FaaS) that satisfies the requirements of high-performance applications. We demonstrate a *software resource disaggregation* approach where placing functions on unallocated and underutilized nodes allows idle cores and accelerators to be utilized while retaining near-native performance.**

## I. Introduction

Modern HPC systems come in all shapes and sizes, with varying computing power, accelerators, memory size, and bandwidth [1]. Yet, most of them share one common characteristic: resource underutilization. Past predictions showed a pessimistic research outlook: *"the goal of achieving near 100% utilization while supporting a real parallel supercomputing workload is unrealistic"* [2]. Node utilization of supercomputer capacity varies between 80% and 94% on different systems [3–5], with up to 75% of memory is underutilized as these resources are overprovisioned for workloads with the greatest demands (Fig. 1). A 10% decrease in monthly utilization can lead to hundreds of thousands of dollars of investment in unused hardware. This gap cannot be addressed with persistent and long-running allocations. HPC operators should incentivize users to use spare CPU cores or idle GPUs to accelerate their applications, improving the cost and energy efficiency of the system. To that end, users need *fine-grained resource allocations* and *elastic programming models*.

In an HPC system, wasted resources are found in idle and allocated nodes. Most idle nodes are inactive only for several minutes (Fig. 1c) and cannot be integrated into long-running and static batch allocations. On the other hand, hardware can remain idle on an allocated node due to overprovisioning and a mismatch between available resources and job demands; even the optimal number of threads is application-specific and *"rarely equal to the number of cores on the processor"* [6].




(a) Idle CPU cores rate (%).

(b) Free memory rate (%).

(c) Duration of idle periods for nodes. Estimation from discrete sampling.

Fig. 1: Piz Daint utilization in March 2022: querying SLURM with a two-minute interval. See Sec. II-A for details.

*Evolving* and *malleable* applications cannot adjust resource allocation in rigid HPC systems [7], leading to severe underutilization, as runtime adaptivity could reduce core-hour consumption by up to 44% in malleable applications [8, 9]. The wasted on-node resources cannot be employed by another application due to the coarse granularity of batch allocations in HPC. In data centers, the problems of underutilization and coarse-grained allocations are resolved with techniques such as *resource disaggregation* and *job co-location*. However, the tight coupling of resources and performance constraints make their direct application to HPC systems difficult.

Disaggregation improves utilization by consolidating resources and allocating later in the amount needed by the application (Sec. II-B). Disaggregation targets specialized hardware [10] and improves memory's performance–per–dollar by up to 87% [11]. Memory disaggregation can be supported in hardware [11–13], but these solutions require dedicated extensions and have high costs [13]. Instead, we propose a software system that capitalizes on available high-performance interconnects and **runs on the HPC systems existing today**.

While sharing HPC nodes by co-located jobs can improve performance and efficiency [14, 15], space-sharing by applications that simultaneously stress the same resources will lead to contention [16, 17]. Memory and I/O contention cause a slowdown of up to three times and several orders of magnitude, respectively [1, 18, 19], and many systems disable node sharing for that reason. To reduce interference, users and system operators have to understand the *symbiosis* of co-located workloads (Sec. II-C). Furthermore, new approaches to security are needed when sharing bare-metal HPC nodes.

We target unused resources on idle and allocated nodes by bringing the flexibility and isolation of cloud abstraction models to HPC. We propose to use the *Function–as–a–Service (FaaS)* programming model, where users invoke fine-grained and short-running functions. Invocations are executed by the system operator on dynamically provisioned resources in a *serverless* fashion. FaaS introduces three major improvements that make it suitable for HPC resource management:

- A temporarily available node can handle time-limited functions and still be quickly drained for batch jobs;
- Functions can be co-located on the same node, where each one is using a different set of resources and is fully isolated from others;
- In FaaS, the system operator has full control over function placement, and can thus opportunistically allocate invocations to fill utilization gaps left by a batch job executing on overprovisioned hardware.

In this paper, we present the first FaaS system that implements **software disaggregation** of resources in a supercomputing system (Fig. 2). We show that dynamic function placement provides a functionally equivalent solution to disaggregated computing on homogenous resources (Sec. III). Our system allocates functions on idle resources while requiring changes to neither the hardware nor the operating systems. Using functions executing in isolated containers, we can securely share node resources between users. We then define the requirements that HPC functions must fulfill to overcome the limitations of the classical, cloud-oriented functions. We present an **HPC-centric FaaS platform** by adapting a high-performance serverless runtime rFaaS [20] to Cray supercomputers and HPC containers, and enhance it with a performance-oriented **programming model and integration** (Sec. IV). We use the pools of idle memory to host function containers, reducing cold startups and increasing resource availability. We evaluate the new system on representative HPC benchmarks (Sec. V). To the best of our knowledge, our work is the first FaaS solution specialized for HPC environments and evolving and malleable jobs.

Our paper makes the following contributions:

- We introduce a novel co-location strategy for HPC workloads that improves system utilization and uses pools of underutilized memory to host function containers.

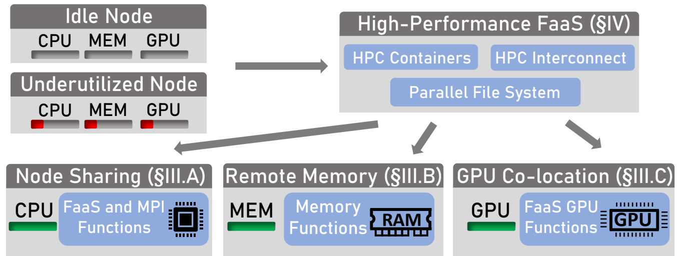



Fig. 2: **Software disaggregation with FaaS**: increasing resource utilization without modifications to HPC hardware.

- We adapt a high-performance FaaS platform to supercomputers and demonstrate the efficiency of HPC functions.
- We present an integration of FaaS into the HPC batch scheduling system and the MPI programming model, and show how functions accelerate HPC applications.

## II. Background and Motivation

Serverless provides a new resource allocation paradigm that can mitigate the low resource utilization (Sec. II-A). Functions provide a software approach to fine-grained resource allocations, overcoming the disadvantages of hardware solutions (Sec. II-B). Functions can improve on the existing techniques and billing systems for co-locating workloads (Sec. II-C).

### A. Resource Utilization in HPC

To assess the modern scale of the HPC underutilization, we analyzed the Piz Daint supercomputer, and disentangled the CPU and memory utilization in Figures 1a and 1b, respectively. While the median number of idle nodes at any sampling point was 252, the rapid and frequent changes indicate that resources do not stay idle long. The median availability time is between 5 and 6.5 minutes, and 70-80% of unallocated nodes stay idle for less than 10 minutes (Fig. 1c); similar results were observed on other systems [21]. **This gap cannot be addressed with persistent and long-running allocations.**

The aggregated and statically allocated computing nodes lead to wasting memory and network resources [10, 22, 23]. The average node memory usage can be as little as 24%, while the memory system contributes roughly 10-18% of the appropriation and operational expenditures [24, 25]. Furthermore, network and memory bandwidth utilization is very low, with occasional bursts of intensive traffic [10]. Unfortunately, this problem is fundamentally not solvable with current static allocations on homogeneous resources because these do not represent the heterogeneity of HPC workloads. While capacity computing applications with poor scaling require gigabytes of memory per process, capability computing can use less than 10% of node memory [24]. The differences between MPI ranks and applications add further imbalance.

Heterogeneity of HPC systems is increasing over time [1], with five more TOP500 systems using GPUs every year [26]. However, actual GPU utilization is often quite low. For example, on the Titan system, only 20% of the overall jobs used GPUs [27]. Furthermore, some applications that use GPUs make no use of the CPUs, reinforcing a need to co-locate GPU and CPU workloads [1].

HPC resources are underutilized and overprovisioned due to diversity of workloads. Batch jobs cannot use idle computing resources due to their short availability.

### B. Resource Disaggregation

Remote and disaggregated memory has been considered in data centers for almost a decade now [11, 13, 28–30]. Disaggregation replaces overprovisioning for the worst case with allocating for the average consumption but retaining the ability to expand resources dynamically. Remote memory has been proposed for HPC systems [22], but it comes with a bandwidth and latency penalty. While modern high-speed networks allow retaining near-native performance in some applications [30], remote memory is considered challenging for fault tolerance, and performance reasons [13].

Hardware-level solutions can elevate performance issues, e.g., by providing a dedicated high-speed network [12] and using dedicated memory blades [11]. However, many methods have not been adopted because of the major investments needed [29], such as changes in the OS and hypervisor, explicit memory management, or hardware support [11, 11, 22, 31].

Resource disaggregation is not common in HPC because of performance overheads and increased complexity.

### C. HPC Co-location

Co-location mitigates the underutilization problem by allowing more than one batch job to run on the same node. While some studies have not found a significant difference between node-sharing and exclusive jobs [32, 33], many applications experience performance degradation through contention in shared memory and network resources [16, 17]. Prior work has attempted to improve scheduling on a node by detecting sharing and contention in shared interfaces [19, 34–37].

*Symbiotic applications* can improve their performance when co-located [6, 18, 34], but determining which workload pairs show positive symbiosis is hard. Methods include user hints and offline experiments [18, 38], profiling and online monitoring [19, 36, 39], and machine learning [16]. For co-location, systems should select applications with different characteristics [15, 18, 38]. *Job striping* provides further performance benefits by spreading application processes and co-locating them with other workloads [6, 15]. Another difficulty imposed by sharing is the unfairness of traditional billing models when applied to jobs with performance impacted by the interference [6, 40]. Finally, sharing introduces security vulnerabilities when tenants are not isolated.

Node sharing is beneficial for the efficiency of HPC, as long as it avoids harmful interference. Short functions are good candidates for interference-aware co-location.

## III. Software Disaggregation with FaaS

Software disaggregation allows jobs to access remote resources by invoking serverless functions (Fig. 3). This new approach is flexible and targets only underutilized nodes. In *hardware* disaggregation, HPC applications always pay the latency penalty of accessing remote resources. In *software* disaggregation, standard HPC applications still run on unmodified nodes and have all hardware resources available locally.

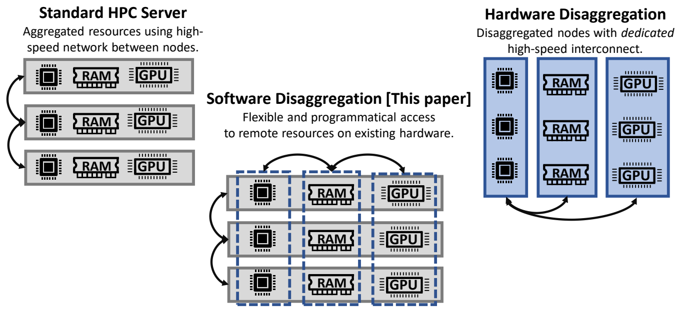
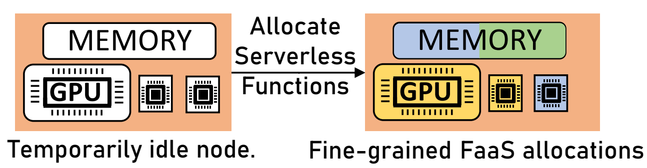



Fig. 3: **Software disaggregation**: co-location provides semantics of resource disaggregation on an unmodified system.

We begin by disaggregating resources available on **idle** nodes. Then, we go further and handle unused resources within active nodes. We focus on the three resources that can be disaggregated: **CPU cores**, **memory**, and **GPUs**. As in *job striping*, where users spread processes across a larger number of nodes to increase throughput, we recommend the same approach to leave at least one core free on each node to access idle memory and GPUs without introducing temporary oversubscription. We co-locate long-running jobs with short-term, flexible tasks with intensive but complementary resource consumption, Serverless functions are perfect for co-location: they offer fine-grained scaling, multi-tenant isolation and are very easy to checkpoint, snapshot, and migrate.

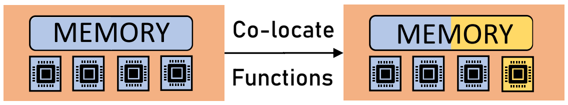
### A. Reusing Idle Nodes




While shortly available nodes are impractical for batch jobs, they can be utilized for time-limited functions that often require just a few seconds to execute [41]. Once a node is available, we deploy there a batch job with a serverless worker to start accepting function invocations. A single multi-core node can support concurrently many fine-grained functions that target newly available CPU cores, memory, and GPUs.

This scenario puts three requirements on the serverless platform: it has to integrate a new node quickly, release it immediately when the batch system needs it, and gracefully handle the node termination. We use the high-performance serverless platform rFaaS that uses leases to manage ephemeral allocations (Sec. IV). Once the node has to be returned, a signal sent to the rFaaS executor blocks any new invocations and waits for current, time-limited functions to finish. At the same time, the executor cancels existing leases, notifying the client libraries to redirect further requests to a new lease.

### B. Co-location - Sharing CPUs and more



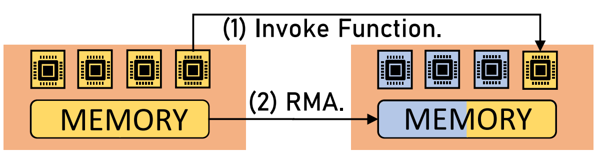

Job with dynamic parallelism. Idle resources used by FaaS.

Scientific applications often have constraints on the parallelism and problem size beyond those imposed by the hardware. For example, LULESH [42] must use a cubic number of parallel processes, making job configurations unlikely to perfectly match the available number of cores. Furthermore, the co-location of many MPI ranks leads to contention [43], forcing users to spread processes across nodes.

We improve utilization by locating FaaS executors to target unused cores in a node. Thus, our new serverless approach implements *job striping*, where MPI processes do not occupy an entire node and are co-located with other applications to better utilize resources [6, 15]. Functions can use the rest of the node's resources while minimizing the performance impact on the batch application. Since FaaS functions are easy to profile and characterize, they can be matched with jobs that present different resource availability patterns. Even when resource consumption cannot be aligned, partitioning shared memory and CPU resources can provide fairness for each application.

Furthermore, short-running MPI processes resemble FaaS functions (Sec. IV-F). Adaptive MPI implementations [8, 44] rescale applications by adding and removing processes on the fly, and new MPI ranks can be allocated in a serverless fashion. We demonstrate the benefits of co-locating such MPI processes with the example of the NAS benchmarks (Sec. V-C).

### C. Memory Service for Applications




Batch Job
Not enough memory.

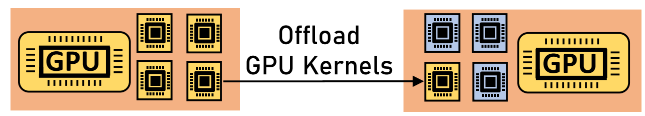
Utilize idle memory as
remote memory with FaaS.

In HPC, the memory usage of a job varies between processes and within the lifetime of the job, with a difference of up to 62.5x for some applications [24]. Furthermore, applications with poor scaling require gigabytes of memory per process, while capability computing can use less than 10% of available memory [24]. Therefore, HPC nodes will always have overprovisioned memory to support heterogeneous workloads. While high-memory jobs are not frequent in HPC systems, they still need to be accommodated, requiring idle memory reclamation to be short-term and ephemeral.

We propose three methods to effectively use idle node memory in HPC applications. First, we use free memory to keep FaaS containers warm and allow functions to be started quickly and efficiently, resolving an important issue of expensive cold starts in serverless (Sec. IV-B). Then, we use the memory to host object storage nodes (Sec. IV-D). Finally, we offer other jobs the ability to run **memory service functions**. Functions allocate a memory block and offer direct access, allowing HPC applications for remote paging [22]. We use one-sided remote memory access (RMA), which adds minimal CPU overhead to the system [29]. Thus, many memory service functions can be co-located, even with compute-intensive applications. Functions enable memory service with fine-grained scalability, easily controllable lifetime, and multi-tenant isolation.

When the batch system needs to reclaim idle memory, function containers can be migrated to other nodes and swapped to the parallel filesystem. The client library can make submitting functions seamless for the user, with functions running either directly from warm containers in otherwise idle memory or loaded from the swapped container if necessary.

### D. GPU Sharing




Providing fine-grained
access to many GPUs.

Co-locating GPU functions
on idle hardware.

While the heterogeneity of HPC systems is growing, not every application can be modified to benefit from GPU acceleration. HPC systems should co-locate CPU-only and GPU-enabled jobs, as these are often complementary [1]. For example, the main version of LULESH does not use accelerators at all, instead relies on MPI and OpenMP.

We disaggregate GPU and CPU resources by co-locating GPU functions. The function can be co-located with CPU-only functions and applications, requiring only a single CPU core to manage device and data transfers. Such functions can be launched with containers specialized for HPC systems (Sec. IV-C). Furthermore, functions can keep warm data in the device's memory until another application needs the device.

Although there exist systems for remote GPU access [45], they add latency to each command. However, applications such as machine learning inference can consist of hundreds of kernels with synchronization in between [46]. By running one CPU function process to ensure GPU access, we avoid adding inter-kernel latency in the remote GPU scenario.

### E. Co-location Policies

To decide if a function can be co-located with a given job, we must consider two factors: availability of shareable resources on a node and potential interference. For the latter, we propose selecting applications that do not stress the same resource simultaneously. To that end, we introduce a design based on the practical job history and a heuristic using the well-studied methods of HPC performance modeling (Fig. 4).

**Availability.** Disaggregation is an entirely *opt-in* system policy where users voluntarily share the node to obtain lower computing prices. We do not modify SLURM but use existing features: to enable co-location, users set the SLURM `shared` flag or submit the job to a designated partition. Job core count and memory size determine the allocated resources on each node, and serverless functions can use the remaining ones. We

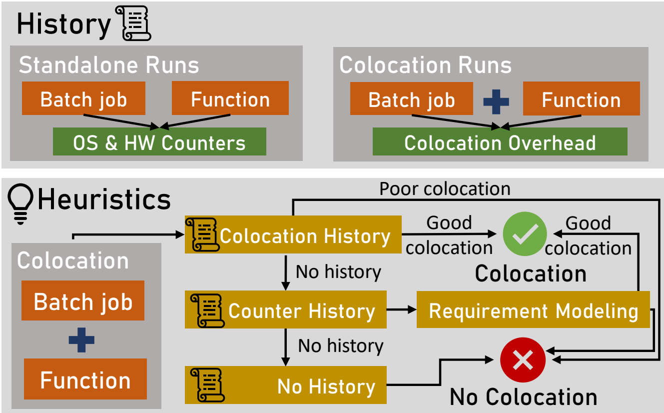



Fig. 4: Co-location policies use lightweight online monitoring.

use the SLURM GRES to determine how many and which GPUs are available [47]. We do not consider GPU sharing due to security and interference issues [48, 49]. Instead, GPU virtualization and partitioning can create isolated sub-devices in the GRES system.

**History.** First, we establish which workload pairs could be co-located. Estimating performance interference between any two applications is a complex problem [50]. Fortunately, the problem can be simplified because HPC systems serve a limited number of applications, e.g., about 115 (Blue Waters) [5] and 650 (NERSC Hopper) [51, 52]. Furthermore, HPC applications are invoked many times with varying parameters, often by multiple tenants. We can cover two-thirds of the total computation time by analyzing no more than 25 applications [5, 52].

Thanks to the limited workload diversity, we can keep a global history available to the serverless resource manager. For each co-location, we record the runtime of the batch job and the function, and compare it later against an exclusive run with the same parameters. A lightweight sampling of hardware and operating system counters gathers information on the FLOPs, memory accesses, and network traffic. Thanks to the shorter runtime and encapsulated form, serverless functions can be executed independently. Thus, when registering a new code container, the function can be profiled using user-provided or synthetic input data [53]. Operators can make function profiling mandatory or compensate users for the additional work of supplying information. Finally, counter data can be used to detect poor utilization and recommend colocation to users.

**Heuristics** We use the colocation history as a primary metric for estimating interference overhead. When the history is unavailable for the first colocation instance, we apply *resource requirement modeling* [54]. This method uses counter measurements to create performance models for different resource classes, allowing us to compare the stress factors for each application. Since models are created in the background, we can remove modeling from the critical path of scheduling. Subsequent serverless invocations will be decided with the help of history entries generated from the first co-located

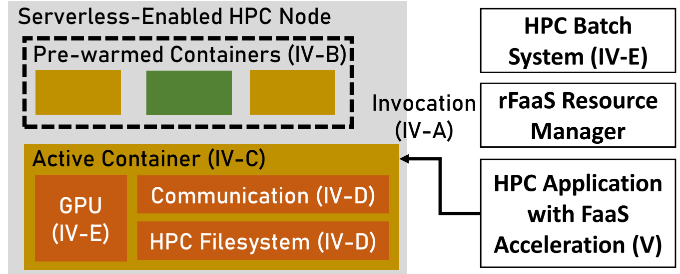



Fig. 5: Specializing serverless platform for HPC requirements.

run. In our approach, users should be incentivized with lower computation prices to provide additional information, such as specifying their application and inputs. Given the limited number of distinct applications, we can practically support many co-location combinations. Furthermore, disaggregation can be composed with methods for estimating performance interference [55] and dedicated pricing models for co-location [40].

### *F. Interference and Sharing Fairness*

Resource sharing can introduce performance interference, a concern for large-scale jobs that are significantly affected by network and OS noise [56–58]. However, the guarantee of exclusive access to resources is illusive, as jobs are affected by the inter-node sharing of network resources [59, 60]. The disaggregation must consider only the contention on node resources, as the user cannot control network performance.

We propose that the disaggregation is applied selectively to workloads, and *hero* jobs are exempted since they allocate a large fraction of the entire system and can be sensitive to interference. Since many jobs use less than 256 nodes [4, 5], disaggregation can target small and medium-scale jobs, increasing system throughput while not impacting scalability.

## IV. HPC FaaS

Serverless computing brings an abstract view of data center resources allocated on the fly by the provider and hidden from the user. This abstraction frees users from any responsibility for provisioning and allows for elastic computing, where users are billed only for the resources used. FaaS is the dominating programming model where users invoke stateless functions to the cloud. However, *classical cloud functions* have been designed for the hardware and software stack common in the cloud. The situation changes in supercomputing systems with performance-oriented architecture and programming models. We map cloud functions into HPC environments and identify five major issues that serverless faces in high-performance systems (Table I). Based on these results, we define requirements that *HPC functions* must meet to become an efficient component of a high-performance application (Fig. 5).

**rFaaS**. To demonstrate how serverless functions can be used in the HPC context, we extend the serverless platform rFaaS [20]. rFaaS allows consecutive invocations to execute on the same resource allocated with a temporary lease. Furthermore, it employs a direct RDMA connection between the client

| | Cloud FaaS | HPC FaaS |
|---|---|---|
| Network | TCP | **uGNI**, ibverbs, AWS EFA |
| Sandbox | Docker, microVM | Singularity, **Sarus** |
| Storage | Object, block | Parallel file system |
| Communication | Storage, DB, queue | Direct communication |
| Placement | VMs, Kubernetes | Batch jobs on HPC nodes |

TABLE I: Comparison of *cloud functions* environments with *HPC functions*. Technologies used in specialization for Cray machines are in bold.

| | Docker | Singularity | Sarus |
|---|---|---|---|
| Image Format | Docker | Custom | Docker-compatible |
| Repositories | Docker registry | None | Docker registry |
| Devices support | Through plugins | Automatic | Automatic |
| Resources | Native, cgroups | Automatic | Automatic |
| Batch System | None | Slurm | Slurm |
| MPI Support | None | Native | Native |

TABLE II: Comparison of container systems for cloud and HPC [71, 72]. Automatic resource and device support in Singularity and Sarus are done via Slurm.

and function executor, optimizing both the latency and the bandwidth of serverless. However, building a portable function environment on a supercomputing system is technically challenging, mainly due to specialized hardware and restricted execution models designed for static and long-running applications. We demonstrate that a serverless platform can be used efficiently in an environment that defaults towards exclusive jobs with implicit resource assignments and homogeneous applications. To that end, we present a specialization of the rFaaS platform to the Cray XC40/XC50 system Piz Daint [61].

### A. *Slow Warm Invocations*

**Problem** Each invocation of a classical function requires centralized scheduling and rerouting payload to a selected function sandbox. Even a *warm invocation* in an existing sandbox can introduce dozens of milliseconds latency [62]. However, functions require microsecond-scale latency to benefit from computing on remote resources (Sec. IV-F).
**Solution** *rFaaS* uses fast networks and a shortened invocation critical path to achieve single-digit microsecond latencies. To deploy rFaaS on a Cray system, we use the *libfabric* to target *uGNI*, the user network interface for Cray interconnects [63]. We faced two major challenges: first, the *libfabric* installation within a container must be replaced with the main system installation to achieve high performance and manage access to uGNI. We resolve the issue by manually mounting system directories in the container, as the available HPC containers do not support injection of *libfabric* at the runtime [64]. Furthermore, *uGNI* is designed to communicate within a single batch job, which is not the case in FaaS: the client in one job communicates with an executor running in another batch job. To support functions on the Cray system, we implement the allocation and distribution of security credentials DRC [65].

### B. *Expensive Cold Starts*

**Problem** When no existing sandbox can handle an invocation, a new one is allocated and initialized with user code. This *cold start* has a devastating effect on performance since it adds hundreds of milliseconds to the execution time in the best case [62, 66, 67]. Standard mitigation techniques include lightweight and prewarmed sandboxes and faster bootup methods [68–70], but the most common one is retaining containers for future invocations. However, its effectiveness is limited as idle containers are purged to free memory.

**Solution** Instead of decreasing negative cold start effects, we focus on reducing their frequency with the the help of unutilized node memory. This solution is compatible with batch systems and fits the short-term availability of resources perfectly because idle containers can be removed immediately without consequences. The availability of CPU cores to handle invocations can be guaranteed by modifying allocations to keep one or two cores per node (out of the 30 or more) available. We adjust the rFaaS resource management to target nodes with warm containers. Then, the cold start overhead is dominated only by establishing RDMA connection.

### C. *Incompatible Container Systems*

**Problem** Serverless in the cloud is dominated by Docker containers and virtual machines [69]. However, the adoption of containers is limited by security concerns, and virtual machines limit access to the accelerator and network devices. Containers must run in the *rootless* mode to avoid privilege escalation attacks. To support multi-tenancy in HPC, these issues must be mitigated while retaining near-native performance.
**Solution** Serverless sandboxes must be tailored to the needs of HPC functions, and we consider containers designed for scientific computing: Singularity [71] and Sarus [72]. Both provide native access to compute and I/O devices and integrate batch resource management (Table II). Furthermore, containers provide native support for high-performance MPI installations with dynamic relinking. This enhancement is essential for HPC functions to support elastic execution of MPI processes.

### D. *Lack of a High-Performance I/O*

**Problem** Classical serverless functions cannot accept incoming network connections in the cloud as they operate behind the NAT gateway and have no public IP address. Instead, functions must resort to using persistent cloud storage, with latencies in the tens of milliseconds, and transmitting results back to the invoker — no high-performance I/O is available to the functions in the data center ecosystem. However, HPC applications can produce terabytes of data, and in such applications, the transmission of results from a function to the invoking MPI process quickly becomes impractical. In HPC, high-performance I/O is offered through the scalable parallel filesystem [73], replacing the need for cloud storage. Furthermore, this environment is too restricted for memory service functions that accept incoming RDMA connections.

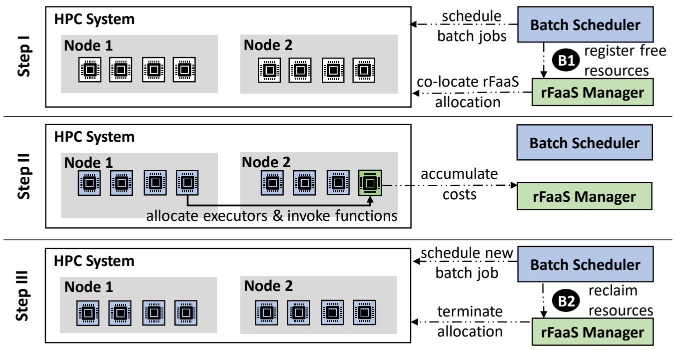



Fig. 6: Co-location made easy: *rFaaS* functions running on batch-managed clusters.

**Solution** First, we make the parallel filesystem accessible by mounting the user's partitions in the function container. This allows for storing large amounts of data from invocations and brings serverless performance in line with what is expected of HPC applications. We continue to use object storage as a warm cache for lower latency on small files (Sec. V-A). Then, we enhance functions with a portable interface to start communication, accept incoming connections, and return data without terminating the invocation, letting HPC users implement functionalities that do not fit the classical cloud model, such as remote memory.

### *E. Incompatible Resource Management*

**Problem** Serverless platforms allow to configure memory size, with CPU resources allocated proportionally to memory [74, 75]. However, software disaggregation techniques require allocations of one hardware resource while not using another one extensively. Using cluster resources requires two new functionalities: a release of nodes for FaaS processing and the removal of executors from the serverless resource pool.

**Solution** We extend the rFaaS resource management with memory and GPU device availability. Computing and memory resources are allocated and billed independently: users configure memory size according to needs and add a GPU device. Since we operate on reclaimed idle resources, there is no monetary loss from partial resource consumption by functions: every allocation increases system utilization.

Then, we implement an interface in *rFaaS* designed for integration with cluster job management systems (Fig. 6). The global resource manager offers a single REST API call to register resources (B1), which are used immediately, supporting allocations on the spare capacity available only for a very short time (Fig. 1). Released resources include CPU cores, memory, and GPUs that have not been explicitly allocated by the tenant. The allocation policy becomes *opt-in* - resources not requested by the user are not assigned by default to their jobs.

Furthermore, we allow the batch manager to retrieve resources for jobs with higher priority. A REST API *remove* call starts resource deallocation (B2). When the request is immediate (no additional computing time is allowed), all active function invocations are aborted, and *termination* replies are sent to clients. Otherwise, active invocations are allowed to finish, but no further invocations will be granted.

### *F. Incompatible Programming Model*

**Problem** Functions are designed for event-based programming in the cloud, and we need a new performance-oriented approach to benefit from serverless in HPC. While applications can use fine-grained invocations to offload computations, the performance depends on the cost of moving data and waiting for a remote task. Thus, we need a model to tell us *when* remote invocations can be integrated into HPC applications.

**Solution** We propose an integration of *rFaaS* invocations based on *LogP* models [76, 77]. The guiding principle – the application never waits for remote invocations to finish – is achieved by dividing the work such that the network transport and computation times are hidden by local work. We learn the network parameters, estimate the compute time of offloaded tasks, and measure the *rFaaS* overheads. We provide a non-exhaustive list of examples that can be adapted to offloading.

*Massively parallel applications* These applications are extremely malleable and can efficiently offload tasks as functions. A solver for the Black-Scholes equation [78] is a good example, as it generates many independent tasks with comparable runtime. To achieve the best possible performance, we measure the runtime of one task $T_{local}$ and then compare this to the runtime $T_{inv}$ of one invocation using *rFaaS*, to which we add the round-trip network time $L$. Time $T_{local}$ can be obtained with offline profiling tools common in performance modeling [43, 79], providing measurements and models for runtime decisions without the overhead of additional invocations. There exists a number $N_{local}$ of tasks such that:

$$N_{local} \cdot T_{local} \geq T_{inv} + L \tag{1}$$

Therefore, if the number of tasks is greater than $N_{local}$, up to $N_{remote}$ tasks can be computed remotely without incurring any waiting time. $N_{remote}$ is determined as the number of tasks necessary to saturate the available bandwidth $B$: $\frac{B}{Data_{inv}}$. Therefore, the throughput of the system only depends on the network link bandwidth and the amount of work available.

*Task-based applications with no sharing* Task dependency graph [80] specifies the order of execution of tasks, which can be offloaded using Eq. 1. However, the number of tasks that can be offloaded depends on the width of the task dependency graph - the wider the graph, the more parallelism is exposed. As an example, we consider the distributed prefix scan in electron microscopy image registration [81], where the width of the task graph varies significantly between program phases.

*MPI Functions* An HPC function can also implement the same computation and communication logic as an MPI process. These can be allocated with lower provisioning latency than through a batch system, and use computing resources with short-time availability. New MPI ranks can be scheduled as functions without going through the batch system, implementing the infrastructure needed to support adaptive MPI [44, 82]. In HPC, FaaS can be more than a backend for website and database functionalities; functions can represent full-fledged

| App / Functions | 1 | 2 | 4 | 8 | 12 | 16 | 24 | 32 |
|---|---|---|---|---|---|---|---|---|
| BT, W | 1 | 1.95 | 3.8 | 6.9 | 9.5 | 11.7 | 17.37 | 23.3 |
| CG, A | 1 | 1.85 | 2.8 | 4.8 | 5.8 | 6 | 8.5 | 11.4 |
| EP, W | 1 | 2 | 3.78 | 6.8 | 10.2 | 13.6 | 20.4 | 27.2 |
| LU, W | 1 | 1.9 | 3.76 | 6.7 | 9.96 | - | 19.7 | - |

TABLE III: Relative throughput of an idle-node handling rFaaS functions executing NAS benchmarks.

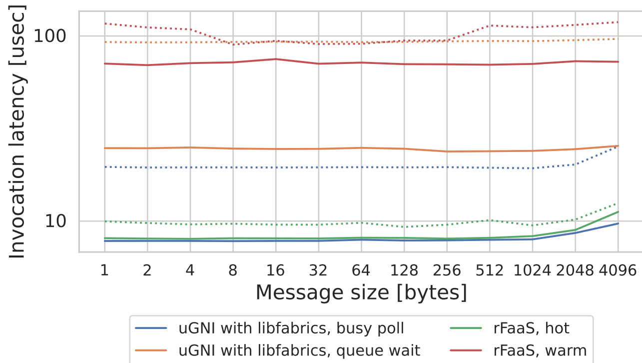



Fig. 7: Latency of rFaaS and *libfabric*. The logarithmic plot shows the median (straight) and the 95th percentile (dotted).

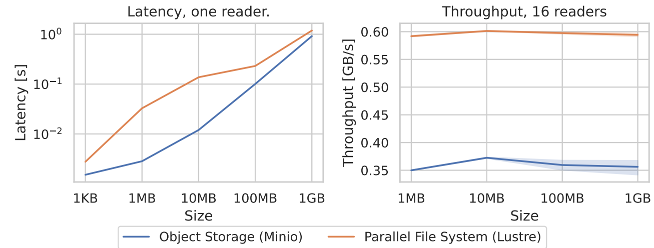



Fig. 8: Performance of I/O systems on Piz Daint: Lustre file system versus MinIO object storage.

computations with communication and synchronization [83].

## V. Case Studies

We evaluate our HPC software disaggregation approach in three steps: attempting to answer the following questions:

1) How does HPC FaaS perform on a Cray supercomputer?
2) What is the cost of co-locating functions with batch jobs?
3) Can disaggregation improve system utilization?
4) Can HPC applications on a supercomputer benefit from serverless acceleration with rFaaS?

We conduct experiments on two HPC systems:

*a) Ault:* We deploy *rFaaS* on cluster nodes with two 18-core Intel Xeon Gold 6154 CPU @ 3.00GHz and 377 GB of memory. We use Docker 20.10.5 with executor image `ubuntu:20.04`, g++ 10.2, and OpenMPI 4.1.

*b) Daint:* We deploy CPU and GPU co-location jobs on the supercomputing system Piz Daint [61]. The multi-core nodes have two 18-core Intel Xeon E5-2695 v4 @ 2.10GHz and 128 GB of memory. The GPU nodes have one 12-core Intel Xeon E5-2690 v3 @ 2.60GHz with 64 GB of memory, and a NVIDIA Tesla P100 GPU. All nodes are connected with the Cray Aries interconnect, and we extend rFaaS with `libfabric` to target the uGNI network communication library. We use Clang 12 and Cray MPICH.

### A. rFaaS on Cray Systems

First, we evaluate whether *rFaaS* provides the low-latency invocations needed in HPC (Sec. IV-A). We measure the round-trip time of function invocations on Piz Daint by using a no-op function with different sizes of input and output data. We test the *warm* invocations with non-busy waiting methods that have lower CPU overhead at the cost of increased latency, and the *hot* invocations that process invocations by continuously polling. We compare rFaaS against state-of-the-art libfabric benchmarks to check how efficiently we use the network infrastructure when serving serverless invocations (Fig. 7). While warm executors need more time to respond, the hot executions have comparable performance to bare-metal network transport and show consistent performance.

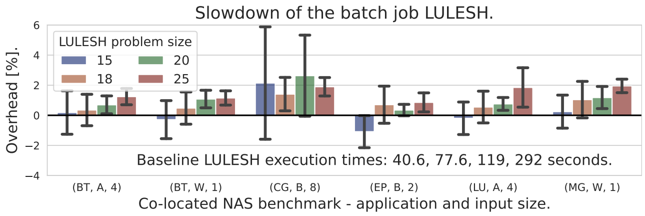
Then, we check if functions can achieve better I/O performance by using the parallel filesystem instead of object storage, a typical solution in FaaS (Sec. IV-D). To that end, we compare I/O read operations of cloud object storage MinIO [84] and the Lustre system on Piz Daint. By deploying a varying number of readers on different nodes (Fig. 8), we show that object storage delivers lower latency for smaller file sizes. However, Lustre achieves higher throughput at scale. Thus, replacing cloud storage with a filesystem provides higher I/O performance for HPC functions at no additional cost.

### B. Idle Nodes

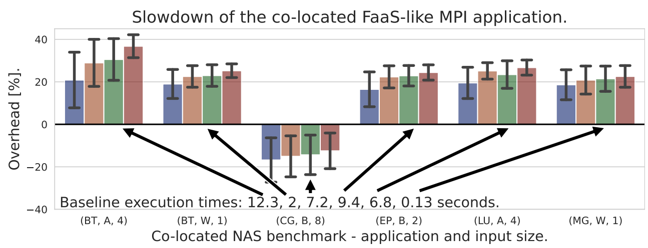
We evaluate the effectiveness of using idle nodes for short-running computations. We select serial NAS benchmarks with runtimes between 0.6 and 4.2 seconds as examples of workloads that can benefit from temporarily idle nodes. We evaluate the overall throughput of the node when increasing the number of co-located functions (Table III). To compute the co-location efficiency, we take the execution with a single rFaaS executor as baseline, and divide the obtained throughput increase by the number of colocated executors. Except for the CG benchmark, co-located functions achieve 70-80% efficiency when running simultaneously. Furthermore, the added overhead of rFaaS execution is around 13% for the shortest CG, and below 1% on other benchmarks. Thus, fine-grained and containerized HPC functions can share the node and handle significantly more invocations than an exclusive allocation.

### C. Co-location

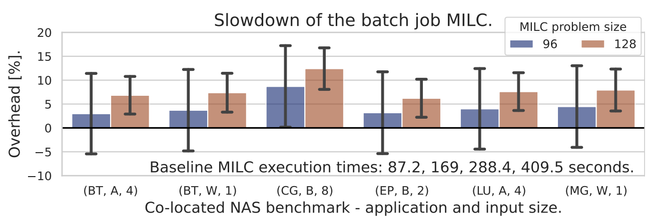
**CPU Sharing.** To evaluate the overhead of co-locating applications by sharing CPUs, we use the LULESH [42] and MILC [85] applications as a classical batch job, using 64 MPI processes and various problem sizes. We deploy LULESH on 2 Piz Daint nodes, using 32 out of the 36 available cores. It's important to note that LULESH can only run using a cubic number of processes, e.g., 8, 27, 64, 125, etc. Therefore, using all cores of a node is impossible in many configurations.




(a) Slowdown of the LULESH batch job.




(b) Slowdown of the FaaS-like MPI application co-located with LULESH.




(c) Slowdown of the MILC batch job.

Fig. 9: Overheads of batch jobs co-located with FaaS-like jobs sharing CPUs on idle cores, reported mean with standard deviation over ten repetitions.

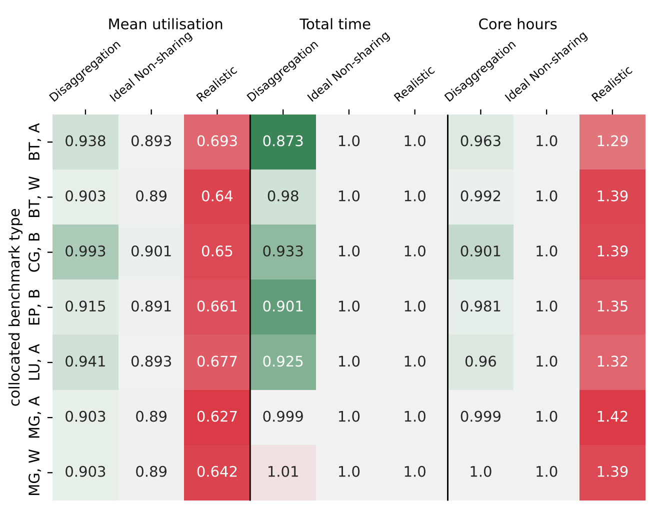
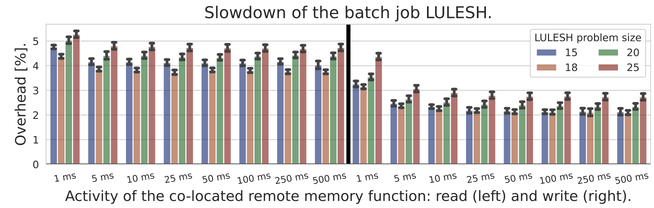
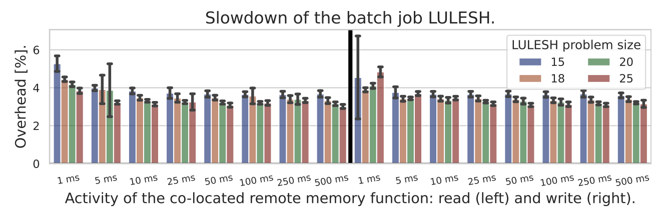



Fig. 10: System utilization of co-located execution, a partially co-located execution, and a standard exclusive node allocation.

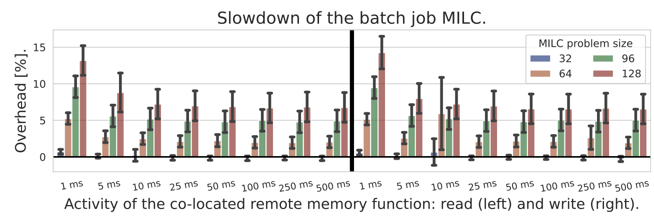
Then, we run concurrently NAS benchmarks in the Sarus container on the remaining cores, using CPU binding of tasks through SLURM. We spread MPI processes equally across two nodes and launch new executions as soon as the previous ones finish. We chose NAS benchmarks because they are a standard performance indicator [86] that represents a variety




(a) LULESH, 27 ranks.




(b) LULESH, 125 ranks.



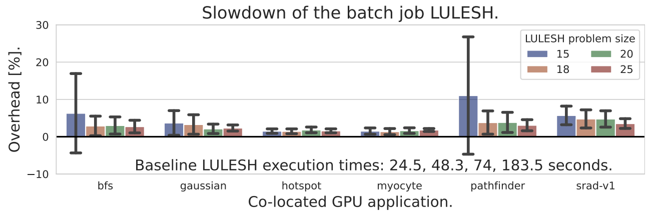

(c) MILC, 32 ranks.

Fig. 11: Overhead of batch jobs co-located with *rFaaS* functions providing remote memory. Reported mean with standard deviation over ten repetitions.

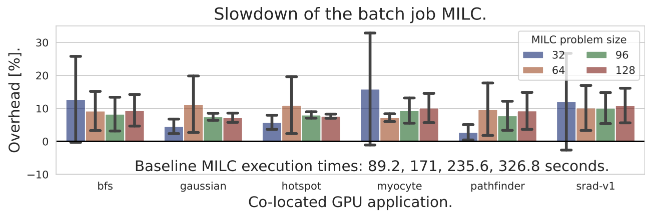
of compute and communication-bound tasks [87, 88], with different memory size [89], data locality and access patterns [90], and communication volume [88, 91]. Since they have a short runtime that corresponds with execution characteristics of functions [41, 62, 92], they can represent a FaaS-like workload covering the large diversity of computational patterns that can appear when offloading HPC tasks to serverless.

Fig. 9 shows that the impact of co-location on the batch job with this workload is **negligible**, with changes in LULESH performance explained by the measurement noise. More importantly, only requesting 32 out of 36 cores on each node translates to a core-hour cost reduction of $\approx 11\%$, more than offsetting any impact of co-location. We evaluate the increased system utilization by comparing our co-location with two other scenarios: a realistic exclusive node allocation and an *ideal* allocation where small-scale jobs execute exclusively but are billed for used cores only. Figure 10 demonstrates significant utilization improvements of up to 52%. While the performance loss on the container is higher, it is not a limitation as HPC functions effectively provide users with a way to use resources that would otherwise be wasted: a co-located FaaS-like application is essentially free.

**Memory Sharing.** We evaluate the impact of allowing *rFaaS* to use idle memory. On the Ault system, we run LULESH using 27 and 125 cores, and MILC using 32 cores out of 36 available cores. We deploy *rFaaS* with the remote memory function setup in a Docker container. The




(a) Slowdown of the LULESH batch job.

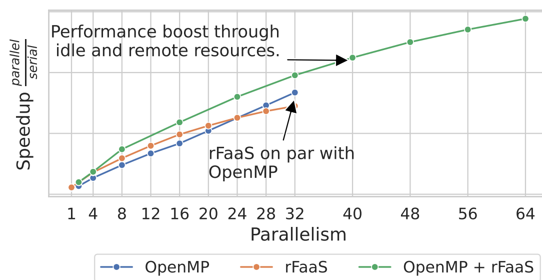



(b) Slowdown of the MILC batch job.

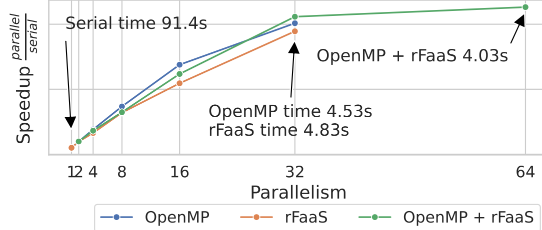
Fig. 12: Overheads of batch jobs sharing node with GPU applications. Reported mean with standard deviation.

*rFaaS* function allocates 1 GB of pinned memory available for RDMA operations, and returns the buffer data to the owner. While running LULESH and MILC, we perform RDMA read and write operations of 10 MB repeatedly with different intervals between repetitions to test how additional traffic affects performance (Fig. 11). The results show that LULESH is not sensitive to the variable perturbation, regardless of problem size, while MILC is more sensitive at larger problem sizes. When scaling LULESH to multiple nodes, the overall runtime of the job is affected minimally, proving that compute-intensive applications can share network bandwidth to improve the overall system throughput. Interestingly, the rate at which data is read or written does not affect performance, even when adding 10GB/s of traffic to the system. This result is not surprising, as MILC is known the be memory-intensive [93, 94], and extremely sensitive to both memory bandwidth [95, 96] and to network performance [95, 97–99]. The memory serving function impacts both the available memory bandwidth and the intra-node communication based on shared memory.

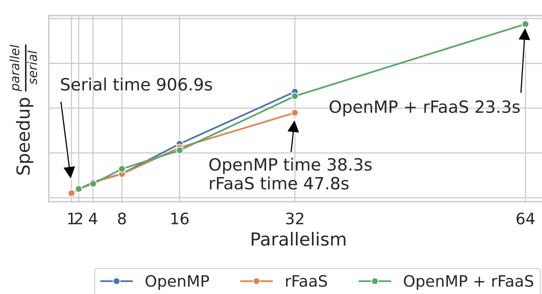
**GPU Sharing.** We also run the GPU version of LULESH and MILC on three GPU nodes of the Piz Daint system using 27 ranks and 9 cores out of the 12 available on each node for LULESH and 32 ranks (divided as 11, 11, and 10 cores) for MILC. Then, we run Rodinia GPU benchmarks [100] in a Sarus container (Fig. 12), binded to one of the remaining CPU cores through SLURM. These benchmarks simulate GPU functions as each only takes a few hundred milliseconds. The overall overhead remains very low ($< 5\%$), except for two outliers ($6.1\%$ and $10.5\%$) – both encountered only for the smallest problem size of LULESH. However, only requesting 9 out of 12 cores on each GPU translates to a core-hour cost reduction of $25\%$, yet again more than offsetting any impact of co-location. For MILC, the overhead is slightly higher, with the smaller problem sizes experiencing a stronger perturbation.




(a) Black-Scholes method, 100 repetitions.




(b) OpenMC, 1,000 particles.




(c) OpenMC, 10,000 particles.

Fig. 13: *rFaaS* in practice, reporting medians.

Co-location of compute-intensive and memory-bound HPC applications with *rFaaS* functions and FaaS-like HPC workloads can be achieved without major overheads in batch jobs, regardless of the shared resource. Allocating only required resources reduces batch job costs, even when considering co-location overheads.

### D. HPC Integration

To prove that offloading computations to HPC functions offers performance competitive to native applications, we integrate rFaaS functions into OpenMP benchmarks executed on the Piz Daint and the Ault cluster. In each application, we move loop code to a separate function, allocate arrays in RDMA-enabled buffers, and replace OpenMP pragmas with *rFaaS* dispatch. We compare the runtimes of benchmarks using OpenMP with runs where the amount of resources has been doubled by allocating one function for each thread. Thus, we test acceleration by offloading computations to cheap and idle resources while constrained by the network bandwidth. This setup allows the dynamic adaptation of parallel allocation and scaling beyond resources available on a single node.

*1) Use-case: Black-Scholes simulation:* Figure 13a demonstrates an OpenMP Black-Scholes benchmark from the PAR-

SEC suite executed with 100 iterations, modified to use *rFaaS* offloading by changing 85 lines of code. The serial execution takes 726 milliseconds on an input of 229 MB. We compare the OpenMP version against complete remote execution with *rFaaS*, and against doubling parallel resources with cheap serverless allocation. The application demonstrates efficient offloading of computations until network saturation is reached.

*2) Use-case: OpenMC:* Figure 13b and Figure 13c demonstrate OpenMC [101], a Monte Carlo particle transport code modified to use *rFaaS* offloading by adding 180 lines of code. We execute the *opr* benchmark [102] modeling an Optimized Power Reactor for the input configurations of simulating 1,000 and 10,000 particles on Ault nodes with two AMD EPYC 7742 64-Core Processor @ 2.25GHz and 256 GB memory each. In both configurations, functions and clients read 410.8 MB input from the parallel filesystem. We compare the OpenMP version against complete remote execution with *rFaaS* and against doubling parallel resources with cheap serverless allocation.

HPC functions can improve massively parallel OpenMP applications, even with millisecond-scale runtime.

## VI. Related Work

Prior work on HPC-oriented serverless targeted scientific applications on a federated platform funcX [103] and HPC workflows [104, 105]. We introduce a platform redesigned to match the software and hardware stack of supercomputers and share nodes with batch jobs. Furthermore, funcX is designed for computation across federated resources, which comes with dozens of milliseconds of invocation latency. Our HPC-specialized rFaaS brings microsecond-scale invocations within the same system, efficiently offloading functions running for less than 100-200 ms (Sec. V-D).

**Resource Underutilization** Snavely et al. [6] proposed node sharing with co-location of applications with compatible resource consumption patterns. However, the detection and avoidance of performance interference requires changes to pricing models [40], batch systems, and schedulers [40, 106–108]. Instead, we propose a decentralized approach with fine-grained functions that does not require changes in batch systems and online monitoring for interference. Przybylski et al. [21] proposed to improve HPC system utilization by using idle nodes for serverless platform OpenWhisk. However, their approach does not consider co-location and fine-grained access to heterogeneous node resources, and uses a generic serverless platform handling cloud workloads. We define requirements for HPC functions, specialize them to supercomputing environments, and demonstrate integration into HPC applications.

In the cloud, utilization is improved by harvesting over-allocated virtual machine resources, including CPU cores [109] and memory [110]. Harvested VMs can be used to host invocations of serverless functions [110, 111]. Freyr and Libra conduct resource harvesting from over-provisioned serverless functions [53, 112]. Our work is focused on HPC resources available for a short time, while the allocation of virtual machines can last for months. Instead of modifying a classical FaaS platform such as OpenWhisk, we specialize in high-performance serverless systems to HPC systems and applications.

**Elastic MPI** Adaptive MPI frameworks implement restarting applications [113], reconfiguration [114], processor virtualization [115], and checkpointing with migration [116–118]. In contrast, HPC functions bring a dynamic acceleration with resources allocated on the fly, and require neither restarting nor reconfiguring the MPI program to incorporate new resources. Supporting malleable and evolving applications requires changes in schedulers and batch systems [82, 119, 120], and MPI extensions are needed to extend and shrink the number of processes [44]. Serverless functions can implement malleable and evolving jobs with high resource availability.

## VII. Discussion

This paper proposes a functionally equivalent alternative to hardware resource disaggregation, achieved by co-locating a serverless platform with classical HPC batch jobs. In the following, we discuss several questions our approach raises.

**How does our solution differ from cloud functions?** While exploring secure multi-tenancy via serverless techniques is already new in the context of HPC, we go beyond that: we use co-location only as the starting point and leverage *rFaaS* to allow the different resource subsets to be accessed separately. Furthermore, unlike the multi-tenant co-location of functions in a cloud, we focus on providing access to different resource categories in the existing node model of an HPC data center.

**What are the limitations imposed by rFaaS?** The programming model offloads tasks to elastic executors, similarly to many other serverless approaches to parallel computing [121–123]. Our disaggregation solution relies on network bandwidth to move tasks without significant delays. Furthermore, HPC applications are adapted to support serverless offloading, a challenge faced by all applications using FaaS.

**How does our approach compare to hardware solutions?** Memory disaggregation needs a software layer for remote paging [124], which can be fully realized in our solution. Since we target idle memory that can be reclaimed, we propose swapping and migrations to avoid data loss (Sec. III-C). However, this limitation does not concern applications using ephemeral memory, e.g., in-memory caching for parallel filesystems. Our approach uses existing HPC interconnects and avoids additional costs. There is no penalty for running an unmodified HPC application on an aggregated system, whereas disaggregation always adds latency to reach remote resources. Although emerging hardware disaggregation technologies can offer nanosecond-scale latency for remote memory, many high-performance applications benefit from remote memory [29, 30, 125], indicating that a software-based approach can offer competitive performance at lower costs.

**Which applications benefit from co-location?** We demonstrate on two representative HPC applications that software disaggregation increases the system's utilization thanks to tolerable performance overheads. However, co-location has been shown to cause minor slowdowns and increase overall system throughput in many HPC applications, including memory-bound and network-sensitive workloads [6, 106, 107, 126–128]. Job striping and spreading [6, 107] can be realized in our system due to the reduced costs of under-allocation.

**How can HPC benefit from short functions?** Different HPC applications can benefit from dynamically offloading computations to idle nodes. Major examples include distributed tasking systems such as Dask and Ray [129–131], malleable and evolving MPI applications [8, 9], HPC workflows [104, 105], and offloading OpenMP loops to remote devices [132, 133]. By including GPU functions, we can support the rapidly growing space of machine learning inference, a computationally expensive and latency-constrained task.

Serverless requires modifications to serialize inputs and compile function code for FaaS deployment. This process can be eased with a single-source compiler [134], and our remote memory can support offloading OpenMP applications. Furthermore, function invocations can be determined by compilers that analyze the parallelism and data movement [135].

## VIII. Conclusions

HPC suffers from underutilization since many systems do not have access to hardware resource disaggregation. Therefore, we propose a *software disaggregation* approach to efficiently co-locate long-running batch jobs with serverless functions. We design targeted FaaS approaches for the three main domains of software disaggregation: idle processors, memory, and accelerators. Using a high-performance serverless platform, we demonstrate that targeting idle and partially allocated nodes allows HPC users to benefit from reclaimed resources while minimizing performance losses, improving system throughput by up to 53% and supporting remote memory with up to 1GB/s traffic without significant performance overheads from node sharing. Finally, we provide users with a path to use the reclaimed resources to accelerate HPC applications.

## Acknowledgment

This project has received funding from EuroHPC-JU under grant agreements DEEP-SEA, No 95560, and RED-SEA, No 055776. This work was partially supported by the ETH Future Computing Laboratory (EFCL), financed by a donation from Huawei Technologies. We would like to thank the Swiss National Supercomputing Centre (CSCS) for providing us with access to their supercomputing infrastructure.

## References

[1] A. Khan, H. Sim, S. S. Vazhkudai, A. R. Butt, and Y. Kim, "An analysis of system balance and architectural trends based on top500 supercomputers," in *The International Conference on High Performance Computing in Asia-Pacific Region*, ser. HPC Asia 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 11–22. [Online]. Available: https://doi.org/10.1145/3432261.3432263

[2] J. P. Jones and B. Nitzberg, "Scheduling for parallel supercomputing: A historical perspective of achievable utilization," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 1–16.

[3] H. You and H. Zhang, "Comprehensive workload analysis and modeling of a petascale supercomputer," in *Job Scheduling Strategies for Parallel Processing*, W. Cirne, N. Desai, E. Frachtenberg, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 253–271.

[4] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari, "Job characteristics on large-scale systems: Long-term analysis, quantification, and implications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[5] M. D. Jones, J. P. White, M. Innus, R. L. DeLeon, N. Simakov, J. T. Palmer, S. M. Gallo, T. R. Furlani, M. T. Showerman, R. Brunner, A. Kot, G. H. Bauer, B. M. Bode, J. Enos, and W. T. Kramer, "Workload analysis of blue waters," *CoRR*, vol. abs/1703.00924, 2017. [Online]. Available: http://arxiv.org/abs/1703.00924

[6] A. D. Breslow, L. Porter, A. Tiwari, M. Laurenzano, L. Carrington, D. M. Tullsen, and A. E. Snavely, "The case for colocation of high performance computing workloads," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 2, pp. 232–251, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3187

[7] D. G. Feitelson and L. Rudolph, "Toward convergence in job schedulers for parallel supercomputers," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 1–26.

[8] A. Mo-Hellenbrand, I. Comprés, O. Meister, H.-J. Bungartz, M. Gerndt, and M. Bader, "A large-scale malleable tsunami simulation realized on an elastic mpi infrastructure," in *Proceedings of the Computing Frontiers Conference*, ser. CF'17. New York, NY, USA: Association for Computing Machinery, 2017, p. 271–274. [Online]. Available: https://doi.org/10.1145/3075564.3075585

[9] S. Iserte, R. Mayo, E. S. Quintana-Ortí, V. Beltran, and A. J. Peña, "Efficient scalable computing through flexible applications and adaptive workloads," in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, 2017, pp. 180–189.

[10] 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 Trans. Archit. Code Optim.*, jan 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3514245

[11] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, “Disaggregated memory for expansion and sharing in blade servers,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 267–278. [Online]. Available: https://doi.org/10.1145/1555754.1555789

[12] C. Pinto, D. Syrivelis, M. Gazzetti, P. Koutsovasilis, A. Reale, K. Katrinis, and H. P. Hofstee, “Thymesisflow: A software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 868–880.

[13] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei, “Remote memory in the age of fast networks,” in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 121–127. [Online]. Available: https://doi.org/10.1145/3127479.3131612

[14] C. Iancu, S. Hofmeyr, F. Blagojević, and Y. Zheng, “Oversubscription on multicore processors,” in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1–11.

[15] M. J. Koop, M. Luo, and D. K. Panda, “Reducing network contention with mixed workloads on modern multicore, clusters,” in *2009 IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–10.

[16] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, and J. Pei, “A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’12. Washington, DC, USA: IEEE Computer Society Press, 2012.

[17] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim, “Measuring interference between live datacenter applications,” in *SC ’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–12.

[18] J. Weinberg and A. Snavely, “Symbiotic space-sharing on sdsc’s datastar system,” in *Job Scheduling Strategies for Parallel Processing*, E. Frachtenberg and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 192–209.

[19] C. Antonopoulos, D. Nikolopoulos, and T. Papatheodorou, “Scheduling algorithms with bus bandwidth considerations for smps,” in *2003 International Conference on Parallel Processing, 2003. Proceedings.*, 2003, pp. 547–554.

[20] M. Copik, K. Taranov, A. Calotoiu, and T. Hoefler, “rFaaS: Enabling High Performance Serverless with RDMA and Leases,” in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 897–907. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPS54959.2023.00094

[21] B. Przybylski, M. Pawlik, P. Żuk, B. Lagosz, M. Malawski, and K. Rzadca, “Using unused: Non-invasive dynamic faas infrastructure with hpc-whisk,” in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15.

[22] I. Peng, R. Pearce, and M. Gokhale, “On the memory underutilization: Exploring disaggregated memory on hpc systems,” in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 183–190.

[23] G. Panwar, D. Zhang, Y. Pang, M. Dahshan, N. DeBardeleben, B. Ravindran, and X. Jian, “Quantifying memory underutilization in hpc systems and using it to improve performance via architecture support,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’52. New York, NY, USA: Association for Computing Machinery, 2019, p. 821–835. [Online]. Available: https://doi.org/10.1145/3352460.3358267

[24] D. Zivanovic, M. Pavlovic, M. Radulovic, H. Shin, J. Son, S. A. Mckee, P. M. Carpenter, P. Radojković, and E. Ayguadé, “Main memory in hpc: Do we need more or could we live with less?” *ACM Trans. Archit. Code Optim.*, vol. 14, no. 1, mar 2017. [Online]. Available: https://doi.org/10.1145/3023362

[25] L. A. Barroso, U. Hölzle, and P. Ranganathan, “The datacenter as a computer: Designing warehouse-scale machines,” *Synthesis Lectures on Computer Architecture*, vol. 13, no. 3, pp. i–189, 2018.

[26] “TOP500, November 2021,” https://www.top500.org/lists/top500/2021/11/, 2021, accessed: 2022-03-10.

[27] F. Wang, S. Oral, S. Sen, and N. Imam, “Learning from five-year resource-utilization data of titan system,” in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–6.

[28] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, “FaRM: Fast remote memory,” in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 401–414. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevic

[29] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, “Efficient memory disaggregation with infiniswap,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 649–667. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[30] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, “Network requirements for resource disaggregation,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 249–264. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[31] S. Liang, R. Noronha, and D. K. Panda, “Swapping to remote memory over infiniband: An approach using a high performance network block device,” in *2005 IEEE International Conference on Cluster Computing*, 2005, pp. 1–10.

[32] J. P. White, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, A. Ghadersohi, C. D. Cornelius, A. K. Patra, J. C. Browne, W. L. Barth, and J. Hammond, “An analysis of node sharing on hpc clusters using xdmod/tacc_stats,” in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, ser. XSEDE ’14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2616498.2616533

[33] N. A. Simakov, R. L. DeLeon, J. P. White, T. R. Furlani, M. Innus, S. M. Gallo, M. D. Jones, A. Patra, B. D. Plessinger, J. Sperhac, T. Yearke, R. Rathsam, and J. T. Palmer, “A quantitative analysis of node sharing on hpc clusters using xdmod application kernels,” in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, ser. XSEDE16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2949550.2949553

[34] A. Snavely, D. M. Tullsen, and G. Voelker, “Symbiotic jobscheduling with priorities for a simultaneous multithreading processor,” in *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS ’02. New York, NY, USA: Association for Computing Machinery, 2002, p. 66–76. [Online]. Available: https://doi.org/10.1145/511334.511343

[35] J. Liedtke, M. Völp, and K. Elphinstone, “Preliminary thoughts on memory-bus scheduling,” in *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*, ser. EW 9. New York, NY, USA: Association for Computing Machinery, 2000, p. 207–210. [Online]. Available: https://doi.org/10.1145/566726.566768

[36] E. Koukis and N. Koziris, “Memory and network bandwidth aware scheduling of multiprogrammed workloads on clusters of smps,” in *12th International Conference on Parallel and Distributed Systems - (ICPADS’06)*, vol. 1, 2006, pp. 10 pp.–.

[37] D. Xu, C. Wu, and P.-C. Yew, “On mitigating memory bandwidth contention through bandwidth-aware scheduling,” in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT ’10. New York, NY, USA: Association for Computing Machinery, 2010, p. 237–248. [Online]. Available: https://doi.org/10.1145/1854273.1854306

[38] J. Weinberg and A. Snavely, “User-guided symbiotic space-sharing of real workloads,” in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 345–352. [Online]. Available: https://doi.org/10.1145/1183401.1183450

[39] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa, “Reqos: Reactive static/dynamic compilation for qos in warehouse scale computers,” *SIGPLAN Not.*, vol. 48, no. 4, p. 89–100, mar 2013. [Online]. Available: https://doi.org/10.1145/2499368.2451126

[40] A. D. Breslow, A. Tiwari, M. Schulz, L. Carrington, L. Tang, and J. Mars, “Enabling fair pricing on hpc systems with node sharing,” in *SC ’13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[41] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider,” in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 205–218. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad

[42] R. D. Hornung, J. A. Keasler, and M. B. Gokhale, “Hydrodynamics challenge problem,” 6 2011. [Online]. Available: https://www.osti.gov/biblio/1117905

[43] M. Copik, A. Calotoiu, T. Grosser, N. Wicki, F. Wolf, and T. Hoefler, “Extracting clean performance models from tainted programs,” in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 403–417. [Online]. Available: https://doi.org/10.1145/3437801.3441613

[44] I. Comprés, A. Mo-Hellenbrand, M. Gerndt, and H.-J. Bungartz, “Infrastructure and api extensions for elastic execution of mpi applications,” in *Proceedings of the 23rd European MPI Users’ Group Meeting*, ser. EuroMPI 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 82–97. [Online]. Available: https://doi.org/10.1145/2966884.2966917

[45] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Ortí, “rcuda: Reducing the number of gpu-based accelerators in high performance clusters,” in *2010 International Conference on High Performance Computing & Simulation*. IEEE, 2010, pp. 224–231.

[46] L. Tobler, "Gpuless – serverless gpu functions," 2022. [Online]. Available: https://spcl.inf.ethz.ch/Publications/.pdf/tobler-gpu-thesis.pdf

[47] (2019) Slurm generic resource (gres) scheduling. https://slurm.schedmd.com/gres.html. Accessed: 2020-01-20.

[48] A. Nayak, P. B., V. Ganapathy, and A. Basu, "(mis)managed: A novel tlb-based covert channel on gpus," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 872–885. [Online]. Available: https://doi.org/10.1145/3433210.3453077

[49] G. Gilman and R. J. Walls, "Characterizing concurrency mechanisms for nvidia gpus under deep learning workloads," *Performance Evaluation*, vol. 151, p. 102234, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166531621000511

[50] C. Delimitrou and C. Kozyrakis, "Qos-aware scheduling in heterogeneous datacenters with paragon," *ACM Trans. Comput. Syst.*, vol. 31, no. 4, dec 2013. [Online]. Available: https://doi.org/10.1145/2556583

[51] Z. Zhao, N. J. Wright, and K. Antypas, "Effects of hyper-threading on the nersc workload on edison," *Cray User Group CUG (May 2013)*, 2013. [Online]. Available: https://cug.org/proceedings/cug2013_proceedings/includes/files/pap106.pdf

[52] K. Antypas, B. Austin, T. Butler, R. Gerber, C. Whitney, N. Wright, W.-S. Yang, and Z. Zhao, "Nersc workload analysis on hopper," *Lawrence Berkeley National Laboratory Technical Report*, vol. 6804, p. 15, 2013. [Online]. Available: https://www.nersc.gov/assets/Trinity--NERSC-8-RFP/Documents/NERSCWorkloadAnalysisFeb2013.pdf

[53] H. Yu, C. Fontenot, H. Wang, J. Li, X. Yuan, and S.-J. Park, "Libra: Harvesting idle resources safely and timely in serverless clusters," in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 181–194. [Online]. Available: https://doi.org/10.1145/3588195.3592996

[54] A. Calotoiu, A. Graf, T. Hoefler, D. Lorenz, S. Rinke, and F. Wolf, "Lightweight requirements engineering for exascale co-design," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 201–211.

[55] A. Shah, M. Müller, and F. Wolf, "Estimating the impact of external interference on application performance," in *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, pp. 46–58.

[56] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the influence of system noise on large-scale applications by simulation," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–11.

[57] ——, "The impact of network noise at large-scale communication performance," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, 2009, pp. 1–8.

[58] D. De Sensi, T. De Matteis, K. Taranov, S. Di Girolamo, T. Rahn, and T. Hoefler, "Noise in the clouds: Influence of network performance variability on application scalability," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, dec 2022. [Online]. Available: https://doi.org/10.1145/3570609

[59] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: Performance degradation due to nearby jobs," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[60] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Watch out for the bully! job interference study on dragonfly network," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 750–760.

[61] "Piz Daint," https://www.cscs.ch/computers/piz-daint/, 2021, accessed: 2020-01-20.

[62] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski, and T. Hoefler, "Sebs: A serverless benchmark suite for function-as-a-service computing," in *Proceedings of the 22nd International Middleware Conference*, ser. Middleware '21. Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3464298.3476133

[63] H. Pritchard, E. Harvey, S.-E. Choi, J. Swaro, and Z. Tiffany, "The gni provider layer for ofi libfabric," in *Proceedings of Cray User Group Meeting, CUG*, vol. 2016, 2016.

[64] A. Madonna and T. Aliaga, "Libfa bric-based injection solutions for portable containerized mpi applications," in *2022 IEEE/ACM 4th International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 2022, pp. 45–56.

[65] J. Shimek, J. Swaro, and M. Saint Paul, "Dynamic rdma credentials," in *Cray User Group (CUG) Meeting*, 2016.

[66] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, "Cold start influencing factors in function as a service," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 181–188.

[67] P. Silva, D. Fireman, and T. E. Pereira, "Prebaking functions to warm the serverless cold start," in *Proceedings of the 21st International Middleware Conference*, ser. Middleware '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–13. [Online]. Available: https://doi.org/10.1145/3423211.3425682

[68] E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "SOCK: Rapid task provisioning with serverless-optimized containers," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 57–70. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/oakes

[69] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 419–434. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/agache

[70] D. Du, T. Yu, Y. Xia, B. Zang, G. Yan, C. Qin, Q. Wu, and H. Chen, "Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 467–481. [Online]. Available: https://doi.org/10.1145/3373376.3378512

[71] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, vol. 12, no. 5, p. e0177459, May 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0177459

[72] L. Benedicic, F. A. Cruz, A. Madonna, and K. Mariotti, "Sarus: Highly scalable docker containers for hpc systems," in *International Conference on High Performance Computing*. Springer, 2019, pp. 46–60.

[73] B. Behzad, S. Byna, Prabhat, and M. Snir, "Optimizing i/o performance of hpc applications with autotuning," *ACM Trans. Parallel Comput.*, vol. 5, no. 4, mar 2019. [Online]. Available: https://doi.org/10.1145/3309205

[74] "AWS API Pricing," https://aws.amazon.com/api-gateway/pricing/, 2020, accessed: 2020-08-20.

[75] "Google Cloud Functions Pricing," https://cloud.google.com/functions/pricing, 2020, accessed: 2020-08-20.

[76] 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*, ser. PPOPP '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 1–12. [Online]. Available: https://doi.org/10.1145/155332.155333

[77] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm, "LogfP - A Model for small Messages in InfiniBand," in *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS), PMEO-PDS'06 Workshop*, Apr. 2006.

[78] A. Heinecke, S. Schraufstetter, and H.-J. Bungartz, "A highly parallel black–scholes solver based on adaptive sparse grids," *Int. J. Comput. Math.*, vol. 89, no. 9, p. 1212–1238, Jun. 2012. [Online]. Available: https://doi.org/10.1080/00207160.2012.690865

[79] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2503210.2503277

[80] S. Shudler, A. Calotoiu, T. Hoefler, and F. Wolf, "Isoefficiency in practice: Configuring and understanding the performance of task-based applications," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 131–143. [Online]. Available: https://doi.org/10.1145/3018743.3018770

[81] M. Copik, T. Grosser, T. Hoefler, P. Bientinesi, and B. Berkels, "Work-stealing prefix scan: Addressing load imbalance in large-scale image registration," *IEEE Transactions on Parallel & Distributed Systems*, vol. 33, no. 03, pp. 523–535, mar 2022.

[82] M. Chadha, J. John, and M. Gerndt, "Extending slurm for dynamic resource-aware adaptive batch scheduling," in *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2020, pp. 223–232.

[83] M. Copik, R. Böhringer, A. Calotoiu, and T. Hoefler, "Fmi: Fast and cheap message passing for serverless functions," in *Proceedings of the 37th International Conference on Supercomputing*, ser. ICS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 373–385. [Online]. Available: https://doi.org/10.1145/3577193.3593718

[84] "MinIO Object Storage," min.io, 2019, accessed: 2023-10-05.

[85] C. Bernard, M. C. Ogilvie, T. A. DeGrand, C. E. DeTar, S. A. Gottlieb, A. Krasnitz, R. L. Sugar, and D. Toussaint, "Studying quarks and gluons on mimd parallel computers," *The International Journal of Supercomputing Applications*, vol. 5, no. 4, pp. 61–70, 1991.

[86] H.-Q. Jin, M. Frumkin, and J. Yan, "The openmp implementation of nas parallel benchmarks and its performance," 1999. [Online]. Available: https://ntrs.nasa.gov/api/citations/20000102377/downloads/20000102377.pdf

[87] S. Seo, G. Jo, and J. Lee, "Performance characterization of the nas parallel benchmarks in opencl," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 137–148.

[88] F. Wong, R. Martin, R. Arpaci-Dusseau, and D. Culler, "Architectural requirements and scalability of the nas parallel benchmarks," in *SC '99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, 1999, pp. 41–41.

[89] H. Shan, F. Blagojević, S.-J. Min, P. Hargrove, H. Jin, K. Fuerlinger, A. Koniges, and N. J. Wright, "A programming model performance study using the nas parallel benchmarks," *Scientific Programming*, vol. 18, p. 715637, Jan 1900. [Online]. Available: https://doi.org/10.3233/SPR-2010-0306

[90] J. Löff, D. Griebler, G. Mencagli, G. Araujo, M. Torquati, M. Danelutto, and L. G. Fernandes, "The nas parallel benchmarks for evaluating c++ parallel programming frameworks on shared-memory architectures," *Future Generation Computer Systems*, vol. 125, pp. 743–757, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21002831

[91] A. Faraj and X. Yuan, "Communication characteristics in the NAS parallel benchmarks," in *International Conference on Parallel and Distributed Computing Systems, PDCS 2002, November 4-6, 2002, Cambridge, USA*, S. G. Akl and T. F. Gonzalez, Eds. IASTED/ACTA Press, 2002, pp. 724–729.

[92] A. Bauer, H. Pan, R. Chard, Y. Babuji, J. Bryan, D. Tiwari, I. Foster, and K. Chard, "The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud," *Future Generation Computer Systems*, vol. 153, pp. 558–574, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X23004703

[93] J. Carter, Y. He, J. Shalf, H. Shan, E. Strohmaier, and H. Wasserman, "The performance effect of multi-core on scientific applications," 5 2007. [Online]. Available: https://www.osti.gov/biblio/923361

[94] G. Bauer, S. Gottlieb, and T. Hoefler, "Performance modeling and comparative analysis of the milc lattice qcd application su3_rmd," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012, pp. 652–659.

[95] K. Antypas, J. Shalf, and H. Wasserman, "Nersc-6 workload analysis and benchmark selection process," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2008. [Online]. Available: https://www.nersc.gov/assets/pubs_presos/NERSCWorkload.pdf

[96] Y. Wang, J. D. McCalpin, J. Li, M. Cawood, J. Cazes, H. Chen, L. Koesterke, H. Liu, C.-Y. Lu, R. McLay, K. Milfield, A. Ruhela, D. Semeraro, and W. Zhang, "Application performance analysis: A report on the impact of memory bandwidth," in *High Performance Computing*, A. Bienz, M. Weiland, M. Baboulin, and C. Kruse, Eds. Cham: Springer Nature Switzerland, 2023, pp. 339–352.

[97] S. Smith, D. Lowenthal, A. Bhatele, J. Thiagarajan, P. Bremer, and Y. Livnat, "Analyzing inter-job contention in dragonfly networks," 2016. [Online]. Available: https://www2.cs.arizona.edu/~smiths949/dragonfly.pdf

[98] Y. Zhang, T. Groves, B. Cook, N. J. Wright, and A. K. Coskun, "Quantifying the impact of network congestion on application performance and network metrics," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 162–168.

[99] A. Patke, S. Jha, H. Qiu, J. Brandt, A. Gentile, J. Greenseid, Z. Kalbarczyk, and R. K. Iyer, "Delay sensitivity-driven congestion mitigation for hpc systems," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 342–353. [Online]. Available: https://doi.org/10.1145/3447818.3460362

[100] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[101] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, "Openmc: A state-of-the-art monte carlo code for research and development," *Annals of Nuclear Energy*, vol. 82, pp. 90–97, 2015, joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030645491400379X

[102] L. M. et al, "Monte carlo reactor calculation with substantially reduced number of cycles," in *Proceedings of PHYSOR 2012*, 2012.

[103] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "Funcx: A federated function serving fabric for science," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 65–76. [Online]. Available: https://doi.org/10.1145/3369583.3392683

[104] R. B. Roy, T. Patel, and D. Tiwari, "Daydream: Executing dynamic scientific workflows on serverless platforms with hot starts," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–18.

[105] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: Making serverless computing useful for hpc workflows via hybrid execution," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 46–60. [Online]. Available: https://doi.org/10.1145/3503221.3508407

[106] A. Frank, T. Süß, and A. Brinkmann, "Effects and benefits of node sharing strategies in hpc batch systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 43–53.

[107] X. Tang, H. Wang, X. Ma, N. El-Sayed, J. Zhai, W. Chen, and A. Aboulnaga, "Spread-n-share: improving application performance and cluster throughput with resource-aware job placement," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–15.

[108] J. Park, S. Park, and W. Baek, "Copart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–16.

[109] P. Ambati, I. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, M. Fontoura, and R. Bianchini, "Providing SLOs for Resource-Harvesting VMs in cloud platforms," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 735–751. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/ambati

[110] A. Fuerst, S. Novaković, I. n. Goiri, G. I. Chaudhry, P. Sharma, K. Arya, K. Broas, E. Bak, M. Iyigun, and R. Bianchini, "Memory-harvesting vms in cloud platforms," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 583–594. [Online]. Available: https://doi.org/10.1145/3503222.3507725

[111] Y. Zhang, I. n. Goiri, G. I. Chaudhry, R. Fonseca, S. Elnikety, C. Delimitrou, and R. Bianchini, "Faster and cheaper serverless computing on harvested resources," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 724–739. [Online]. Available: https://doi.org/10.1145/3477132.3483580

[112] H. Yu, H. Wang, J. Li, X. Yuan, and S.-J. Park, "Accelerating serverless computing by harvesting idle resources," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1741–1751. [Online]. Available: https://doi.org/10.1145/3485447.3511979

[113] A. Raveendran, T. Bicer, and G. Agrawal, "A framework for elastic execution of existing mpi programs," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 940–947.

[114] G. Martín, D. E. Singh, M.-C. Marinescu, and J. Carretero, "Enhancing the performance of malleable mpi applications by using performance-aware dynamic reconfiguration," *Parallel Computing*, vol. 46, pp. 60–77, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819115000642

[115] C. Huang, G. Zheng, L. Kalé, and S. Kumar, "Performance evaluation of adaptive mpi," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 12–21. [Online]. Available: https://doi.org/10.1145/1122971.1122976

[116] I. Cores, P. González, E. Jeannot, M. J. Martín, and G. Rodríguez, "An application-level solution for the dynamic reconfiguration of mpi applications," in *High Performance Computing for Computational Science – VECPAR 2016*, I. Dutra, R. Camacho, J. Barbosa, and O. Marques, Eds. Cham: Springer International Publishing, 2017, pp. 191–205.

[117] K. El Maghraoui, B. K. Szymanski, and C. Varela, "An architecture for reconfigurable iterative mpi applications in dynamic environments," in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Waśniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 258–271.

[118] K. El Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela, "Dynamic malleability in iterative mpi applications," in *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, 2007, pp. 591–598.

[119] S. Prabhakaran, M. Iqbal, S. Rinke, C. Windisch, and F. Wolf, "A batch system with fair scheduling for evolving applications," in *2014 43rd International Conference on Parallel Processing*, 2014, pp. 351–360.

[120] S. Prabhakaran, M. Neumann, S. Rinke, F. Wolf, A. Gupta, and L. V. Kale, "A batch system with efficient adaptive scheduling for malleable and evolving applications," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 429–438.

[121] E. Jonas, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," *CoRR*, vol. abs/1702.04024, 2017. [Online]. Available: http://arxiv.org/abs/1702.04024

[122] J. Thorpe, Y. Qiao, J. Eyolfson, S. Teng, G. Hu, Z. Jia, J. Wei, K. Vora, R. Netravali, M. Kim, and G. H. Xu, "Dorylus: Affordable, scalable, and accurate GNN training with distributed CPU servers and serverless threads," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021, pp. 495–514. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/thorpe

[123] G. París, P. García-López, and M. Sánchez-Artigas, "Serverless elastic exploration of unbalanced algorithms," in *2020 IEEE 13th International Conference*

*on Cloud Computing (CLOUD)*, 2020, pp. 149–157.

[124] M. K. Aguilera, E. Amaro, N. Amit, E. Hunhoff, A. Yelam, and G. Zellweger, "Memory disaggregation: Why now and what are the challenges," *SIGOPS Oper. Syst. Rev.*, vol. 57, no. 1, p. 38–46, jun 2023. [Online]. Available: https://doi.org/10.1145/3606557.3606563

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

[126] D. Álvarez, K. Sala, and V. Beltran, "nos-v: Co-executing hpc applications using system-wide task scheduling," 2022. [Online]. Available: https://arxiv.org/abs/2204.10768

[127] K. Brown and S. Matsuoka, "Co-locating graph analytics and hpc applications," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, 2017, pp. 659–660.

[128] H. Xu, S. Song, and Z. Mao, "Characterizing the performance of emerging deep learning, graph, and high performance computing workloads under interference," 2023.

[129] M. Rocklin *et al.*, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th python in science conference*, vol. 130. SciPy Austin, TX, 2015, p. 136.

[130] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *et al.*, "Ray: A distributed framework for emerging {AI} applications," in *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, 2018, pp. 561–577.

[131] A. Gueroudji, J. Bigot, and B. Raffin, "Deisa: Dask-enabled in situ analytics," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2021, pp. 11–20.

[132] W. Lu, B. Shan, E. Raut, J. Meng, M. Araya-Polo, J. Doerfert, A. M. Malik, and B. Chapman, "Towards efficient remote openmp offloading," in *OpenMP in a Modern World: From Multi-device Support to Meta Programming*, M. Klemm, B. R. de Supinski, J. Klinkenberg, and B. Neth, Eds. Cham: Springer International Publishing, 2022, pp. 17–31.

[133] A. Patel and J. Doerfert, "Remote openmp offloading," in *High Performance Computing*, A.-L. Varbanescu, A. Bhatele, P. Luszczek, and B. Marc, Eds. Cham: Springer International Publishing, 2022, pp. 315–333.

[134] L. Möller, M. Copik, A. Calotoiu, and T. Hoefler, "Cppless: Productive and performant serverless programming in c++," https://spcl.inf.ethz.ch/Publications/index.php?pub=508, 2023, accessed: 2024-01-19.

[135] T. Ben-Nun, J. de Fine Licht, A. N. Ziogas, T. Schneider, and T. Hoefler, "Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures," 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/10.1145/3295500.3356173