26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

SharP Data Constructs: Data Constructs to Enable Data-centric Computing

Ferrol Aderholdt and Manjunath Gorentla Venkata
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37830
Email: {aderholdtwf1, manjugv}@ornl.gov

Zachary W. Parchman
Computer Science Department
Tennessee Technological University
Cookeville, Tennessee 38501
Email: zwparchman42@students.tntech.edu

***Abstract*—Extreme-scale applications (i.e., *Big-Compute*) are becoming increasingly data-intensive, i.e., producing and consuming increasingly large amounts of data. The HPC systems traditionally used for these applications are now used for *Big-Data* applications such as data analytics, social network analysis, machine learning, and genomics. As a consequence of these trends, the system architecture should be flexible and data-centric. This can already be witnessed in the pre-exascale systems with TBs of on-node hierarchical and heterogeneous memories, PBs of system memory, low-latency, high-throughput networks, and many threaded cores. As such, the pre-exascale systems suit the needs of both *Big-Compute* and *Big-Data* applications. Though the system architecture is flexible enough to support both *Big-Compute* and *Big-Data*, we argue there is a software gap. Particularly, we need data-centric abstractions to leverage the full potential of the system, i.e., there is a need for native support for data resilience, the ability to express data locality and affinity, mechanisms to reduce data movement, the ability to share data, and abstractions to express User's data usage and data access patterns. In this paper, we (i) show the need for taking a holistic approach towards data-centric abstractions, (ii) show how these approaches were realized in the SHARed data-structure centric Programming abstraction (SharP) library, a data-structure centric programming abstraction, and (iii) apply these approaches to a variety of applications that demonstrate its usefulness. Particularly, we apply these approaches to QMCPack and the Graph500 benchmark and demonstrate the advantages of this approach on extreme-scale systems.**

I. Introduction

During the petascale era, the architectures of the extreme-scale systems made a transition from the use of only *Central Processing Unit*s (CPUs) to the use of both CPUs and manycore compute accelerators. This allowed researchers to significantly increase the computational performance of a system. Additionally, this extended the memory hierarchy and allowed for a heterogeneous hierarchy. As we transition to the pre-exascale era, this heterogeneous memory hierarchy is deepening as the architectures are now being composed of multiple high performing CPUs and manycore accelerators as well as non-volatile random access memory (NVRAM). Examples of this include the upcoming systems to be installed at the *Oak Ridge Leadership Computing Facility* (OLCF) and Lawerence Livermore National Laboratory (LLNL), Summit and Sierra, respectively.

With the hierarchical and heterogeneous memories in addition to the multiple CPUs and manycore compute accelerators, there are additional implementation and optimization challenges for scientific researchers. These challenges stem from the hierarchy no longer being flat, but being deep with differing latencies between memories as well as memories with affinities to specific devices [1]. While there has been much work over the years to adapt many scientific applications' algorithms to suite the addition of compute accelerators in the petascale systems [2], [3], many of these adaptations did not account for the latencies and affinities of the memories of these devices.

The adaptation of applications to best utilize these memories is imperative for applications to fully leverage the computational resources of these systems [4]. Effective adaptation of the algorithms can be accomplished by focusing on the application with a data-centric point of view. More clearly, understanding the current data layout, organization, and usage in order to provide data locality and affinity with the devices accessing the data, provide data resilience for commonly accessed data, and reduce data movement between memories on a single node and across a cluster.

The SharP library [5] was designed and developed to help accomplish these goals by creating and managing data structures including arrays and hashes on heterogeneous and hierarchical memories throughout a cluster. The creation and management of the application's data structures is completed based on *User-defined Hints* and *Constraints*, which inform SharP on how the memory is used. However, the *Hints* and *Constraints* originally defined in SharP did not holistically express an application's data-centric usage.

In this work, we define our view of data-centric computing and enhance the SharP library to be fully capable of expressing that view. We then analyze both a petascale capable application, *QMCPack*, and a petascale benchmark, the Graph500 benchmark, in terms of data locality, affinity, movement, and resilience. Through this analysis, we have identified multiple areas in which the applications can be optimized, and implemented these optimizations with the use of

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

0-7695-6380-5/18/$31.00 ©2018 IEEE
DOI 10.1109/PDP2018.2018.00031

the enhanced SharP library. To demonstrate the value of these modifications, we performed an experimental evaluation and found using a data-centric approach can significantly reduce checkpoint latency for data resilience, reduce the CPU cache miss rate, decrease the load balancing overhead in *QMCPack*, and increase the Traversed Edges Per Second (TEPS) of the Graph500 benchmark by decreasing the latency of network operations. This work makes the following contributions:

- We identify and motivate four areas of data-centric computing that are of importance as we transition to next-generation systems and use these to enhance the SharP library such that it can express these areas for use with user's data structures. Then, we analyze two petascale capable applications, *QMCPack* and the Graph500 benchmark, in these four areas.
- We adapt *QMCPack* and the Graph500 benchmark to leverage a data-centric approach by porting them to the SharP library. For *QMCPack*, we develop a lightweight and flexible *Checkpoint/Restart* (C/R) mechanism as well as provide data locality and affinity with respect to its computation and networking capabilities. For Graph500, we are able to provide data locality and affinity as well.
- We evaluate and demonstrate the utility of our modifications to both *QMCPack* and the Graph500 benchmark. In our evaluation, we found that *QMCPack* with our modifications showed up to a 72% reduction in time required for the load balancing Walkers and up to a 9.9% improvement in performance for the Graph500 benchmark.

II. Data-centric Taxonomy

In this section, we will detail various, well-known aspects of data-centric computing that we see as important concepts that should be easily enabled for *Users* of modern and future extreme-scale systems. These aspects include (i) data resilience, (ii) data locality, affinity, and movement, (iii) data sharing, and (iv) abstractions to capture *User* intent.

A. Data Resilience

As systems move toward increasingly larger scales, the amount of data used for computation is increasing. This increase in size allows for many potential failures or faults that may affect an application's data structures. Examples of these failures include catastrophic failures, transient failures, and silent data corruption [6]. This increases the need for providing resilience for the application's data structures.

B. Data Locality, Affinity, and Movement

Providing data locality has long been a technique to increase performance by keeping the data used by the application near the computation. With the addition of the hierarchical and heterogeneous memories with differing latencies between memories and multiple devices that can perform computation (i.e., CPUs and accelerators), providing data locality to multiple computing devices will be challenging. In addition, there is also a need to provide locality to devices such as the NIC [1], [7], which may be near a subset of the node's computing devices. Providing this locality often results in an increase in communication performance.

C. Data Sharing

The convergence of *Big-Data* and *Big-Compute* coupled with the growing data size of applications has made data-sharing an important topic. Data sharing can exist in two forms: (i) between processes of a single application and (ii) between processes of multiple applications.

For (i), sharing data between processes within the same application is a commonly used technique to reduce the amount of duplicated data existing in the global state of an application. On a single node, this can be accomplished through the use of shared-memory or other constructs that would allow for multiple processes or threads to access the data local to the node concurrently. For sharing remote data between multiple nodes executing the same application, globally shared memory can be used and was demonstrated in [8]. Globally shared memory allows for the data elements to span evenly across nodes within a system while allowing for used elements of data to be cached local to the Processing Element (PE) using the elements. This allows for the consumption of fewer resources.

In (ii), the amount of data generated by exascale applications will likely become exceedingly large. To provide the proper analytics necessary to process the data, converging *Big-Data* with *Big-Compute* is one methodology to keep the data to be studied local and perform in-situ techniques to provide data analytics or visualization without moving the data.

D. Intent-based Data Allocation

Another aspect of providing a data-centric approach for applications and libraries on extreme-scale systems is the thought process of the developer of high-performance scientific and analytic applications in *Big-Compute* and *Big-Data* environments. One method to provide this is to have the user determine the proper data structure and its intended usage with respect to the potential memories used. The user can define these through *usage hints*, which implicitly define how a user will make use of their data structure. Along with *usage hints*, the user can also define *access hints* to specify how the data will be accessed. Both of these hints will allow a data-centric library to optimize a data structure's organization and access.

III. The SharP Programming Abstraction

The SharP programming abstraction provides a solution for researchers with respect to data-centric software abstractions [5]. The contributions SharP provides as a linkable library include (i) abstractions of the hierarchical and heterogeneous memories within a node as well as across a cluster, (ii) a simple, intuitive interface for the creation, modification, and management of distributed data structures, and (iii) an implementation that provides interoperability with the many popular programming models currently used in scientific applications (e.g., MPI, OpenSHMEM, etc.) through a communication layer that can leverage MPI, Unified Communication

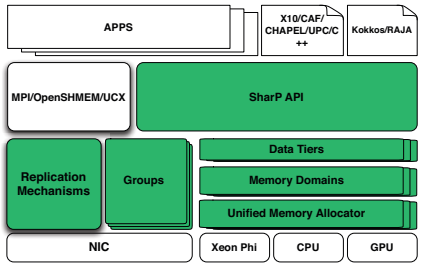


Fig. 1. The components of the SharP Library. [5]

X (UCX) [9], and OpenSHMEM as communication conduit. UCX is the default conduit. The usage of UCX allows SharP to provide high-performance, one-sided communication for data manipulation of SharP managed data structures. Meanwhile, optimized collective communication and point-to-point communication are handled through other communication libraries such as MPI. Thus, MPI+SharP and OpenSHMEM+SharP are both possible, which limits the porting effort of existing applications. The various components composing the SharP library can be seen in Figure 1.

In future exascale systems, it is anticipated that many hierarchical and heterogeneous memories will be used including DRAM, High-bandwidth Memory (HBM) (i.e., GPU HBM or Intel KNL HBM), and NVRAM. To remove the expert knowledge requirement to use these memories, SharP is composed of many data constructs including Memory Domains, Data Tiers, and a Unified Memory Allocator (UMA). Memory Domains are a representation of the physical memories present in the system, while Data Tiers are logical abstractions of the Memory Domains based on *User Hints* and *Constraints*, which define how the allocation of memory will be used by the application. The resulting Data Tier would then be used by the UMA to collectively allocate memory on a particular memory or set of memories throughout a cluster. For example, the user may wish to allocate memory on a NUMA node with an affinity to an accelerator to reduce the distance between the memory and the accelerator, thus improving the latency of the data movement between the CPU and the accelerator.

After memory is allocated as data objects, data layout mappings may be used to create data structures (i.e., arrays and hashes) from the allocated data objects. These data structures are flexible and allow for users to easily manipulate the data layout and organization to optimize performance for the application or allow for a simple transition from computation to analysis (e.g., in-situ visualization). The mappings include a uniform mapping, which uniformly distributes the data among all of the PEs within a network group (e.g., MPI Communicator and OpenSHMEM Active Sets), and a custom mapping, which allows the user to specify the data layout (e.g., strided chunking). As an example, the implicit tiling and block-cyclic data layouts that are used by [10] and [11] can be explicitly defined through the custom mappings.

IV. SharP Support for Data-Centric Approaches

Currently, the SharP library provides varying levels of support for the aspects of data-centric computing detailed in Section II. In this section, we will detail the support and where the support is lacking for (i) data resilience, (ii) data locality, affinity, and movement, (iii) data sharing, and (iv) intent-based data allocations.

The SharP library natively enables data resilience. This support is provided by the allocations of data objects and distributed data structure interfaces attached to these data objects, which keeps memory in contiguous buffers. Overall, this allows applications using SharP to (i) reduce checkpoint latency through fine-grained or coarse-grained checkpoints, (ii) provide sequential and non-sequential checkpoints, and (iii) support coordinated and uncoordinated checkpoints. However, this requires the *User* to use NVRAM rather than any stable storage through the persistent *usage hint* or *constraint*. This limits the usability of the approach as not all systems will be equipped with NVRAM.

The SharP library has many enabling data constructs for data locality and affinity, as well as reducing data movement between devices. These constructs were explained in detail and demonstrated in [5], and included concepts such as explicitly and implicitly allocating memory near devices. However, while a *User* could allocate memory explicitly near a NIC, the *User* was not capable of allocating memory implicitly, which limited the performance portability of the approach and required users to determine the hardware architecture of their cluster to improve communication performance.

SharP is capable of natively sharing data between processes of the same application (i.e., (i) from II-C) through the creation and management of global and local data structures. This was demonstrated in [5]; however, SharP's capability for sharing data between applications was not clearly defined and requires users to implement this feature themselves.

For SharP, user intent was captured through *usage hints* and *Constraints*. These were flexible enough to allow the user to easily define how memory allocations should occur across multiple PEs within a cluster, which allowed for portability of the application between different computing architectures. However, the *Hints* and *Constraints* were limited to the use of a data structure for computation and not how the data structure will be accessed by other PEs within the same application or in another application. By providing these access *Hints* and *Constraints*, the user would be able to more clearly define a data structures usage and how the data structure should be allocated.

V. Extensions to SharP's Data Constructs

To best provide the support necessary to enable the many data-centric approaches that have been discussed thus far, extensions to SharP's *User Hints* and *Constraints* can be made to provide a more thorough mapping between the *User*'s intent and the physical memories present in the system. These extensions include information on how the *User*'s data will be accessed by the application. More specifically, we can

extend the *Hints* and *Constraints* to provide *Access Hints* and *Constraints*.

Access hints can be used by the *User* to provide SharP with information concerning data affinity to particular devices, which will provide data locality and reduce data movement. Such hints can be specified with:

- `SHARP_HINT_LATENCY_OPT`: When this access hint is provided SharP will provide a best-effort approach to identify and construct a mapping between the *User*'s data structure and memory with an affinity to the NIC or fabric. It will accomplish this by attempting to maximize both the compute and latency performance characteristics by utilizing a matrix of relative distances between devices and finding a NIC with an affinity to the calling PE's processing location (i.e., NUMA node) and the NIC.
- `SHARP_HINT_COMPUTE_OPT`: This usage hint is used to suggest to SharP that the memory allocated should be as close as possible to the calling PE. This will result in a memory allocation that is also bound to that memory.

In addition to access hints, access constraints can be used to better specify how data will be used by the application and require SharP to provide memory allocations enabling this usage. The added constraints can be seen below:

- `SHARP_ACCESS_INTRAP`: Requires SharP to allocate memory for a data object such that it is at least accessible to threads of the calling process.
- `SHARP_ACCESS_INTERP`: Requires SharP to allocate memory that is accessible between processes within the same job through RDMA operations.
- `SHARP_ACCESS_INTERJ`: Requires SharP to allocate memory that is accessible between processes of multiple jobs through RDMA operations or file operations.

With these new *access Hints* and *Constraints*, SharP will be capable of enabling the features that were lacking in Section IV. For data resilience, applications can now implicitly make use of the parallel file-system instead of only NVRAM through the combination of the *Hints* and *Constraints*: `SHARP_CONSTRAINT_PERSISTENT` and `SHARP_ACCESS_INTERJ`, which will allocate a persistent data object accessible across jobs. In addition, the same access constraint may be used to enable data sharing between applications. For Data locality and affinity, support for communication-based locality is provided through the `SHARP_HINT_LATENCY_OPT` *Hint*.

VI. Data-centric Analysis of Applications

In this section, we will analyze applications and benchmarks such as *QMCPack* and the Graph500 benchmark regarding the topics listed in Section II. This will begin with an assessment of their current implementation followed by a description of the modifications we performed enable a data-centric approach with SharP.

A. Analysis of QMCPack

For the analysis of *QMCPack*, we will focus on the following areas: data resilience, locality, affinity, and movement. In each area, the focus will be narrowed to the Variational Monte Carlo (VMC) and Diffusion Monte Carlo (DMC) methods supported by *QMCPack*. The analysis will be followed by a description of the modifications made to *QMCPack*.

1) Data Resilience for QMCPack: The *QMCPack* implementation executes iteratively with multiple blocks. At the end of each block, all of the PEs will synchronize with a data exchange and perform a checkpoint based on the user-defined checkpoint frequency. *QMCPack* will use either the Adaptable IO System (ADIOS) library [12] or the HDF5 library to save the current configuration and Walker data to stable storage. We will focus on the ADIOS library as it has been shown to be more performant [13]. The checkpointing approach is single-level, coordinated, and sequential, which requires synchronization. The use of coordinated and sequential checkpointing may lead to significant checkpoint latency. Additionally, the single-level checkpointing approach will likely suffer from performance issues as the scale increases into the exascale era [14].

Focusing on the single-level C/R approach, a reduction of checkpoint latency and overhead can be accomplished by keeping data structures in a state ready for checkpointing without data packing. Additionally, taking advantage of data affinity to memories and devices would allow for checkpoints to move efficiently to stable storage and provide capabilities that would allow for uncoordinated checkpoints, which may reduce the checkpoint latency.

2) Data Locality/Affinity for QMCPack: The *QMCPack* implementation has two primary areas in which data locality is necessary. These areas include, (i), the abstractions used for the underlying implementation and, (ii), the methodologies used to load balance Walkers between PEs during each branching step of the DMC algorithm.

For (i), the implementation of *QMCPack* contains many dynamic, discontiguous data structures, which allow the application to support an unbounded amount of Walkers and data sets. The use of these data structures does not allow for a simple use of heterogeneous and hierarchical memories and will lack locality to the devices using them. Additionally, during the load balancing phase, Walker data is required to be packed and unpacked, which can diminish performance.

In (ii), the memory used by the communication library to exchange Walkers during the load balancing phase may not be allocated near the NIC. Because of this, a data movement penalty could be seen when exchanging Walkers between PEs.

By moving the Walker's data from discontiguous data structures to contiguous data structures, the Walker would no longer need to be packed or unpacked during an exchange. Additionally, the control plane (i.e., notification of message delivery) of the swapping of Walkers is completed by MPI, which may not be performing its communication with data near the NIC. Instead, adapting the control plane to use data near the NIC will decrease the time between the PE acknowledging the arrival of data and continuing to the next computational step.

3) Data Movement for QMCPack: Data movement can be viewed as local and remote. Local data movement refers to the movement of data from main memory to the CPU. Remote data movement refers to the movement of data over the network.

For the data movement between the local memory and CPU, there is currently a considerable amount of movement. This is because all of the information stored by *QMCPack* is stored within discontiguous data objects as mentioned in Section VI-A2. By placing the data objects in memory discontiguously, the implementation is removing potential spatial locality benefits over time.

Data movement between PEs occurs within the DMC method. This data movement is realized in the branching step of the DMC algorithm. During this step, the *QMCPack* implementation performs a packing operation when sending a Walker. The receiver will then unpack the Walker into a new buffer and add this Walker to its list of Walkers.

4) Modifications made to QMCPack: The first modification completed was moving the important data structures of *QMCPack* to SharP. Placing the data structures in SharP allows the state of the data structures to be globally and locally shared. Additionally, *usage* and *access hints* can be used to provide data locality and affinity. This enables the modifications described in Section VI-A1, VI-A2, and VI-A3.

For data resilience, we modified the C/R functionality of *QMCPack*. The approach we utilized retains the semantics established in *QMCPack*, however, our approach allows for flexibile checkpointing. More clearly, our approach does not keep us from also applying techniques such data deduplication, compression, migration of data between PEs, or multi-level checkpointing. Additionally, our approach allows for data locality between devices and memories.

Our C/R mechanism employs a sequential checkpointing approach utilizing two globally shared arrays. Both arrays make use of *access hints* to enable persistence, which will have the arrays allocated on some stable storage including the parallel file-system, NVRAM, etc. The first array contains all of the Walkers for each PE. The second array contains relevant information to the simulation including trial energy, reference variance, etc. Because the checkpoint state is already resident in SharP arrays, the implementation of the C/R mechanism's functionality becomes a simple copy of data between data structures. This allows the approach to be uncoordinated and extensible to further enable policy-driven modifications such as multi-level checkpointing [15] because the checkpoint data is not abstracted away from the user.

For data locality, affinity, and movement, *QMCPack* was modified with respect to two aspects: (i) contiguous data storage and (ii) the control plane and mechanisms for communication when swapping Walkers between PEs.

In (i), the use of discontiguous storage has been replaced with contiguous allocations. The transition between the data structures required modification to the data objects of the Walkers that are commonly used for computation. This modification allows for increased data locality for computations, provides for decreased data movement between memories and caches, and allows for Walkers to be moved between PEs without packing and unpacking the data.

In (ii), when changing the control plane for swapping Walkers, we also modified how the Walkers are swapped. Because SharP uses one-sided operations, we modified the swapping of Walkers to be one-sided while retaining the original semantics of the two-sided approach for the sake of comparison (i.e., emulated two-sided communication with one-sided communication). Thus, semantically, only the message delivery notification of the swapped Walker is different from the original *QMCPack* version. In our approach, we used the *access hints* of SharP to place memory near the NIC to leverage data affinity and reduce data movement.

B. Graph500

Because Graph500 is a benchmark, we will only discuss the topics of data locality, affinity, and movement.

1) Data Locality/Affinity for Graph500: The Graph500 benchmark performs 64 BFS operations originating from random vertices within a generated graph. The MPI one-sided communication port of the benchmark uses a predecessor map to determine previously visited nodes within the graph during each BFS operation. The data structure is allocated near each PE prior to initiating the BFS operation and is shared globally. Within each BFS operation, a copy of the predecessor map and two bitmap queues are allocated and also shared. With these data structures, one map and queue are used for reading while the others are used for writing. This is done due to the use of MPI-2 one-sided semantics, which uses a fence operation to begin modifications to a shared data structure and another fence to guarantee completion of the modifications.

In each iteration of the BFS, a series of MPI Accumulate operations are completed. Whether or not the operation is required is not known because the operations are not completed until after the final fence operation. Because the data structure is primarily used for writing using network operations, placing these data elements near the NIC may provide increased performance through decreased latency.

2) Data Movement for Graph500: Currently, the benchmark has all of its data structures in memory near the PE. Because the data is located near the PE and not necessarily near the NIC, the distance the data is forced to move is large. Placing data near the NIC will reduce data movement for each BFS operation.

3) Modifications to Graph500: The first modification made during the adaptation of the benchmark was to move the important data structures to the SharP library. These data structures include the predecessor maps and queue bitmaps. To obtain data locality and affinity and reduce data movement, *usage* and *access hints* are used.

To obtain similar semantics to the one-sided MPI-2 port, modifications were made to the BFS implementation. These modifications replaced the MPI Accumulate operations with *Atomic Memory Operations* (AMO) operations similar to those

discussed in [16]. In [16], the authors replaced MPI Accumulate operations with AMO operations that would first retrieve the current value from the map or bitmap being modified and determine if the modification should take place. This was accomplished with a `fetch-and-add` operation, which we replaced with a *Get* operation. This improved performance of the implementation and retained correctness.

VII. Experimental Evaluation

This Section will present an experimental evaluation of the data-centric modifications enabled by SharP of both *QMCPack* and the Graph500 benchmark and compare these versions with their vanilla counter parts. The testbed used to perform these evaluations consisted of the Titan system located at Oak Ridge National Laboratory (ORNL). The Titan system consists of 18,688 nodes each equipped with a single NVIDIA Tesla K20 GPU, 32 GB of memory, and the Cray Gemini interconnect.

A. QMCPack

To evaluate the data-centric adaptation and the vanilla *QMCPack*, we used three different solid structures depending on the experiment being conducted. These solids include:

1) Body-centered cubic Hydrogen (bccH) 1x1x1 supercell. This is a very small and simple solid requiring minimal computation in both the VMC and DMC methods. Each Walker is near 1 KB in size.
2) Diamond 2x1x1 supercell. This solid requires more computation than the bccH solid, but is a light workload. Each Walker is roughly 20 KB in size.
3) Graphite 4x2x2 supercell. This solid requires a significant amount of computation per Walker. Each Walker is roughly 16 MB in size.

1) Checkpoint/Restart: For an evaluation of data resilience with *QMCPack*, we will perform an evaluation with two compute sizes resulting in small and large checkpoint sizes. The first workload will consist of the Diamond supercell and the second workload will be the Graphite supercell. This will give us an understanding of the behavior of the C/R mechanisms with differing workloads. For the Diamond supercell experiment, we used 8 Walkers per PE with 8 PEs per node. For the Graphite supercell experiment, we used 4 Walkers per PE and 4 PEs per node. The results of this experiment are shown in Figure 2.

As expected, there is a considerable difference in checkpoint latency between ADIOS and the data-centric version. This is due to how the checkpoint is stored and the coordination between PEs. For ADIOS, the checkpoint is stored in a single, shared file on the Lustre filesystem, while the data-centric version used a single file per PE, which does not require coordination between PEs. Thus, the checkpoints could be completed asynchronously with PE synchronization occurring later in execution. For the Diamond supercell experiment, the data-centric version's checkpoint latency is less than 1% of the latency of ADIOS's with 4 nodes (i.e., 32 PEs) and continues to be less than 1% regardless of the scaling up to 128 nodes

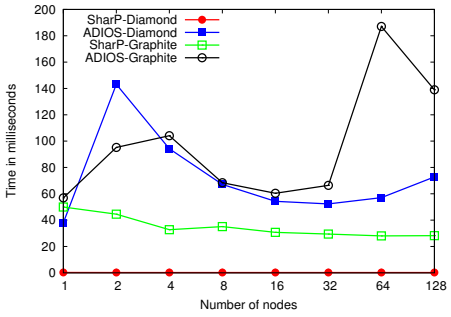


Fig. 2. Average checkpoint latency of *QMCPack* using either ADIOS or SharP to perform checkpoints. This includes both the Diamond and Graphite supercell experiments. Lower is better.

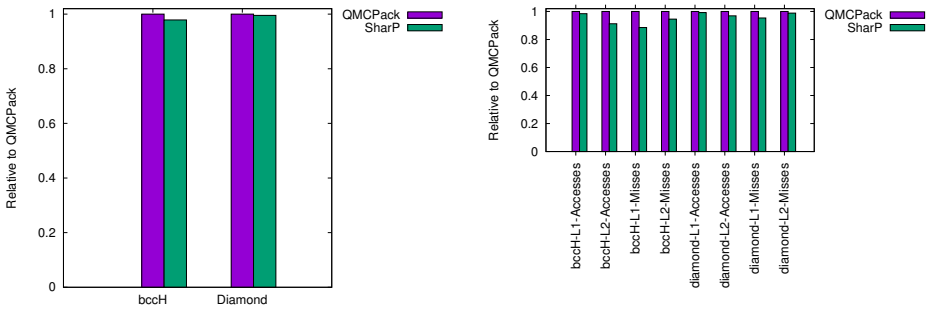
(i.e., 1024 PEs). For the Graphite supercell experiment, the adapted version also outperforms ADIOS by up to 6X.

2) Data Locality, Affinity, and Movement: Two modifications were made to *QMCPack* to provide data locality and affinity: (i) the Walker data structures were modified to be contiguous and (ii) the control plane's memory was allocated near the NIC. Both modifications were previously discussed in Section VI-A4.

In (i), we evaluated our modifications by exploring the execution timings of the VMC method. Additionally, we measured the CPU's L1 and L2 cache to understand the affect on data movement and locality. For this experiment, we used the bccH and Diamond supercells due to their size. We excluded the Graphite supercell because the size of the Walkers is larger than the caches on Titan. The results of these experiments can be seen in Figure 3(a) and 3(b).

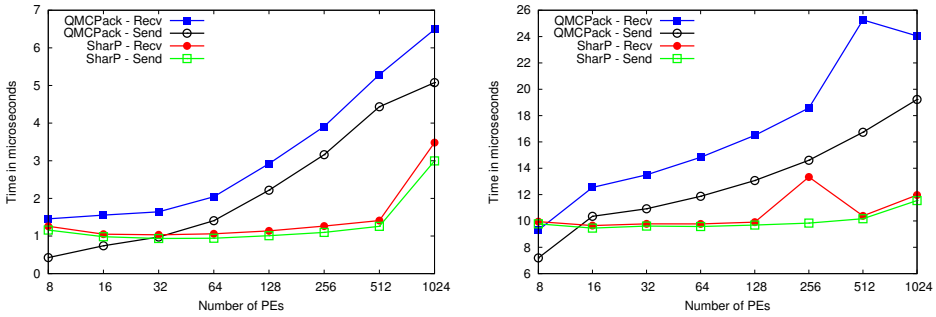
For the VMC execution timings, we found that the use of contiguous data structures improves the overall performance of small workloads such as bccH by up to 2.1%. For a Diamond run, which is a larger workload than bccH, we saw a negligible increase in performance of 0.5%. The reasons for these performance differences can be seen in Figure 3(b). The data-centric adaptation lowered the number of accesses to the L1 and L2 cache for the bccH data set by 1.6% and 8.8%, respectively. This suggests there is less data movement between the CPU and memory. Further, the L1 and L2 miss rates were lowered by as much as 11.5%. The Diamond experiment had similar results. With the Diamond experiment, the cache miss reduction within the L1 cache was up to 4.7% lower and 1.2% lower with the L2 cache.

For (ii), we used both the bccH and Diamond supercells for the evaluation. The results of the experiments can be seen in Figure 4 where Figure 4(a) and 4(b) are the average time taken to send and receive a Walker. The results for both the bccH and Diamond experiments were similar. In both, the data-centric approach outperformed the vanilla approach when sending and receiving Walkers, which was expected. When sending Walkers in the bccH experiment, the time to send a Walker was reduced by up to 72% with 512 PEs. In the Diamond experiment, the time to send a Walker was reduced by up to 40% with 1024 PEs. Similarly, the time to receive a



(a) (b)

Fig. 3. The VMC execution time (a) and CPU cache (b) results of the vanilla QMCPack and the data-centric QMCPack.



(a) (b)

Fig. 4. Average DMC send and receive times of Walkers for (a) bccH and (b) Diamond tests.

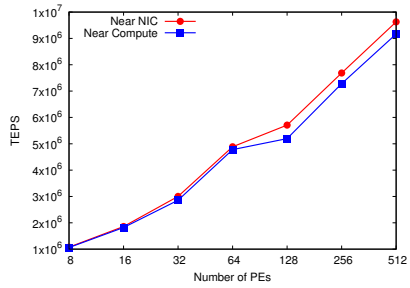


Fig. 5. The Graph500 benchmark with memory allocated near the NIC and near compute. The benchmark is weakly scaled with an initial scale factor of 15. Higher is better.

Walker was reduced by up to 73% for the bccH experiment and 59% for the Diamond experiment.

B. The Graph500 Benchmark

To understand the utility of providing both data locality and affinity to the NIC for the Graph500 benchmark, we used our adapted Graph500 benchmark and modified the *Hints* given to SharP to produce two different implementations: one with memory allocated near the NIC and one with memory allocated near the compute (i.e., PE), which is the default implementation for the Graph500 benchmark. These implementations were weakly scaled with a initial scale of 15. For this experiment, we used 8 PEs per node.

As expected, the implementation with memory near the NIC outperformed the default allocation location as the graph size was increased. While the performance is similar for less than 128 PEs (i.e., roughly 1.4%), the difference increases to up to 9.9%. This suggests that data-intensive applications would benefit from this type of approach.

VIII. Related Work

This work touches on many aspects of data-centric computing including topics such as providing data-resilience, data-locality, and data-affinity as well as reducing data-movement through abstractions of User intent. There have been many prior works on locality and affinity also touching on either (i) the importance of providing locality with current architectures or (ii) automatic provisioning of data locality based on software hints.

In (i), there are multiple works by Moreaud et al. [1], [7] and Goglin et al. [17]. In [1], [7], and [17], the authors discuss the importance of data locality to the NIC. More specifically, the authors show a roughly 15% improvement in performance when providing locality to the NIC in NUMA systems. Much of this work was used to motivate and develop the `hwloc` library [18], which is commonly used to obtain locality information between devices in an individual system. Each of these works serve as motivation for our work.

For (ii), works providing automatic provisioning of data locality, there are multiple works including DASH [10], TiDA [11], and Kokkos [19], which provide some form of data locality for applications. DASH is a C++, MPI-based PGAS abstraction implemented with templates executing on

the DASH Runtime (DART). This work gives array data structures a global view while providing locality for elements of the array. This work was furthered in [20] and [21] where the user can specify hints on how the array will be accessed to assist in the locality of the data placement. TiDA is a similar work to DASH in that it provides data locality for array elements based on how the array is used. However, unlike DASH, the locality also takes into account locality between NUMA nodes.

Unlike these works, this work takes a holistic approach to enable many aspects of data-centric computing including data resilience, locality, affinity, movement, and the abstraction of User's intent. Likewise, the discussions in this work do not relate to a single type of data structure (i.e., an array), but to many possible data structures and their usage.

IX. Conclusion

As we move to the pre-exascale systems and eventually reach exascale systems, the architectures of extreme-scale systems will be composed of many high-performing computing devices, a high-performance NIC, and hierarchical and heterogeneous memories. With these systems, it is important to make use of a data-centric approach to fully take advantage of the future systems.

In this paper, we have enhanced the SharP library to support a wider range of capabilities to enable data-centric computing. Additionally, we have explored, discussed, and demonstrated the importance of using a data-centric approach for the development of scientific applications. The particular topics of a data-centric approach that were explored and discussed include resilience in Section II-A, locality, affinity, and movement in Section II-B, sharing in Section II-C, and abstractions capturing User intent in Section II-D.

We analyzed and adapted two petascale capable applications, *QMCPack* and the Graph500 benchmark, to make use of a data-centric approach in Section VI. We evaluated these adaptations and found significant improvements for *QMCPack* with respect to data resilience with checkpoint latencies decreased by as much as 99%. With respect to data locality and affinity, we increased cache usage and decreased network latency by as much as 73%. For the Graph500 adaptation, we had similar results with data locality and affinity to the NIC providing as much as 9.9% performance improvement when compared to locality to the PE.

References

[1] S. Moreaud, B. Goglin, and R. Namyst, *Adaptive MPI Multirail Tuning for Non-uniform Input/Output Access*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 239–248.

[2] K. Esler, J. Kim, D. Ceperley, and L. Shulenburger, "Accelerating quantum monte carlo simulations of real materials on gpu clusters," *Computing in Science Engineering*, vol. 14, no. 1, pp. 40–51, Jan 2012.

[3] F. Wang, C.-Q. Yang, Y.-F. Du, J. Chen, H.-Z. Yi, and W.-X. Xu, "Optimizing linpack benchmark on gpu-accelerated petascale supercomputer," *Journal of Computer Science and Technology*, vol. 26, no. 5, p. 854, 2011.

[4] A. Tate *et al.*, "Programming Abstractions for Data Locality," PADAL Workshop 2014, April 28–29, Swiss National Supercomputing Center (CSCS), Lugano, Switzerland, Research Report, Nov. 2014. [Online]. Available: https://hal.inria.fr/hal-01083080

[5] M. G. Venkata, F. Aderholdt, and Z. W. Parchman, "Sharp: Towards programming extreme-scale systems with hierarchical and heterogeneous memory," in *Proceedings of the 6th International Workshop on Heterogeneous and Unconventional Cluster Architectures and Applications*, Aug 2017.

[6] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *The International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.

[7] S. Moreaud and B. Goglin, "Impact of numa effects on high-speed networking with multi-opteron machines," in *Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems*, ser. PDCS '07. Anaheim, CA, USA: ACTA Press, 2007, pp. 24–29. [Online]. Available: http://dl.acm.org/citation.cfm?id=1647539.1647545

[8] Q. Niu, J. Dinan, S. Tirukkovalur, L. Mitas, L. Wagner, and P. Sadayappan, "A global address space approach to automated data management for parallel quantum monte carlo applications," in *2012 19th International Conference on High Performance Computing*, Dec 2012, pp. 1–10.

[9] ORNL, "UCX: Unified Communication X," http://www.openucx.org, 2015.

[10] K. Fürlinger, C. Glass, J. Gracia, A. Knüpfer, J. Tao, D. Hünich, K. Idrees, M. Maiterth, Y. Mhedheb, and H. Zhou, *DASH: Data Structures and Algorithms with Support for Hierarchical Locality*. Cham: Springer International Publishing, 2014, pp. 542–552.

[11] D. Unat, T. Nguyen, W. Zhang, M. N. Farooqi, B. Bastem, G. Michelogiannakis, A. Almgren, and J. Shalf, *TiDA: High-Level Programming Abstractions for Data Locality Management*. Cham: Springer International Publishing, 2016, pp. 116–135.

[12] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible io and integration for scientific codes through the adaptable io system (adios)," in *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, ser. CLADE '08. New York, NY, USA: ACM, 2008, pp. 15–24.

[13] S. Herbein, M. Matheny, M. Wezowicz, J. Krogel, J. Logan, J. Kim, S. Klasky, and M. Taufer, "Performance impact of i/o on qmcpack simulations at the petascale and beyond," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, Dec 2013, pp. 92–99.

[14] F. Cappello, G. Al, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomput. Front. Innov.: Int. J.*, vol. 1, no. 1, pp. 5–28, Apr. 2014. [Online]. Available: http://dx.doi.org/10.14529/jsfi140101

[15] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11.

[16] E. F. D'Azevedo and N. Imam, *Graph 500 in OpenSHMEM*. Cham: Springer International Publishing, 2015, pp. 154–163.

[17] B. Goglin and S. Moreaud, "Dodging non-uniform i/o access in hierarchical collective operations for multicore clusters," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, May 2011, pp. 788–794.

[18] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Feb 2010, pp. 180–186.

[19] Edwards, H. Carter and Sunderland, Daniel, "Kokkos array performance-portable manycore programming model," in *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*, ser. PMAM '12. New York, NY, USA: ACM, 2012, pp. 1–10.

[20] T. Fuchs and K. Fürlinger, *Expressing and Exploiting Multi-Dimensional Locality in DASH*. Cham: Springer International Publishing, 2016, pp. 341–359.

[21] T. Fuchs and K. Frlinger, "A multi-dimensional distributed array abstraction for pgas," in *2016 IEEE 18th International Conference on High Performance Computing and Communications (HPCC 2016)*, Dec 2016, pp. 1061–1068.