Towards a Scalable and Distributed High-performance SHAD C++ Library

Nanmiao Wu [⋆,†] wnanmi1@lsu.edu
Vito Giovanni Castellana [⋆] vitogiovanni.castellana@pnnl.gov
Hartmut Kaiser [†] hkaiser@cct.lsu.edu

[⋆]Pacific Northwest National Laboratory
Richland, WA, USA
[†]Center for Computation & Technology, Louisiana State University
Baton Rouge, LA, USA

ABSTRACT

SHAD is the Scalable High-performance Algorithms and Data-structures C++ library, providing general purpose building blocks and supporting high-level custom utilities. SHAD is designed with scalability, flexibility, productivity, and portability in mind, and serves as a playground for research in parallel programming models, runtime systems, and, their applications. SHAD's portability is achieved through the Abstract Runtime Interface, which decouples the upper layers of the library and hides the low level details of the underlying architecture. This layer enables SHAD to support by different platforms, via different runtimes, e.g., Intel TBB and global memory and threading (GMT). However, current backends targeting distributed systems, rely on a centralized controller which limits scalability up to hundreds of nodes and creates a network hot spot due to all to one communication for synchronization, resulting in degraded performance at high process counts. In this research, we explore the scalability and performance of SHAD when interfacing with HPX, the C++ standard library for parallelism and concurrency, as the underlying backend.

INTRODUCTION

Recent Big Data and Exascale system architectures have presented challenges to both industry and the research community, expecting software solutions to be scalable on such complex systems. SHAD, a high-performance algorithm and data-structure library, providing scalability, flexibility, productivity, and portability, is designed to address these challenges [1]. SHAD consists of three layers. The upper layer, called SHAD extensions, contains high-level libraries obtained by composing data structures and/or other extensions, e.g., graph library. The middle layer provides general purpose data structures, e.g., Array, Vector, Set, and Map. The bottom layer is an Abstract Runtime Interface, which abstracts underlying hardware systems and manages remote procedures execution and data movements.

In this work, we first describe the design of Abstract Runtime Interface of SHAD and then employ HPX, a C++ standard library for concurrency and parallelism as the backend [2].

2. METHODOLOGY

In this section, we first present the Abstract Runtime Interface in Sect. 2.1, then describe HPX in Sect. 2.2, and demonstrate how HPX supports remote calls for SHAD and the benefits of using HPX in Sect.2.3.

2.1 ABSTRACT RUNTIME INTERFACE

The Abstract Runtime Interface relies on three main concepts: locality, task, and handles. A locality is denoted as an entity wherein the memory is directly accessible. A locality can be a node in a cluster, a core, or a NUMA domain. A task is the basic unit of computation, which can be executed synchronously or asynchronously on any locality. Handles are identifiers for spawning tasks and can be used to check the completion of a task. Note that multiple tasks can be associated to the same handle, forming a task group.

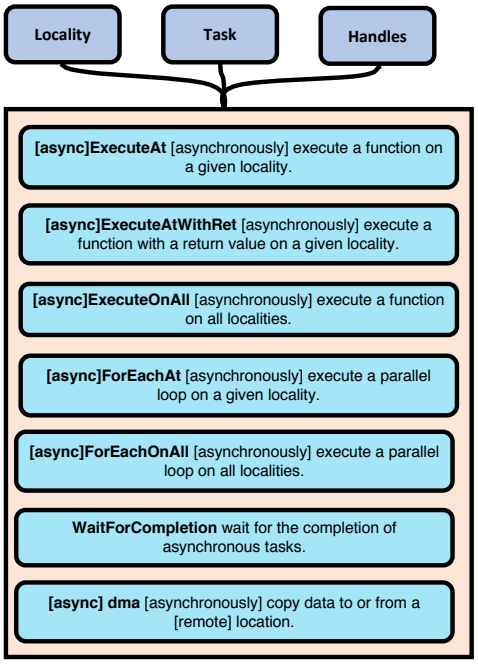

Figure 1. Main methods offered by abstract runtime interface.

The Abstract Runtime Interface offers several methods for (asynchronous) task execution, including methods to check for the completion of asynchronous tasks, as shown in Figure 1.

2.2 HPX

HPX is a C++ standard library for concurrency and parallelism, with the following major components.

- A threading system which manages the lightweight threads and ensures working-stealing and work-sharing policies.
- A parcel transport system, which implements active messages that are used to encapsulate remote calls.
- An active global address space (AGAS), which spans a global address space over all localities.
- Local control objects, which support controlling parallelization, synchronization and hide latency.
- A performance counter which provides a means of monitoring system metrics.

2.3 SHAD + HPX

We investigate the scalability and performance of SHAD when mapping its runtime interface to HPX, as shown in Figure 2. If an action or a remote procedure is called on a global object of SHAD, AGAS would ensure that the global object could be transparently migrated from one locality to another locality. Specially, AGAS would resolve the address and deliver the active message to the responding locality and enqueue it in the locality's thread scheduler.

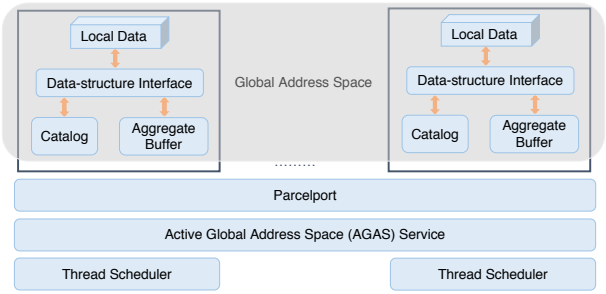

Figure 2. SHAD + HPX.

When interfacing with HPX, SHAD benefits from fine-grained parallelism, message driven computation, constraint-based synchronization, overlapping computation and communication, and minimal overheads from the lightweight threading system.

EVALUATION

In this section, we evaluate the scalability and performance of the proposed design. The experiments have been run on a cluster where each node is equipped with two Intel Xeon E5-2680 v2 CPUs working at 2.8 GHz and 768GB of memory per node. Each socket of a compute node is used as one locality. We have conducted the experiment using SHAD STL algorithms with the SHAD Array data structure.

Performance on one locality: We first test the execution time of each STL algorithms varying the size of the data from 1 billion to 10 billion, and finally to 100 billion elements.

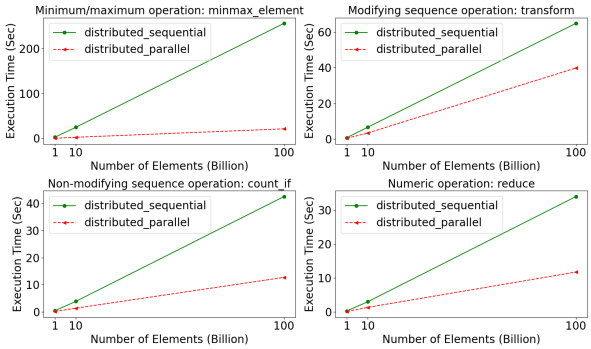

Figure 3. SHAD STL algorithms performance using SHAD Array.

We observed a linear increasing execution time, as shown in Figure 3. Note that we choose one algorithm from the each category of STL algorithms for the sake of simplicity.

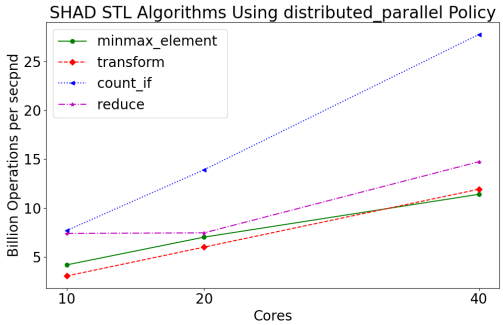

Figure 4. Weak scaling using SHAD array for STL algorithms when inserting 1 billion elements per locality.

Performance on multiple localities: We also run the experiments on multiple localities and increase the cores from 10 to 40. Figure 4 illustrates the scalability and performance of SHAD array and SHAD STL algorithms. The experimental results show that the SHAD STL algorithms have almost linearly weak scaling performance. Note that for the reduce algorithm we did not observe speedup when increasing the core count from 10 to 20, which corresponds to crossing the NUMA boundaries.

REFERENCES

[1] Vito Giovanni Castellana and Marco Minutoli. 2018. SHAD: The Scalable High-Performance Algorithms and Data-Structures Library. In *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018, Washington, DC, USA, May 1-4, 2018*. 442–451.

[2] Hartmut Kaiser, Patrick Diehl, Adrian S Lemoine, Bryce Adelstein Lelbach, Parsa Amini, Agustín Berge, John Biddiscombe, Steven R Brandt, Nikunj Gupta, Thomas Heller, and others. 2020. HPX - The C++ Standard Library for Parallelism and Concurrency. *Journal of Open Source Software* 5, 53 (2020).