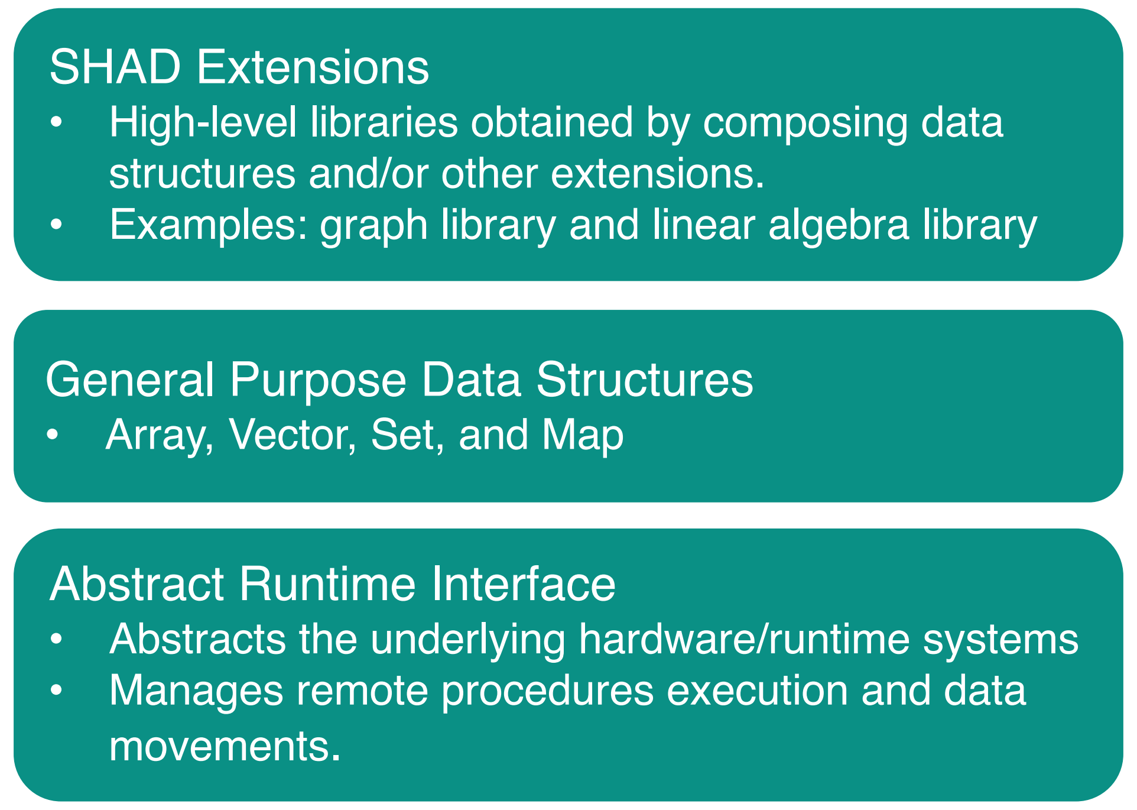
Towards a Scalable and Distributed High-performance SHAD C++ library

Nanmiao Wu[1][2], Vito Giovanni Castellana[1], and Hartmut Kaiser[2]

[1]Pacific Northwest National Laboratory, Richland, WA
[2]Center for Computation and Technology, Louisiana State University, Baton Rouge, LA

SHAD

- ✓ SHAD is a high-performance algorithm and data structure library, providing general purpose building blocks and supporting high-level custom utilities.
- ✓ SHAD is also a platform for research in parallel programming models, runtime systems and their applications.
- ✓ SHAD provides scalability, high-performance, flexibility, productivity, and portability.
- ✓ SHAD is composed of three layers. An upper layer, called SHAD extensions, a middle layer of general purpose data-structures and algorithms, and a bottom layer of an Abstract Runtime Interface.



SHAD Extensions
- High-level libraries obtained by composing data structures and/or other extensions.
- Examples: graph library and linear algebra library

General Purpose Data Structures
- Array, Vector, Set, and Map

Abstract Runtime Interface
- Abstracts the underlying hardware/runtime systems
- Manages remote procedures execution and data movements.

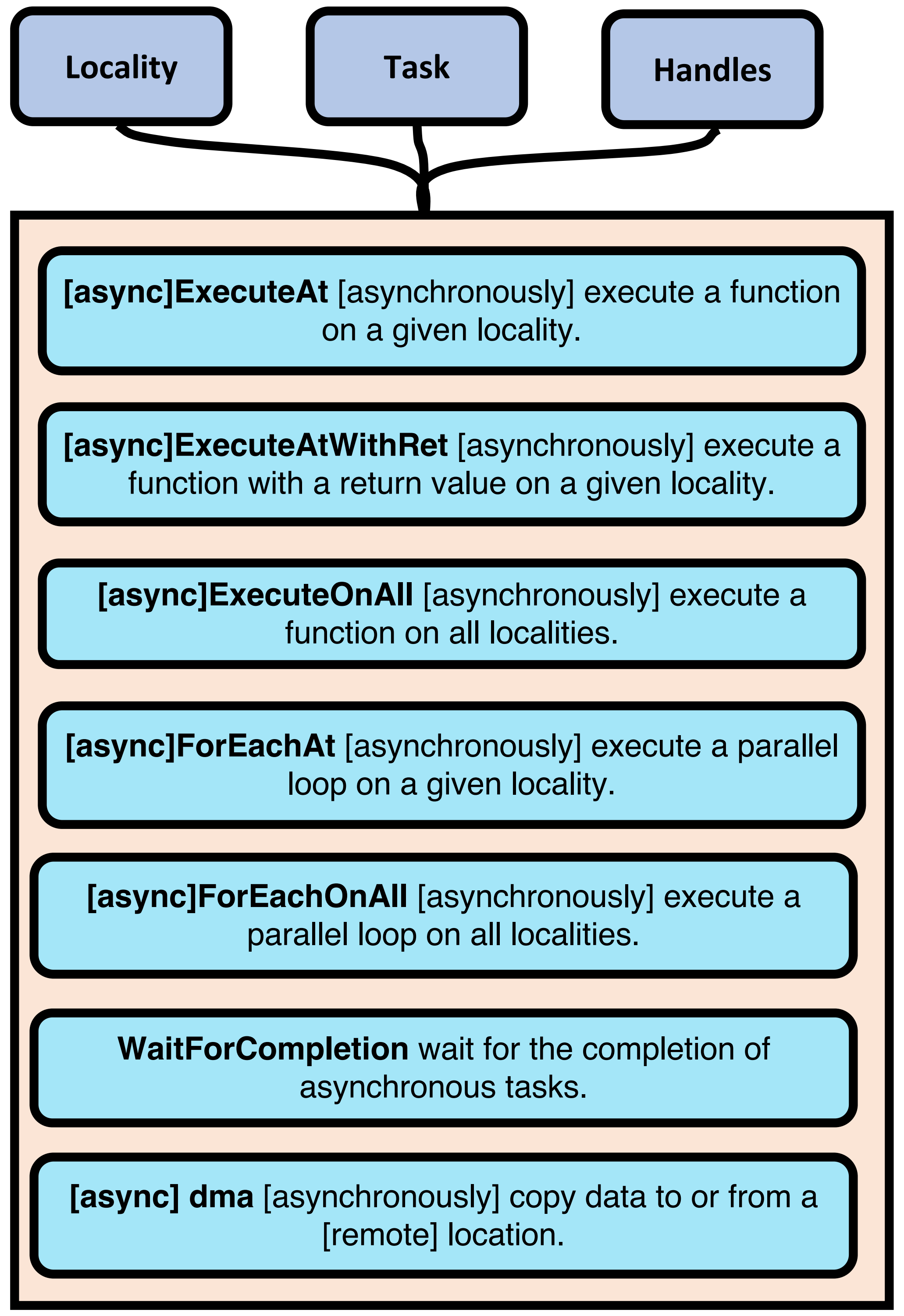
Figure 1. SHAD Stack

Abstract Runtime Interface

SHAD is built on an Abstract Runtime Interface, as shown in Figure 2, enabling the portability of SHAD on different platforms, e.g., Intel TBB and global memory and threading (GMT) by decoupling the upper layers of the stack and hiding the low level details of the underlying architecture.

Main concepts:
- ✓ Locality: an entity in which memory is directly accessible. A locality can be a node in a cluster, a core, or a NUMA domain.
- ✓ Task: a basic unit of computation, which can be executed synchronously or asynchronously on any locality.
- ✓ Handles: identifiers for spawning activities, which can be used to check for task completion. Multiple tasks can be associated to the same handle, forming task groups.

Motivation:
However, there are limitations of scalability using aforementioned backends. For Intel TBB, it is only well suitable for shared memory computers; while for GMT, it relies on a centralized controller which limits its scalability and creates a network hot spot due to all to one communication for synchronization, resulting in degraded performance at high process counts. Here, we explore the scalability, parallelism, and concurrency of SHAD library using HPX, a general purpose runtime system.



[async]ExecuteAt [asynchronously] execute a function on a given locality.

[async]ExecuteAtWithRet [asynchronously] execute a function with a return value on a given locality.

[async]ExecuteOnAll [asynchronously] execute a function on all localities.

[async]ForEachAt [asynchronously] execute a parallel loop on a given locality.

[async]ForEachOnAll [asynchronously] execute a parallel loop on all localities.

WaitForCompletion wait for the completion of asynchronous tasks.

[async] dma [asynchronously] copy data to or from a [remote] location.

Figure 2. Main Methods Offered by Abstract Runtime Interface

HPX Runtime

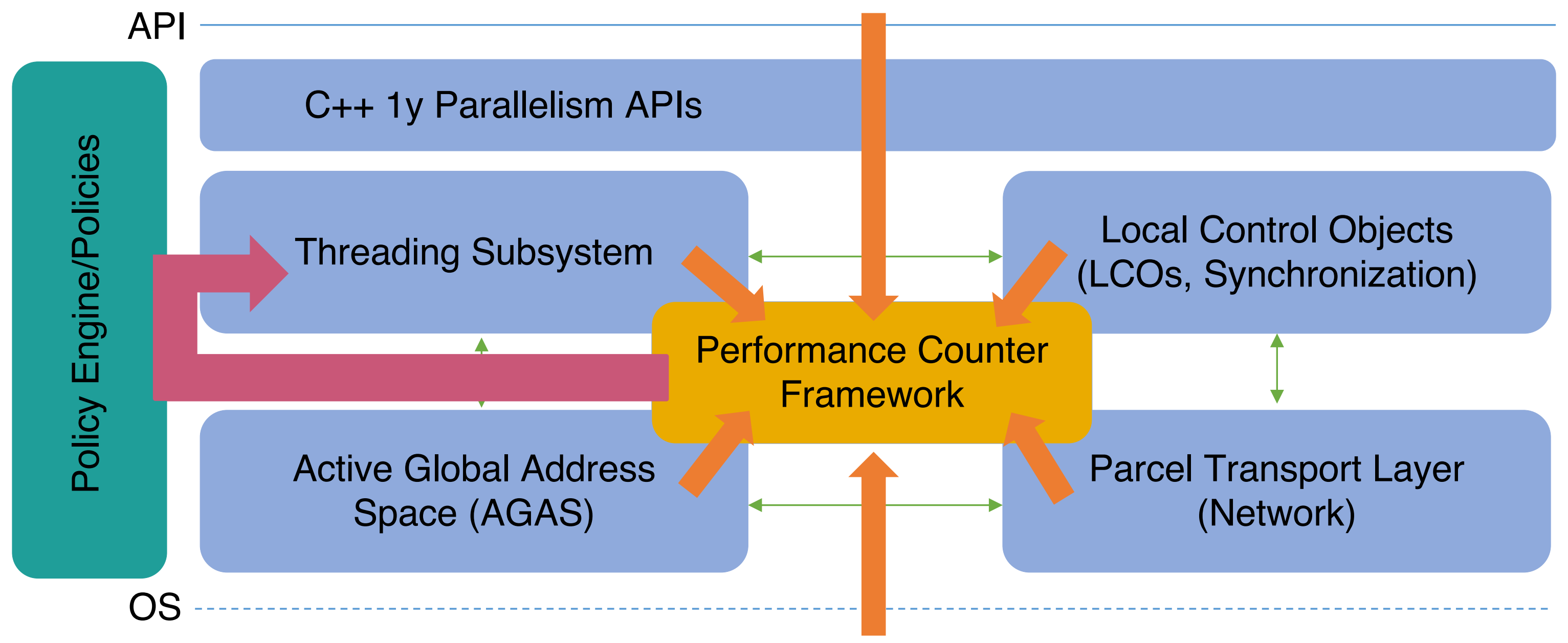

API
Policy Engine/Policies
C++ 1y Parallelism APIs
Threading Subsystem
Local Control Objects (LCOs, Synchronization)
Performance Counter Framework
Active Global Address Space (AGAS)
Parcel Transport Layer (Network)
OS

Figure 3. HPX Runtime System

HPX is a C++ library for concurrency and parallelism, with following major components.
- ✓ Unified and C++ standard-conforming API.
- ✓ Apex: an profiling framework.
- ✓ A work-stealing lightweight task scheduler.
- ✓ An Active Global Address Space (AGAS).

SHAD + HPX Runtime

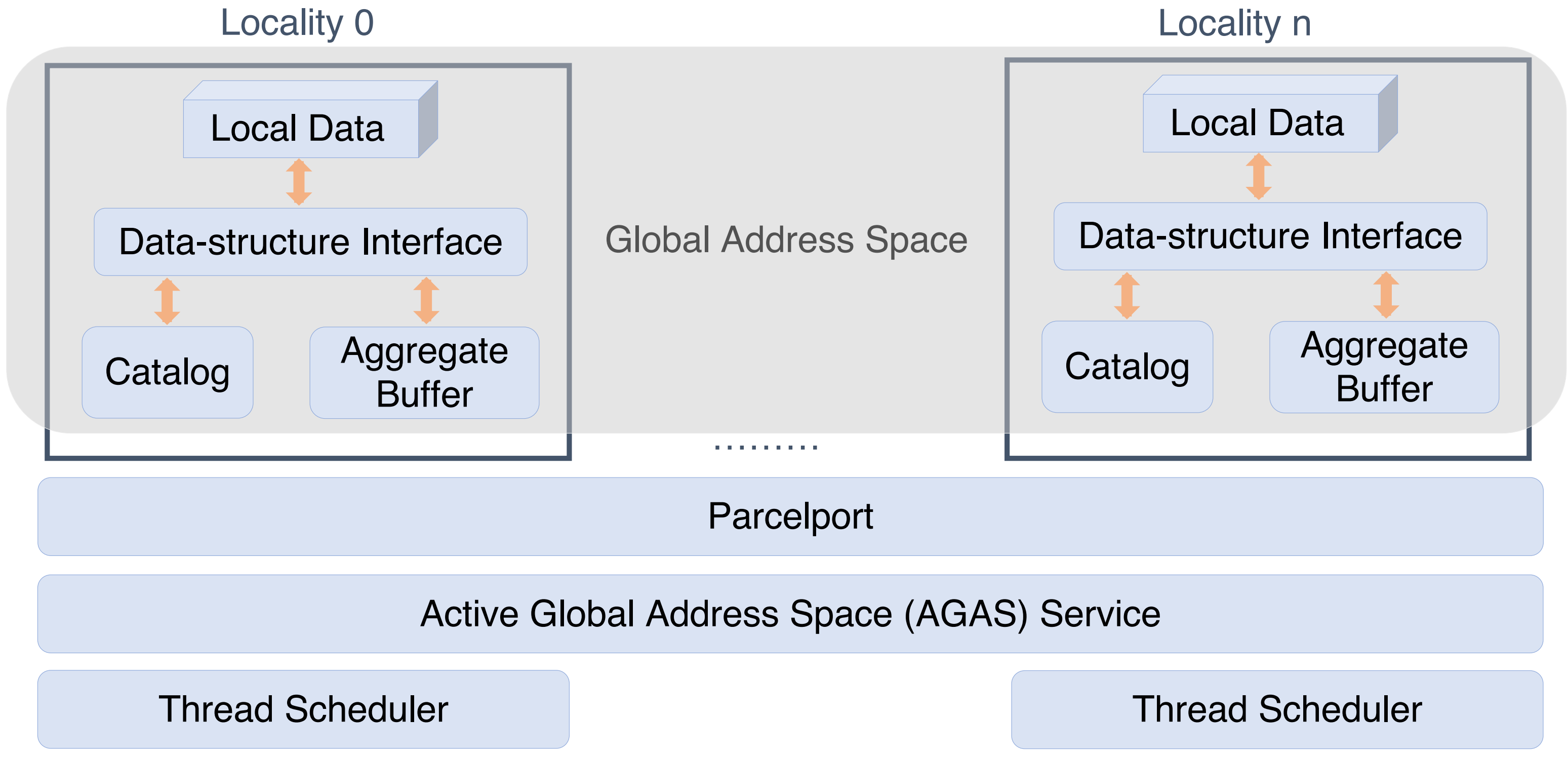


Figure 4. SHAD+ HPX Runtime

Using HPX as the runtime system, SHAD benefits from fine-grained parallelism, message driven computation, constraint-based synchronization, implicit overlapping computation and communication, and minimal overheads from the lightweight threading system.

High-level SHAD STL Compliant

- ✓ Semantics, concepts, and syntax are analogous to STL's API
  - ❑ iterators, ranges, algorithms.
- ✓ SHAD's iterators
  - ❑ distributed iterators, local iterators, local "chunk" iterators, insert iterators.
- ✓ Additional execution policies for performance
  - ❑ distributed_sequential: algorithms with sequential semantics.
  - ❑ distributed_parallel: analogous to std::par.
- ✓ Memory access patterns based design
  - ❑ map-fold, map-reduce, map-map, etc.
- ✓ Optimized performance by avoiding or reducing sync remote memory accesses

Experiments

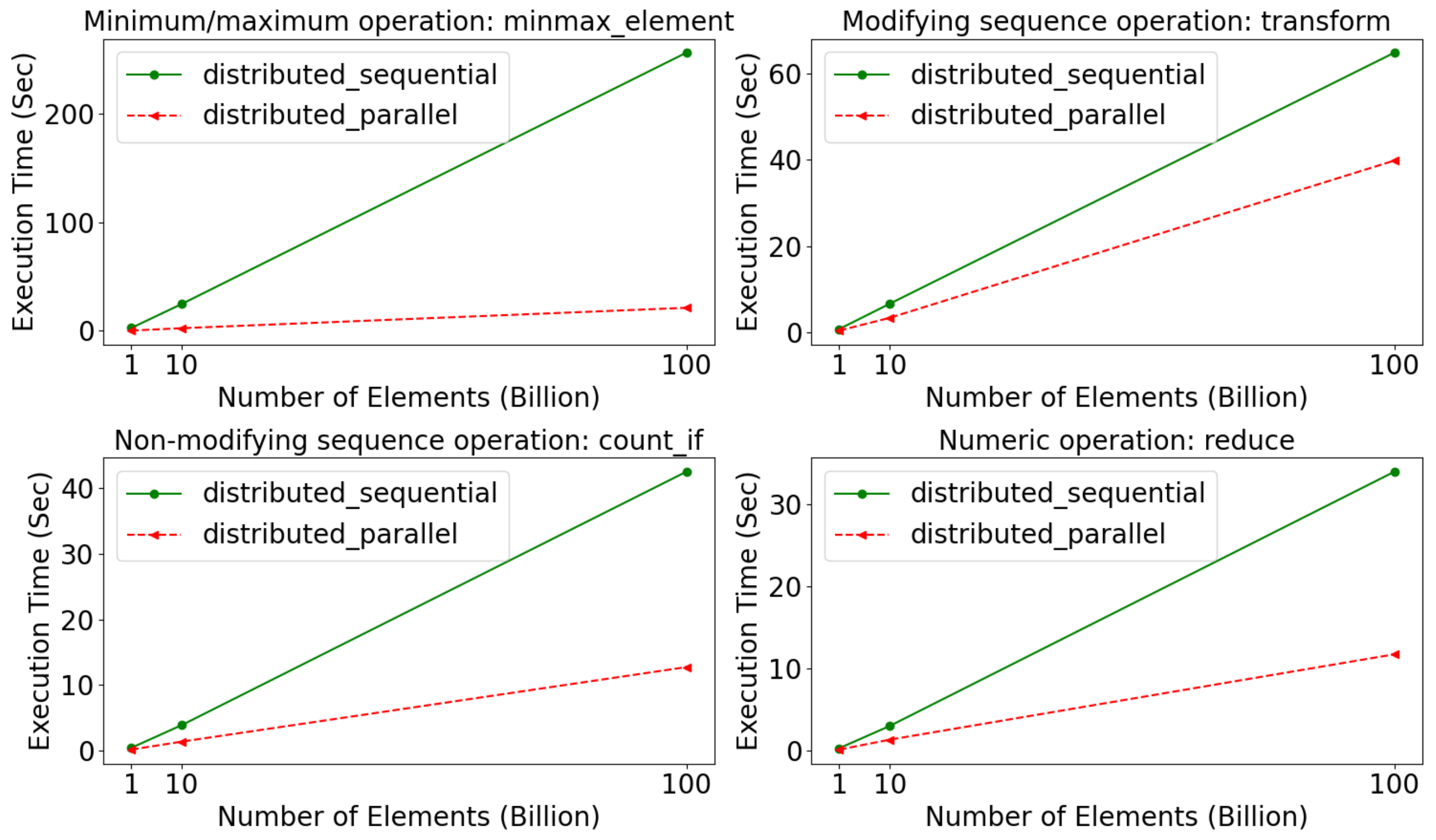


Figure 5. STL Algorithms Performance of SHAD Array

Evaluation:
- ✓ STL algorithms on a cluster of 20 core Intel(R) Xeon(R) E5-2680 v2 @2.80GHz and 768GB of memory per node. Note that we choose one algorithm from the each category of STL algorithms for the sake of simplicity.
- ✓ Use SHAD::array data structure with changing size, i.e., number of elements, from 1B to 100B in Figure 5.
- ✓ Insert 1 Billion elements per locality for SHAD::array, in Figure 6.

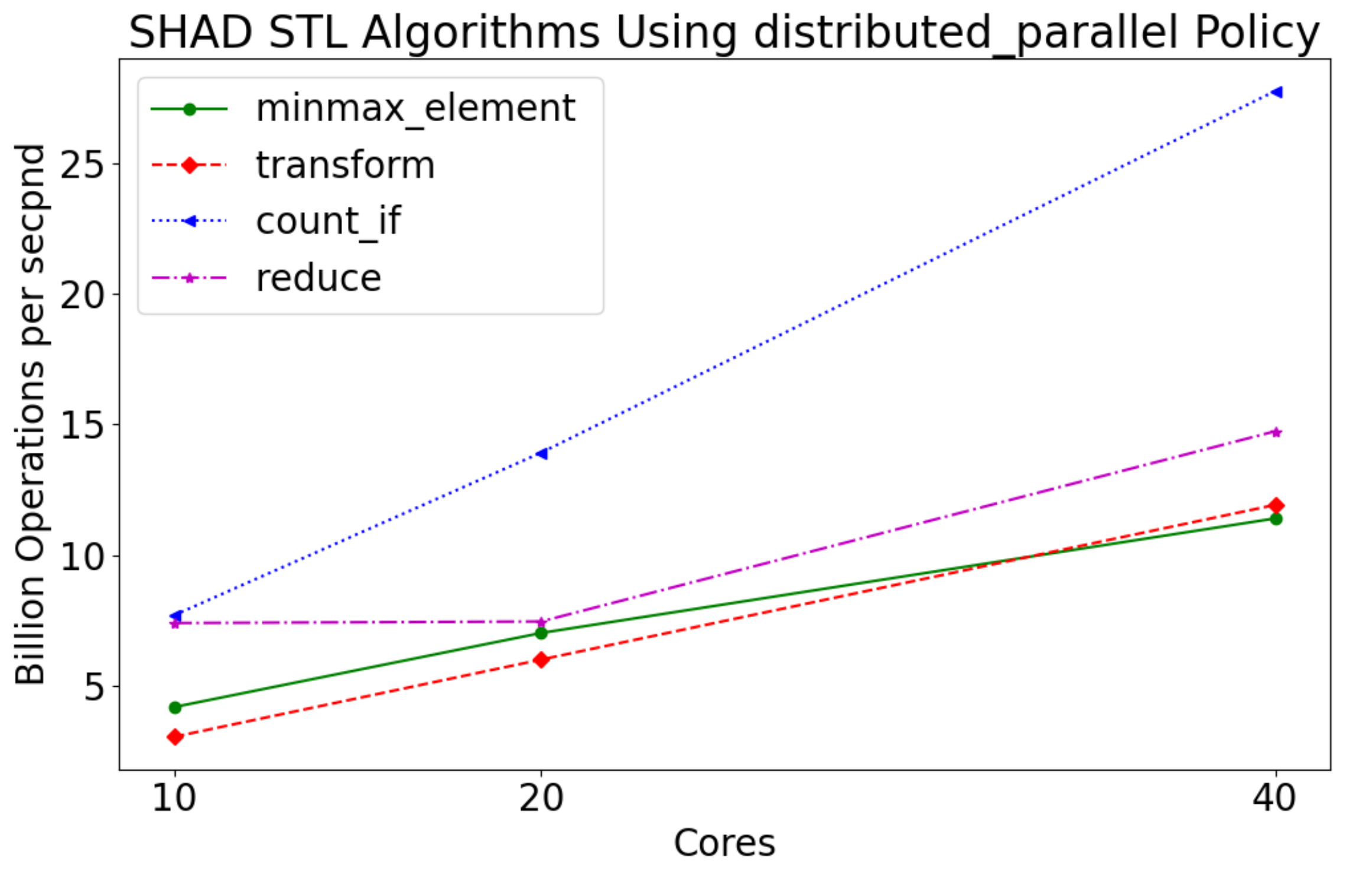


Figure 6. Weak Scaling using SHAD Array for STL Algorithms

Reference:

[1] V. G. Castellana and M. Minutoli, "SHAD: The Scalable High-Performance Algorithms and Data-Structures Library," 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Washington, DC, USA, 2018.

[2] H. Kaiser et al., "HPX-The C++ Standard Library for Parallelism and Concurrency ," Journal of Open Source Software, 2020.