Seminar summary of STAPL: Standard Template Adaptive Parallel Library

Aydan Namdar Ghazani 11709245
TU Wien (Software Engineering & Internet Computing)
Vienna, Austria
aydan.ghazani@tuwien.ac.at

1 INTRODUCTION

With the increasing amount of processors and cores in computer architectures the demand for parallel solutions in application development is emerging.

Designing a framework which brings portability and efficiency is what STAPL[1] promises. With their adaptive framework they provide the users with adaptive algorithm selection during runtime, abstracting away underlying communication and memory complexity. Ensuring high productivity and customizability while being scalable and performant.

Several application examples highlight the versatility of STAPL, including seismic ray-tracing for geophysics, particle transport for radiation simulation, and motion planning for robots.

Portability

Parallel algorithms are highly sensitive to architecture and hardware, emphasizing the necessity to maintain performance when transitioning from one hardware platform to another. Their is no benefit in using the library if the performance of a program is similar to a sequential alternative.

Maintainability

Ensuring maintainability involves concealing the intricate details of communication and program execution.

Inspired by the philosophy of C++, STAPL extends the STL containers, offering similar qualities. The primary goal is to provide good performance to the novice and more sophisticated control to experienced programmers.

To achieve this, STAPL introduces three levels of abstraction[1]:

- **Application Developer**
  - Writes applications
  - Utilizes `pContainers` and `pAlgorithms`
- **Library Developer**
  - Creates new `pContainers` and `pAlgorithms`
  - Utilizes `pRange` and `RTS (Run-time System)`
- **Run-time System Developer**
  - Interaction with the OS
  - Develops task scheduling modules
  - Utilizes native threading and communication libraries

2 RELATED WORK

A substantial body of work shares similar goals with STAPL, drawing inspiration from the STL philosophy of providing concepts like containers, iterators, and algorithms. Notable projects in this domain include:

PSTL

The Parallel Standard Template Library (PSTL) shares common goals with STAPL, employing parallel iterators as a parallel equivalent to STL iterators. While PSTL focuses on STL compatibility, STAPL extends STL by introducing additional parallel data structures and manages data dependencies for generic parallel algorithms.

TBB

Intel's Threading Building Blocks (TBB) implements some concepts from STAPL but targets only shared memory systems, specifically multi-cores. STAPL, in contrast, targets both shared and distributed systems with a strong emphasis on extensibility.

NESL, CILK, Split-C, Chapel, X10

These projects focus on nested parallelism, providing the ability to exploit nested parallel structures. STAPL, on the other hand, not only supports nested parallelism but also aims to automatically generate recursive parallelization without user intervention.

Split-C, X10, Chapel, Titanium

These languages provide a Partitioned Global Address Space (PGAS). STAPL provides a shared memory abstraction while exposing a PGAS architecture for advanced users.

3 OVERVIEW OF STAPL

STAPL, inspired by the C++ Standard Library, provides a shared object view of data with objects distributed across memory using unified addresses. The framework is designed to be platform-independent, aiming to offer a robust experience across diverse architectures. For the less experienced users memory management and communication gets abstracted away, while the experienced can still interact with the lower modules if needed. An important aspect of the framework is extendability and composability. Compose and conquer is the favored approach compared to divide and conquer.

pContainers

The distributed, thread-safe counterpart of STL containers, providing parallel methods that can be invoked concurrently. They are composable and extensible through inheritance.

STAPL offers counterparts of various STL containers, such as pArray, pVector, pList, pMap, alongside unique additions like pMatrix and pGraph. These pContainers offer both semantically equivalent methods to their sequential counterparts and methods specific to parallel computations.

pContainers Framework

Base Containers. Under the hood pContainers consist of one or multiple base containers (bContainers). The current bContainers in STAPL are based on the STL containers.

Global Identifiers. Each element in a pContainer is uniquely identified by a Global Identifier (GID), enabling a shared object view. GIDs are associated with elements, allowing the pContainer to locate them on a distributed memory machine.

Domain. A domain is the finite set of GID's of a container. It represents the universe of GIDs identifying its elements and facilitating unique traversal.

Data Distribution. The Data Distribution provides thread-safety and correct routing to the shared object methods. It handles the locations of the data. A location is a space which has contiguous memory address space and execution capabilities.

Location Manager. Within each location, a Location Manager is employed to store a subset of bContainers. The location manager may use different optimizations based on the properties of specific data structures, such as employing different memory allocators for space allocation by the bContainers.

pAlgorithms

pAlgorithms, on the other hand, are the parallel equivalent of STL algorithms, covering a wide range of parallel algorithms, including counterparts of STL algorithms.

STAPL introduces the concept of views, allowing pContainers to present multiple interfaces to users. For example, a pMatrix can be viewed as a row-major or column-major matrix or even as a linearized vector, providing different access to the data inside a container.

The execution of pAlgorithms is represented by pRanges, a graph structure where vertices are tasks and edges denote dependencies. The executor, a distributed shared object, manages parallel execution, updating dependencies and determining tasks for execution in collaboration with the scheduler. Nested parallelism is realized by invoking a pAlgorithm from within a task.

Runtime System

The runtime system is a platform dependent component which provides the API to the underlying operating system. It serves as a crucial component, adapting the system to new architectures, managing task scheduling modules, and utilizing native threading and communication libraries. The RTS includes the ARMI (Adaptive Remote Method Invocation) communication library, abstracting interprocessor communication, along with the executor, scheduler modules, and a performance monitor. ARMI simplifies communication across the distributed memory machine through a common remote method invocation interface to all STAPL components.

The RTS supports RMI versions of common aggregate operations, categorized as one-sided or collective. These operations are defined within communication groups, enabling nested parallelism. The executor executes task graphs corresponding to pAlgorithms, identifying independent tasks and scheduling them using a customizable scheduler module. The executor treats incoming RMI requests and algorithmic tasks as RTS tasks, which can be assigned to execution threads and are considered independent.

The RTS is not intended to be used directly by the developer it serves as an abstraction layer hiding the complex procedures of memory allocation and communication.

4 CONCLUSION

STAPL, a C++ parallel programming library, designed for productive development of parallel applications with a focus on ease of programming and performance portability, offers interfaces resembling the C++ standard library and includes components such as pContainers and pAlgorithms. We had a brief look into the core components of the framework. Their results have shown that the framework is scalable on tens of thousands of processors and portable across different architectures.

REFERENCES

[1] Buss, A., Harshvardhan, Papadopoulos, I., Pearce, O., Smith, T., Tanase, G., Thomas, N., Xu, X., Bianco, M., Amato, N. M., et al. Stapl: Standard template adaptive parallel library. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference* (2010), pp. 1–10.