INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN


**Masterarbeit**

# Dynamic PGAS Data Structures

Stefan Effenberger

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN


**Masterarbeit**

# Dynamic PGAS Data Structures

Stefan Effenberger

| | |
|---|---|
| Aufgabensteller: | Prof. Dr. Dieter Kranzlmüller |
| Betreuer: | Tobias Fuchs |
| Abgabetermin: | ADD DATE |

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den ADD DATE

. . . . . .

*(Unterschrift des Kandidaten)*

## Abstract

ADD ABSTRACT

# Inhaltsverzeichnis

# 1 Introduction

Many scientific projects are largely enabled by simulation. Because such simulations often require huge computational capabilites, single compute nodes with a shared-memory architecture cannot provide enough computation power and storage for numerous cases. For this reason, in High Performance Computing (HPC), work is distributed among multiple, interconnected nodes to facilitate the solving of large problems in a timely manner. Since processors cannot directly access the memory of other nodes, the traditional programming model for such systems requires programmers to explicitly distribute data between nodes via message passing. This imposes high demands on the programming skills of scientists who might not have a background in computer science.

Therefore, with the Partitioned Global Address Space (PGAS) model, a new approach is proposed: The memory space of individual nodes in a system is unified within a global address space so that each node can directly access the memory of all other nodes. Programmers are still required to keep data access between nodes to a minimum because data transferal over an interconnect is costly. To further reduce the demands on the programmer, distributed data structures that handle data distribution and load balancing are needed.

Furthermore, data-intensive tasks have been gaining a continually growing interest in the scientific community. Traditionally, applications in HPC follow a computation-centric approach by solving numerical algorithms in the fastest possible way. As "Big Data" is becoming increasingly important in scientific projects, a shift towards more data-oriented applications can be observed in recent HPC projects [ZZZ+14]. This trend requires distributed data structures that allow for the storage of large amounts of irregular data and cater to the needs of ever-changing dynamic data.

## 1.1 Problem statement

Data can be represented in numerous ways. The most generic form of data representation is enabled by *graphs*. A graph G(V, E) is a pair with a set of vertices V and a set of edges E that connect the vertices. This allows for the representation of data and its relationships in regular as well as irregular patterns.

On distributed machines, graph data structures can be implemented using a variety of different characteristics. This has lead to many different implementations - usually a new implementation for each algorithm - which are hardly compatible with each other. To overcome this situation, generic programming abstractions to facilitate reuse of existing code and to lower the demands on programmers are needed.

As of today, no generic graph abstractions implementing the PGAS model exist. This work therefore aims to provide a graph abstraction for C++ containers that allows for the implementation of arbitrary graph algorithms following the PGAS model on distributed memory machines.

## 1.2 Scope and Objectives

In this work, a C++ concept for graph containers following the PGAS model is presented. The concept is part of the DASH C++ Template Library and thus uses concepts already present in the library.

The graph concept is meant to provide a generic framework for the programming of arbitrary graph algorithms in the context of distributed machines and especially the Partitioned Global Address Space model. This means that it meets the following requirements:

- Native support for one-sided communication
- Support for the programming of synchronous graph algorithms
- Support for the programming of asynchronous graph algorithms
- Portability across platforms (PORTABLE EFFICIENCY?)
- Support for heterogenous systems
- FINISH REQUIREMENTS

Furthermore, this work provides concepts for the dynamic allocation of graph data across multiple machines with a focus on optimized data locality. LOAD BALANCING?

A reference implementation is then used to verify the usability, correctness and universality of the given concepts. While the concepts are designed for high performance, the reference implementation is not. This means that the evaluation of this implementation does not account for the performance of this work's concepts.

# 2 Background

This chapter covers some fundamental background knowledge needed for a better understanding of the following chapters of this thesis. Only explanations directly relevant to the topics of this thesis are provided.

Since the result of this work is a C++ concept, some important language expressions and concepts are firstly discussed, along with a description of the Standard Template Library on which concepts this work is built upon. The reader is then introduced to the domain of High Performance Computing which is the main application area for this work. A brief overview of the Partitioned Global Address Space programming model is then followed by a description of the DASH Library which provides core concepts used in this thesis.

## 2.1 C++ Concepts

### 2.1.1 Language Concepts

### 2.1.2 Standard Template Library

## 2.2 High Performance Computing

High Performance Computing (HPC) is a broad term describing advances for the fastest possible computation of a given problem. Gustafson's Law [Gus88] suggests that a compute system can linearly grow with the problem size: A problem of two times its original size can be computed on a system with twice as many processors in the same time (best case scenario). This means that very large problems can be computed in an acceptable timeframe if there is a sufficiently large compute system available. Depending on the problem size, two different system architectures are used in HPC:

**Shared Memory** A shared memory system consists of a single node with multiple processors connected to the same random access memory. Memory access for the different processors can be uniform, but many systems implement a non-uniform memory access (NUMA) design where a part of the memory is assigned to each of the processors. A processor in a NUMA system can access its assigned memory faster than the memory of the other processors. Because processors can access all data at all times, communication between processors has a low cost which simplifies programming on these systems in comparison to distributed memory systems. Achieving high performance on NUMA systems is more problematic because the programmer has to take data locality into account [Lam13].

**Distributed Memory** Multi-processor systems in which each processor has access to its own memory space are called distributed memory systems. These systems usually consist of several shared memory nodes with the processors of one node not being able to directly

access memory of other nodes. While single shared memory systems can only be scaled to a certain extent, the scalability of distributed systems is much higher [PTM96].

The nodes are connected with a network interconnect for communication between the processors. Due to the latency of the interconnect being significantly higher than the latency of a memory bus in a shared memory system, communication is much more costly. This imposes higher demands on the programmers' skills in comparison to shared memory systems.

The largest problems in science are computed on "supercomputers" like the *SuperMUC* at the *Leibniz Rechenzentrum* in Munich. These distributed memory machines consist of hundreds or even thousands of homogeneous nodes that are connected with a specialized interconnect. To this date, *message passing* is the prevalent programming model for such systems.

## 2.3 Partitioned Global Address Space

## 2.4 DASH Library

# 3 Related Work

## 3.1 Shared Memory

### 3.1.1 STINGER

### 3.1.2 Ligra

## 3.2 Distributed Memory

### 3.2.1 Parallel Boost Graph Library

### 3.2.2 STAPL Parallel Graph Library

# 4 Graph Container Concepts

## 4.1 Memory Space

## 4.2 Index Space

## 4.3 Iteration Space

## 4.4 Semantics

# 5 Reference Implementation

# 6 Case studies

## 6.1 Static structure

### 6.1.1 Graph traversal

### 6.1.2 Shortest path evaluation

## 6.2 Dynamic Structure

### 6.2.1 Graph partitioning

### 6.2.2 De Bruijn Graph construction

# 7 Evaluation

## 7.1 Micro-benchmarks

# 8 Conclusion

## 8.1 Summary

## 8.2 Assessment

## 8.3 Outlook

# Abbildungsverzeichnis

# Literaturverzeichnis


[Gus88] Gustafson, John L.: Reevaluating Amdahl's Law. In: *Commun. ACM* 31 (1988), Mai, Nr. 5, 532–533. http://dx.doi.org/10.1145/42411.42415. – DOI 10.1145/42411.42415. – ISSN 0001–0782

[Lam13] Lameter, Christoph: NUMA (Non-Uniform Memory Access): An Overview. In: *Queue* 11 (2013), Juli, Nr. 7, 40:40–40:51. http://dx.doi.org/10.1145/2508834.2513149. – DOI 10.1145/2508834.2513149. – ISSN 1542–7730

[PTM96] Protic, J. ; Tomasevic, M. ; Milutinovic, V.: Distributed shared memory: concepts and systems. In: *IEEE Parallel Distributed Technology: Systems Applications* 4 (1996), Summer, Nr. 2, S. 63–71. http://dx.doi.org/10.1109/88.494605. – DOI 10.1109/88.494605. – ISSN 1063–6552

[ZZZ+14] Zhao, D. ; Zhang, Z. ; Zhou, X. ; Li, T. ; Wang, K. ; Kimpe, D. ; Carns, P. ; Ross, R. ; Raicu, I.: FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In: *2014 IEEE International Conference on Big Data (Big Data)*, 2014, S. 61–70