

# **Evaluation of performance and productivity metrics of potential programming languages in the HPC environment**

**— Bachelor Thesis —**

Division Scientific Computing  
Department of Informatics  
Faculty of Mathematics, Informatics und Natural Sciences  
University of Hamburg

Submitted by:	Florian Wilkens
E-Mail:	1wilkens@informatik.uni-hamburg.de
Matriculation number:	6324030
Course of studies:	Software-System-Entwicklung
First assessor:	Prof. Dr. Thomas Ludwig
Second assessor:	Sandra Schrø"der
Advisor:	Michael Kuhn

Hamburg, 01.01.2012

# Abstract

This thesis aims to analyze new programming languages in the context of HPC. To compare not only speed but also development productivity and general inner metrics, a basic traffic simulation is implemented in C, Mozilla's Rust and Google's Go. These two languages were chosen on their basic promise of performance as well as memory-safety in the case of Rust or easy multithreaded execution (Go). The implementations are limited to shared-memory parallelism to achieve a fair comparison since the library support for inter-process communication is rather limited at the moment.

Nonetheless the comparison should allow a decent rating of the viability of these two languages in high-performance computing.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Motivation . . . . .	4
1.2	Candidates . . . . .	4
1.3	Goals of this Thesis . . . . .	6
<b>2</b>	<b>Design</b>	<b>7</b>
<b>3</b>	<b>Implementation</b>	<b>8</b>
<b>4</b>	<b>Evaluation</b>	<b>9</b>
<b>5</b>	<b>Conclusion</b>	<b>10</b>
	<b>Bibliography</b>	<b>11</b>
	<b>List of Figures</b>	<b>12</b>
	<b>List of Tables</b>	<b>13</b>
	<b>List of Listings</b>	<b>14</b>
	<b>Appendices</b>	<b>15</b>
<b>A</b>	<b>Appendix</b>	<b>16</b>

# 1. Introduction

*In diesem Kapitel ...*

## 1.1. Motivation

The world of high-performance computing is evolving rapidly and programming languages used in this environment are held up to a very high standard. It comes as no surprise that runtime performance is the top priority in language selection when an hour of computation costs thousands of dollars. The focus on raw power led to C and Fortran having an almost monopolistic position in the industry, because their execution speed is nearly unmatched.

However there are several problems inherent to these rather antique languages. Although they are still in active development, their long lifespans resulted in unintuitive syntax (Fortran) and `..(C)`. Especially newcomers are often torn away by the seemingly endless ways a C program can cause a segmentation fault.

## 1.2. Candidates

This section aims to provide a rough overview of possible candidates to be used in high performance computing. Each language is introduced and categorized and at the end a quick comparison shows which are evaluated further in the thesis.

### Python

Most of the people programming for high performance computers are scientist of research fields other than computer science. With this in mind Python seems a logical programming language to use. It is an interpreted general-purpose programming language which aims to be very expressive and flexible. Compared with C and Fortran which sacrifice feature richness for performance, Python's huge standard library combined with the automatic memory management offers a low border of entry and quick prototyping capabilities. As a matter of fact many introductory computer science courses at universities in the United States recently switched from Java to Python as their first programming language.(Guo 2014) This allows the students to focus on core concepts and algorithms instead of boilerplate code.

---

// code example (hello world?)

In addition to the very extensive standard library the Python community has created a lot of open source projects aiming to support especially scientific applications. There is NumPy<sup>1</sup> which offers efficient implementations for multidimensional arrays and common numeric algorithms like Fourier transforms or MPI4Py<sup>2</sup>, an MPI abstraction layer able to interface with various backends like OpenMPI or MPICH.

Unfortunately dynamic typing and automatic memory management come at a high cost...

## Erlang

Erlang is a relatively niche programming language originally designed for the use in telephony applications. It features a high focus on concurrency and a garbage collector which is enabled through the execution inside the BEAM virtual machine.

- brief history?
- code example (not hello world rather show message passing)
- Upsides - Great concurrency - Message passing is default (no locks) - Hot swap? - Downsides - Bad interfacing to other languages - Weird syntax - Limited (community/-support?)

## Go

Go is a relatively new programming language which focusses on simplicity and clarity while not sacrificing too much performance.

- brief history
  - hello world or something similar

Go was chosen as a candidate because it provides simple concurrency primitives as part of the language (goroutines) while having a C-like syntax and being reasonably performant. It also compiles to native code without external dependencies which makes it usable on cluster computers which might not have required libraries installed.

- Prediction implementation - A bit of syntax weirdness - Relatively quick PoC with decent concurrency aspects - Some fixing/optimization afterwards regarding common concurrency errors -> More time spent after initial PoC but less than in C

## Rust

The last possible candidate is Rust. Developed in the open but strongly backed by Mozilla Rust aims to directly compete with C and C++ as a systems language. It focuses

---

<sup>1</sup>[www.numpy.org](http://www.numpy.org)

<sup>2</sup>[www.mpi4py.scipy.org](http://www.mpi4py.scipy.org)

---

on memory safety which is checked and verified at compile without (or with minimal) impact on runtime performance.

- Key features
- Up/Downside
- Prediction Implementation - Moderately quick PoC without concurrency at first - Nearly only optimization afterwards since compilation secures memory safety -> More time spent before initial PoC than after

## Comparison

Language	Execution model	Biggest advantages for HPC	Relative performance
Python	interpreted	very expressive, low barrier of entry	slow to average
Erlang	compiled to bytecode	native concurrency support	average to fast

## 1.3. Goals of this Thesis

## 2. Design

*In diesem Kapitel ...*

# **3. Implementation**

*In diesem Kapitel ...*



## 4. Evaluation

*In diesem Kapitel ...*

## 5. Conclusion

*In diesem Kapitel ...*

- Only evaluated shared memory -> Multi process implementations -> C: MPI, Rust: MPI via C FFI & opaque pointer, Go: MPI via wrapper? (less idiomatic code)

# Bibliography

Guo, Philip (2014). *Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities*. URL: <http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext> (visited on 12/07/2014).

## List of Figures

## List of Tables

## List of Listings

# Appendices

## A. Appendix

- System configuration -> clang/gcc version -> rustc and cargo version (including commit hash!) -> go version



## Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen – benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

**Optional:** Ich bin mit der Einstellung der Bachelor-Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik einverstanden.

Hamburg, den 26.11.2014 .....