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 Schröder
Advisor:	Michael Kuhn, Sandra Schröder

Hamburg, February 6, 2015

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 multhithreaded 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.

Table of Contents

1. Introduction

This chapter aims to provide some background information to high-performance computing. The first section describes problems with the currently used programming languages and motivates the search for new candidates. After that the chapter concludes with a quick rundown of the thesis' goals.

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 programming in these rather antique languages can be rather difficult. Although they are still in active development, their long lifespans resulted in sometimes unintuitive syntax and inconsistent behaviour. Especially C's *undefined behaviour* often causes inexperienced programmers to write unreliable code which is unnecessarily dependant on implementation details of a specific compiler. Understanding and maintaining these programs requires deep knowledge of memory layout and other computer internals?

Considering the fact that scientific applications are mostly written by scientist without a concrete background in information technology it is evident that the current situation is less than ideal. - new technologies emerged - badly (and late) or not supported at all by c and fortran -> time to evaluate new languages while keeping performance in mind

1.2. Goals of this thesis

This thesis aim to evaluate Rust and Go as potential programming languages in the high-performance computing environment. The comparison is based on a reimplementation of an existing parallel application in the two languages as well as C. This application is streets4MPI, a traffic simulation software written in Python using MPI to parellelize calculations. Since libraries for interprocess communication in Rust and Go are nowhere near production-ready this thesis will focus on shared memory parallelization to avoid unfair bias based solely on the quality of the supporting ecosystem.

The final application is a simplified version of the original streets4MPI but will behave nearly identical. It uses the OpenStreeMap Project's .osm.pbf files as input and writes the results to a custom output format for later analysis. To reduce complexity it does not support additional commandline arguments and has limited error handling regarding in- and output.

While performance will be the main concern additional software metrics will also be reviewed to measure the complexity and overall quality of the produced applications. Another aspect to review is the tool support and ease of development.

- evaluate languages for use in (scientific) high-performance computing - shared memory -> thread-parallelization - tools, (common) frameworks, - through implementation of streets4mpi in C, go and rust - performance(!) - metrics

2. State of the art

In diesem Kapitel ...

- state of C and Fortran (section name?)
- technological advancements in low level languages - static analysis - .. -> But no real adaption possible, because language level support is missing

2.1. Weaknesses of C and Fortran

As stated in the introduction high-performance computing is largely dominated by C and Fortran. To understand why a new language is needed it is essential to understand the shortcomings of these programming veterans. (//language?)

// Candidates here for now might need another chapter for those

2.2. 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 programing language. [Guo14] 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[1] which offers efficient implementations for multidimensional arrays and common numeric algorithms like Fourier transforms or MPI4Py[2], an MPI abstraction layer able to interface with various backends like OpenMPI or MPICH. Especially the existance of the latter shows the ongoing attempts to use Python in a cluster environment and there have been successful examples of scientific high-performance applications (//need ref).

Unfortunately dynamic typing and automatic memory management come at a rather high price. The speed of raw numeric algorithms written in plain Python is almost always orders of magnitude slower than implementations in C or Fortran. As a consequence nearly all of the mentioned libraries implement the critical routines in C and focus in optmizing the interop (// wording) experience. This often means one needs to make tradeoffs between idiomatic Python - which might not be transferable to the foreign language - and maximum performance.

- python losing expressiveness (find alternatives) -> python losing justification for HPC - alternative implementations (medusa! and others)

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 beeing resonably performant. It also compiles to native code without external dependencies which makes it usable on cluster computers which might not have required libraries installed.

[1] www.numpy.org
[2] www.mpi4py.scipy.org

- 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 candidate discussed in this chapter 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 on memory safety which is checked and verified at compile without (or with minimal) impact on runtime performance.

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

Comparison

	Python	Erlang	Go	Rust
Execution model	interpreted	compiled to bytecode	compiled	compiled to native code
Advantages	low barrier of entry	builtin (lockfree) concurrency support	adv go	adv rust
Disadvantages	speed	obscure syntax	mandator	disadv rust
Relative speed	slow to average	average to fast	speed go	speed rust

3. Approach

3.1. Overview: Streets4MPI

4. Implementation

In diesem Kapitel ...

5. Evaluation

In diesem Kapitel ...

6. 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

[Guo14] Philip Guo. *Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities*. Oct. 7, 2014. 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 (possibly including commit hash!) -> go version and implmentation (cgo?)

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 ....