High Performance Rust

Jim Walker

July 22, 2019

MSc in High Performance Computing

The University of Edinburgh

Year of Presentation: 2019

Abstract

This dissertation examines the suitabilty of the Rust programming language, to High Performance Computing (HPC). This examination is made through porting three HPC mini apps to Rust from typical HPC languages and comparing the perfomance of the Rust and the original implementation. We also investigate the readability of Rust's higher level programming syntax for HPC programmers through the use of a questionnaire.

Contents

List of Tables

List of Figures

Acknowledgements

This template is a slightly modified version of the one developed by Prof. Charles Duncan for MSc students in the Dept. of Meteorology. His acknowledgement follows:

This template has been produced with help from many former students who have shown different ways of doing things. Please make suggestions for further improvements.

Chapter 1

Introduction

In the field of high performance computing, it is difficult to say what is the most popular programming language. Firstly, we must define what we mean by popularity. Do we mean how many CPU hours are spent running programs from a particular language? Or do we mean the languague in which most of the development of new high performance programs is occuring? Or even, do we mean which programming language is most well liked by HPC programmers? The Rust programming language promises 'High-level ergonomics and low-level control' to help 'you write faster, more reliable software' [1].

Chapter 2

Methodology

2.1 Mini-Apps

Mini-apps are a well established method of assessing new programming languages or techniques within HPC [2, 4, 3]. A mini-app is a small program which reproduces some functionality of a common HPC use case. Often, the program will be implemented using one particular technology, and then ported to another technology. The performance of the two mini-apps will then be benchmarked, to see which technology is better suited to the particular problem represented by that mini-app.

2.1.1 Selection

So that a bredth of usage scenarios were examined, three mini-apps were selected based on their conformity to the following set of criteria.

- **The program's kernel (i.e. the part of the program responsible for more than two thirds processing time) should not be more than 1500 lines.** To ensure that I fully implemented three ports of exisiting mini-apps, it was necessary to limit the size of the mini-apps that could be considered. This was an unfortunately necessary decision to make. Whilst it reduced the field of possible mini-apps, an analysis of the rejected mini-apps found that many of them devoted lots of code to subtle computational variations, which were of more importance to a particular rareified domain, rather than presenting a novel approach to parallelism. (i could cite some benchmarks here like bookleaf or something)
- **The program must use shared memory parallelism and target the CPU.** Rust's (supposed) zero cost memory safety features are its unique feature. The best way to test the true cost of Rust's memory safety features would be through shared memory parallelism, where a poor implementation of memory management will make itself evident through poor performance.

- **The program run time should reasonably decrease as the number of threads increases, at least until the number of threads reaches 32.** It is important that any mini-app considered is capable of scaling to the high core counts normally seen in HPC.
- **The program should perform at least two mathematical operations on data greater than the CPU's L3 Cache** so that we can be sure that the mini-app is representative of working on large data sets.
- **The program must be written in C or C++.** This restriction allows us to choose work which is more representative of HPC programs that actually run on HPC systems, rather than python programs which call out to pre-compiled libraries. C and C++ also use array indexing and layout conventions simillar to Rust, which will make porting programs from them easier.
- **The program must use OMP.** This is the defacto standard for shared memory parallelism in HPC. Use of a library to do the parallel processing also further standardises the candidate programs.

2.1.2 Implementation

Babel Stream

2.2 Questionnaire

Chapter 3

Babel Stream

3.1 Development

3.2 Comparison

Chapter 4

Sparse Matrix Multiplication

4.1 Development

4.2 Comparison

Chapter 5

K-means

5.1 Development

5.2 Comparison

Chapter 6

Rust's usability

Here are some questionnaire results.

Chapter 7

Conclusions

This is the place to put your conclusions about your work. You can split it into different sections if appropriate. You may want to include a section of future work which could be carried out to continue your research.

Appendix A

Stuff which is too detailed

Appendices should contain all the material which is considered too detailed to be included in the main bod but which is, nevertheless, important enough to be included in the thesis.

Appendix B

Stuff which no-one will read

Some people include in their thesis a lot of detail, particularly computer code, which no-one will ever read. You should be careful that anything like this you include should contain some element of uniqueness which justifies its inclusion.

Bibliography

[1] Steve Klabnik and Carol Nichols. The rust programming language.

[2] A. C. Mallinson, S. A. Jarvis, W. P. Gaudin, and J. A. Herdman. Experiences at Scale with PGAS versions of a Hydrodynamics Application. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '14, pages 9:1–9:11, New York, NY, USA, 2014. ACM.

[3] Matthew Martineau and Simon McIntosh-Smith. The arch project: physics mini-apps for algorithmic exploration and evaluating programming environments on hpc architectures. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 850–857. IEEE, 2017.

[4] Elliott Slaughter, Wonchan Lee, Sean Treichler, Michael Bauer, and Alex Aiken. Regent: A High-productivity Programming language for HPC with Logical Regions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 81:1–81:12, New York, NY, USA, 2015. ACM.