Work Package 5.2: A Report

Alasdair Macindoe

Acknowledgements

We acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541). Further we would like to thank Dr Markus Pfeiffer for his supervision and all his help that was invaluable for this project.

Introduction

Whilst enumerating finite semigroups is computationally expensive there are algorithms that can run - in a practical sense - faster than naively enumerating the semigroup. One example of this algorithm was published in 1997 by Froidure and Pin[3] (which will hereby be called the Froidure-Pin algorithm). Unfortunately this algorithm does not run concurrently which is a problem for modern computational problems; thus this lead to another version of Froidure-Pin to be published[5] (which we will henceforth call the concurrent Froidure-Pin) which can run concurrently and for which a C++ implementation exists[1].

The aim of this project was to re-implement the concurrent Froidure-Pin algorithm in HPC-GAP[4] and analyse its scalability. The full repository for this project can be found on Github[6].

Structure

All the code can be located on Github[6]. The following may be of interest:

`README.md`: Installation and testing instructions

`versions.md`: Explanation of implementation details

`runtimes.md`: Raw data from experiments

`data.md`: Explanation of different semigroups for testing

`experiment.g`: File to run experiments

`gap/`: Various implementations

`tst/`: The test file

Implementation

`Version 2.1` is an exact implementation of the version given in the paper. Since the paper does not dictate what data structures should be used the full implementation details can be found in `versions.md`. This gives very detailed overview of the algorithm and why it can run locklessly, as well as the reasons for the specific data structures.

Analysis

Please refer to `runtimes.md` for the raw data, only summaries are provided below. [[TO BE COMPLETED]]

Contributions to HPC-GAP

A bug in HPC-GAP was discovered which would cause a segmentation fault when an empty list was attempted to be migrated between tasks. This was fixed by Dr Markus Pfeiffer[7].

Future Work

We have identified some potential future works that could be done:

1. Fix any outstanding bugs and perform more significant testing
2. Covert data structures across to the HPC-GAP's datastructures package[2]
3. Investigate performance increases by using HPC-GAP's threads and processes API
4. Investigate performance increases in HPC-GAP's tasks system

Conclusion

[[TO BE COMPLETED]]

Bibliography

[1] J.D Mitchell et al. Semigroups - gap package. *https://github.com/gap-packages/Semigroups*.

[2] Max Horn et al. Datastructures. *https://github.com/gap-packages/datastructures*.

[3] Véronique Froidure and Jean-Eric Pin. *Algorithms for computing finite semigroups*, pages 112–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[4] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.8.8*, 2017.

[5] J. Jonušas, J. D. Mitchell, and M. Pfeiffer. Two variants of the Froiduire-Pin Algorithm for finite semigroups. *ArXiv e-prints*, April 2017.

[6] Alasdair G. Macindoe. Parallel semigroups gap. *https://github.com/Alasdair-Macindoe/parallel-sgrp-gap*.

[7] Markus Pfeiffer. Fix migrateobjects for empty list of objects. *https://github.com/gap-system/gap/pull/1602*.