Frederik Hantho
Jan Wiberg
Rune Bromer

Project title: pupyMPI - Pure Python MPI

Authors: Frederik Hantho, Jan Wiberg & Rune Bromer

Project consultant: Brian Vinter

ECTS: 15

Background

The demand for easy, highly scalable, parallel computing is growing daily as many-core systems become prevalent. The Message Passing Interface (MPI) is a widely used and globally accepted model for expressing parallelism. The success of MPI stems from its portability, completeness and performance i.r.t. massively parallel programs, amongst others. However, MPI is generally not considered simple, nor easy to program, particularly as it is most commonly implemented in C or FORTRAN. This is a problem since many potential users are not skilled programmers, but rather researchers who are capable of expressing their algorithms in a deterministic logical framework.

The difficulty involved in ensuring the correctness and maintainability of C has lead to the rise of higher level languages, such as Python. Python is a language that originated in academia, and has large and growing popularity in this group where it is often used for prototyping solutions to scientific problems. Amongst Pythons many advantages are a strong typing system, a highly readable syntax, existence on many platforms, and it is quite mature.

Python has no built-in support for MPI, but several other projects have grafted MPI functionality onto Python, with varying degrees of completeness. At present, there seems to be no native Python MPI implementations openly available.

Motivation

Coupling Python with MPI would seem to have several distinct advantages.

1. The advanced type system alleviates the complexities of MPI types as Python types can be easily inferred at runtime.
2. Python is standardized across platforms and is either already installed, or very easy to install, on almost any major platform of interest to HPC users. Python is also highly modular and easy to extend.
3. Implementing MPI in native Python means that there is no need to maintain a separate native MPI installation, and by extension no need to keep

specific versions of Python, the MPI wrapper and the MPI library around to maintain binary compatibility.

4. The low bar to entry to Python means that it is highly useful as a prototyping language, and educators can focus on teaching central concepts of concurrency, without the clutter of boilerplate code so typical to e.g. C.

Learning goals

At the end of the project, we will be able to:

- Analyze the shortcomings of the MPI libraries written in a strict programming language.
- Reason about the advantages and disadvantages of a native Python MPI compared with other Python MPI solutions.
- Reason about different internal communication strategies for achieving high performance.
- Choose a design for a high performance MPI-like library, and argue for the choice.
- Reflect on the performance and reliability of a prototype implementation of our design.