The Journal of Open Source Software

schwimmbad: A uniform interface to parallel processing pools in Python

Adrian M. Price-Whelan[1] **and Daniel Foreman-Mackey**[2]

1 Lyman Spitzer, Jr. Fellow, Princeton University **2** Sagan Fellow, University of Washington

DOI: 10.21105/joss.00357

Software

- Review
- Repository
- Archive

Summary

Many scientific and computing problems require doing some calculation on all elements of some data set. If the calculations can be executed in parallel (i.e. without any communication between calculations), these problems are said to be *perfectly parallel*. On computers with multiple processing cores, these tasks can be distributed and executed in parallel to greatly improve performance. A common paradigm for handling these distributed computing problems is to use a processing "pool": the "tasks" (the data) are passed in bulk to the pool, and the pool handles distributing the tasks to a number of worker processes when available.

In Python, the built-in `multiprocessing` package provides a `Pool` class for exactly this design case, but only supports distributing the tasks amongst multiple cores of a single processor. To extend to large cluster computing environments, other protocols are required, such as the Message Passing Interface (MPI; Forum 1994). `schwimmbad` provides new `Pool` classes for a number of parallel processing environments with a consistent interface. This enables easily switching between local development (e.g., serial processing or with Python's built-in `multiprocessing`) and deployment on a cluster or supercomputer (via, e.g., MPI or JobLib). This library supports processing pools with a number of backends:

- Serial processing: `SerialPool`
- `Python` standard-library `multiprocessing`: `MultiPool`
- `OpenMPI` (Gabriel et al. 2004) and `mpich2` (Lusk, Doss, and Skjellum 1996) via the `mpi4py` package (Dalcín, Paz, and Storti 2005; Dalcín et al. 2008): `MPIPool`
- `joblib`: `JoblibPool`

All pool classes provide a `.map()` method to distribute tasks to a specified worker function (or callable), and support specifying a callback function that is executed on the master process to enable post-processing or caching the results as they are delivered.

References

Dalcín, Lisandro, Rodrigo Paz, and Mario Storti. 2005. "MPI for Python." *Journal of Parallel and Distributed Computing* 65 (9): 1108–15. doi:http://dx.doi.org/10.1016/j.jpdc.2005.03.0

Dalcín, Lisandro, Rodrigo Paz, Mario Storti, and Jorge D'Elía. 2008. "MPI for Python: Performance Improvements and Mpi-2 Extensions." *Journal of Parallel and Distributed Computing* 68 (5): 655–62. doi:http://dx.doi.org/10.1016/j.jpdc.2007.09.005.

Forum, Message P. 1994. "MPI: A Message-Passing Interface Standard." Knoxville, TN, USA: University of Tennessee.

Gabriel, Edgar, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra,

Jeffrey M. Squyres, Vishal Sahay, et al. 2004. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation." In *Proceedings, 11th European Pvm/Mpi Users' Group Meeting*, 97–104. Budapest, Hungary.

Lusk, Ewing, Nathan Doss, and Anthony Skjellum. 1996. "A High-Performance, Portable Implementation of the Mpi Message Passing Interface Standard." *Parallel Computing* 22: 789–828.