Available online at www.sciencedirect.com

Future Generation Computer Systems 22 (2006) 123–157

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

Using B SP and Python to simplify parallel programming

Konrad Hinsen[a,*], Hans Petter Langtangen[b], Ola Skavhaug[b], Åsmund Ødegård[b]

[a] *Centre de Biophysique Moléculaire, UPR 4301 CNRS, Rue Charles Sadron, 45071 Orleans Cedex 2, France*
[b] *Simula Research Laboratory (SRL), P.O. Box 134, 1325 Lysaker, Norway*

Available online 3 March 2004

Abstract

Scientific computing is usually associated with compiled languages for maximum efficiency. However, in a typical application program, only a small part of the code is time-critical and requires the efficiency of a compiled language. It is often advantageous to use interpreted high-level languages for the remaining tasks, adopting a mixed-language approach. This will be demonstrated for Python, an interpreted object-oriented high-level language that is well suited for scientific computing. Particular attention is paid to high-level parallel programming using Python and the BSP model. We explain the basics of BSP and how it differs from other parallel programming tools like MPI. Thereafter we present an application of Python and BSP for solving a partial differential equation from computational science, utilizing high-level design of libraries and mixed-language (Python–C or Python–Fortran) programming.
© 2004 Published by Elsevier B.V.

Keywords: BSP; Python; Parallel programming

1. Introduction

Scientific computing has some specific requirements that influence the choice of programming tools. The most outstanding property of scientific computing is its explorative nature: although some standard methods are used over and over again, they are used in different combinations every time, and often it is necessary to add custom algorithms and programs to a collection of well-established standard code. Although the literature on scientific computing may leave the impression that all that matters are efficient number-crunching and visualization methods, the day-to-day work of a computational scientist involves a lot of interfacing, file format conversion, bookkeeping, and similar tasks, often made difficult by bad user interface design and lack of documentation. These lengthy and unattractive tasks often discourage scientists to pursue a potentially interesting idea. Good programming tools can thus make a significant contribution to good computational science.

High-level languages can help in several ways. At the simplest level, they can be used to write all tools that are not time-critical, such as simple analysis programs,

* Corresponding author.
E-mail addresses: hinsen@cnrs-orleans.fr (K. Hinsen), hpl@simula.no (H. Petter Langtangen), skavhaug@simula.no (O. Skavhaug), aasmundo@simula.no (Å. Ødegård)

0167-739X/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.future.2003.09.003

file format converters, etc. As a general rule, high-level languages are better suited for I/O- and text-oriented tasks than the standard low-level programming languages used in scientific computing: Fortran, C, and C++. However, as this paper will show, they can be useful in number-crunching applications as well, making the programs easier to develop and use. The key to these applications is mixed-language programming, i.e., combining a high-level and a low-level language in order to get the best of both worlds.

To avoid misunderstandings, an explanation of the term “high-level” is in order. Most of all, it implies no judgment of quality. High-level languages are by definition those whose constructs and data types are close to natural-language specifications of algorithms, as opposed to low-level languages, whose constructs and data types reflect the hardware level. With high-level languages, the emphasis is on development convenience, whereas low-level languages are designed to facilitate the generation of efficient code by a compiler. Characteristic features of high-level languages are interactivity, dynamic data structures, automatic memory management, clear error messages, convenient file handling, libraries for common data management tasks, support for the rapid development of graphical user interfaces, etc. These features reduce the development and testing time significantly, but also incur a larger runtime overhead leading to longer execution times.

We remark that what is called “high-level” in this paper is often referred to as “very high level”; different authors use different scales. Many people also use the term “scripting languages”.

The high-level language used as an example in this paper is Python [7], a language that is becoming increasingly popular in the scientific community. Although other suitable languages exist and the choice always involves personal preferences, Python has some unique features that make it particularly attractive: a clean syntax, a simple yet powerful object model, a flexible interface to compiled languages, automatic Interface Generators for C/C++ and Fortran, and a large library of reusable code, both general and scientific. Of particular importance is Numerical Python [6], a library that implements fast array operations and associated numerical operations. Many numerical algorithms can be expressed in terms of array operations and implemented very efficiently using Numerical Python. Moreover, Numerical Python arrays are used at the interface between Python and low-level languages, because their internal data layout is exactly the same as a C array. We will in this paper adopt the widely used term NumPy as a short form of Numerical Python.

The outline of this paper is as follows. Section 2 discusses two different programming styles for utilizing high-level languages for scientific computations. We then turn to the topic of high-level parallel computation, using the BSP model, in Section 3. This simple model of parallel computing, in combination with Python, enables communication of high-level data types, and eliminates the risk of deadlocks often encountered in other parallel computing tools like MPI and PVM. Section 4 describes how to implement a parallel partial differential equation simulator, using Python and BSP. In this section, we also show how Python can be extended by migrating time-critical functions to tailor-made C and Fortran code. We then run some numerical benchmarks against similar Matlab and C code to quantify the loss of efficiency by using Python and BSP for simplified, high-level parallel computing. The final section summarizes the findings and states some concluding remarks.

2. High-level language programming styles

There are basically two ways of utilizing a high-level language like Python in scientific computing. Either we equip an external application or library with a Python interface, or we create a new application from scratch in Python and migrate time-critical operations to Fortran or C. These two approaches are described next.

2.1. Python interfaces to existing numerical codes

A typical situation in computational science is the following: an existing program contains all the relevant methods, but its user interface is cumbersome, I/O facilities not sufficient, and interfacing with other programs could be easier. Another common case is the existence of a library of computational algorithms which is used by relatively simple application programs that are constantly modified. In this case, modification and testing of the applications often take a significant amount of time.