High-Level User Interfaces for the DOE ACTS Collection

L. Anthony Drummond[1], Vicente Galiano[2], Violeta Migallón[3], and José Penadés[3]

[1] Lawrence Berkeley National Laboratory,
One Cyclotron Road, Berkeley CA 94703, United States of America
LADrummond@lbl.gov
[2] Departamento de Física y Arquitectura de Computadores,
Universidad Miguel Hernández, ES-03202 Elche, Alicante, Spain
vgaliano@umh.es
[3] Departamento de Ciencia de la Computación e Inteligencia Artificial,
Universidad de Alicante, ES-03071 Alicante, Spain
{violeta, jpenades}@dccia.ua.es

Abstract. The ACTS collection project comprises a set of state-of-the-art software tools to speed up the development of High-Performance Computing Applications in science and engineering. We look at the development of High Level user interfaces using scripting languages like Python, to facilitate the access to ACTS technology to a wide community of computational scientists. PyACTS is our main project here, but we also visit other efforts within the community of developers of ACTS tools.

1 Introduction

The Advanced CompuTational Software (ACTS) [1] Collection comprises a set of computational tools developed primarily at DOE laboratories, sometimes in collaboration with universities and other funding agencies (NSF, DARPA), aimed at simplifying the solution of common and important computational problems. A number of important scientific problems have been successfully studied and solved by means of computer simulations built on top of tools available in the ACTS Collection [2]. The ACTS Collection brings robust and high-end software tools to the hands of application developers to accelerate the development of computational science codes and consequent results. However, this transfer of technology is not always successful due in part to the intricacy in understanding the interfaces associated with the software tools and the time an application scientists spends installing and learning the use of a given tool. Here we present a set of Python based interfaces to some of the tools in the ACTS Collection, PyACTS. We also present some examples of it applications and future development directions.

B. Kågström et al. (Eds.): PARA 2006, LNCS 4699, pp. 251–259, 2007.

2 Some of the Tools in the ACTS Collection

In Table 1 we briefly list some of the numerical functionality available in the ACTS Collection. The tools in Table 1 have development projects that include interfaces in Python. A closer look at the functionality offered by these tools [1], we see that there are some tools that compliment others (i.e., the use of a direct solver inside a preconditioner used by an iterative scheme, or the use of a preconditioner from *Tool A* inside *Tool B*, etc.) and tools with functionality that overlap. Selecting the appropriate tool is not trivial and problem specific, it may require in some cases not only expertise in numerical linear algebra but also extensive testing and tuning. To be able to explore the full functional plethora in the ACTS Collection, a user may spend months learning the different interfaces and parameterizations of a giving tool. Our goal with PyACTS is to build a high level user interface that directly reduces the amount of work a user spends simply learning to use a tool, facilitates a faster development of her or his application.

PyACTS [15,16,17] provides a didactical user interface to assist with their first application prototype and following production code development. Here we look at the PyACTS development project and existing functionalities.
The reader is referred to the ACTS Information Center [18] for more details on these tools and others available in the collection. Our initial work has been focused on the development of PyScaLAPACK which is introduced in the next section.

3 PyACTS: A Python Interface to the ACTS Collection

Python [19] is an interpreted, interactive, object-oriented programming language. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. New built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface. Python is designed to make integration with other software components in a system as simple as possible. Programs written in Python can be easily blended with other languages. For instance, Python scripts can call out existing C and C++ libraries, Java classes, and much more. Actually, it is this feature of Python that is employed in our current work.

Additionally, Python is portable: it runs on many brands of UNIX, on Windows, Mac, and many other platforms. Python is copyrighted but freely usable and distributable, even for commercial use. Python is an ideal language for prototype development and other ad–hoc programming tasks, without compromising maintainability and it uses an elegant syntax for readable programs. All of the ACTS tools listed in the previous section use MPI as one of the methods for supporting message passing. In the PyACTS, we use pyMPI [20], which enables us to use the same Python modules and rich functionality. We have also tested other Python implementations of MPI and these can be replaced without any portability issues because of the MPI functionality used inside PyACTS is available in all flavor implementations and the observed performance is quite similar.

Table 1. A subset of the Numerical Tools in the ACTS Collection with their Python based Interfaces. At this time, all these third party Python based projects are independent of PyACTS and not a part of the ACTS collection.

TOOL	Short Description
ScaLAPACK [3] and PyScaLAPACK from PyACTS	Library of high-performance linear algebra routines for distributed-memory message-passing Multiple Instruction Multiple Data (MIMD) computers and networks of workstations. The library contains routines for solving systems of linear equations, least squares, eigenvalue problems and singular value problems. It also contains routines that handle matrix factorizations or estimation of condition numbers.
SuperLU [4] and PySuperLU from PyACTS	General purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines. The library is written in C and is callable from either C or Fortran. The library routines perform an LU decomposition with numerical pivoting and triangular system solves through forward and back substitution.
PETSc [5] and PyPETSc [6]	The **P**ortable, **E**xtensible **T**oolkit for **S**cientific **c**omputation [7], provides sets of tools for the parallel, as well as serial, numerical solution of PDEs that require solving large-scale, sparse linear and nonlinear systems of equations. PETSc includes nonlinear and linear equation solvers that employ a variety of Newton techniques and Krylov subspace methods.
SUNDIALS and some available Python bindings	**SU**ite of **N**onlinear and **DI**fferential/**AL**gebraic equation **S**olvers, and it refers to a family of four closely related solvers; CVODE [8,9], for systems of ordinary differential equations; CVODES [10], variant of CVODE for sensitivity analysis; KINSOL [11], for systems of nonlinear algebraic equations; and IDA [12], for systems of differential-algebraic equations.
Trilinos [13] and PyTrilinos [14]	A framework for the development of parallel solvers and libraries within an object-oriented environment. AztecOO is one of the libraries available in Trilinos and it is part of the ACTS Collection. The Trilinos framework offers a variety of mechanisms for a software package to interact with other software packages.

PyACTS is a collection of carefully designed and written *software wrappers* to the ACTS tools, it also includes other routines written in Python to provide high

level users interfaces. Therefore, wrappers written for PyPBLAS, PyBLACS and PyScaLAPACK were generated first with the help of F2PY [21], and then we loaded these wrappers with functionality that will automatically validate the arguments passed to the actual PyACTS routines, and check for consistency between the types of objects expected by the ACTS tools. These wrappers also provide us with the ability to transparently convert data types between PyACTS modules to support interoperability. More details on these wrappers are giving later in this section. In additional, PyACTS interfaces contain fewer arguments in their calls but generate automatically other parameters that are later passed to the actual ACTS tool interfaces. An example of this abstraction is shown with a PBLAS 3 example in Figure 1. In the panel (a) of this figure, we notice that there are parameters like the PBLAS descriptors that do not contribute directly with the operation, $C = \alpha AB + \beta C$, but are there to support the parallelism and optimize the algorithmic implementation. Many users in computational science and engineering do not care for these levels of details.

In example illustration (Figure 1), PyACTS removes these extra arguments from the PyPBLAS user interface. And internally PyPBLAS will automatically generate the missing parameters for the user and execute the proper call to the corresponding PBLAS routine. Furthermore, one more complex part of the ScaLAPACK and SuperLU interfaces is the handling of the two dimensional cyclic distribution. PyACTS provides an automatic mechanism to create the data layout and manage the resulting data distributions for the user.

While our PyACTS implementations automatically generates many of the parameters for the user, and provides support functionality, the user can still modify this parametric behavior by calling directly a routine at a lower lever of the PyACTS structure. Thus, PyACTS has resulted in a modular tool that support users with different levels of expertise with ACTS tools.

Figure 2 illustrates the internal structure of the PyACTS software. As illustrated in this graph we use components of pyMPI and Numerical Python, NumPy [22], to provide array management and parallelism. Additionally we have created a set of utilities to facilitate I/O of different formats (e.g., NETCDF, ascii), and general purpose processing routines.

The utilities module is shared by all the components of PyACTS and they are not particular to a tool in ACTS. The individual tool modules (e.g., the PyScaLAPACK module, PySuperLU module, etc.) contain the Python bindings to the ACTS tools.

The Python wrappers provide not only a level of transparency to some tool arguments but also a set of well designed validation procedures and generation of extra arguments to call the Fortran or C language libraries. Validation procedures verify that the correct variables are passed as parameters to a given routine. For instance when calling a PyScaLAPACK routines that takes a matrix as argument, the verification will consist of a checking of the ScaLAPACK type matrix and that the matrix has indeed values before it calls to the actual ScaLAPACK matrix. Internally, it also checks whether the corresponding ScaLAPACK contexts for the distributed arrays have been created.

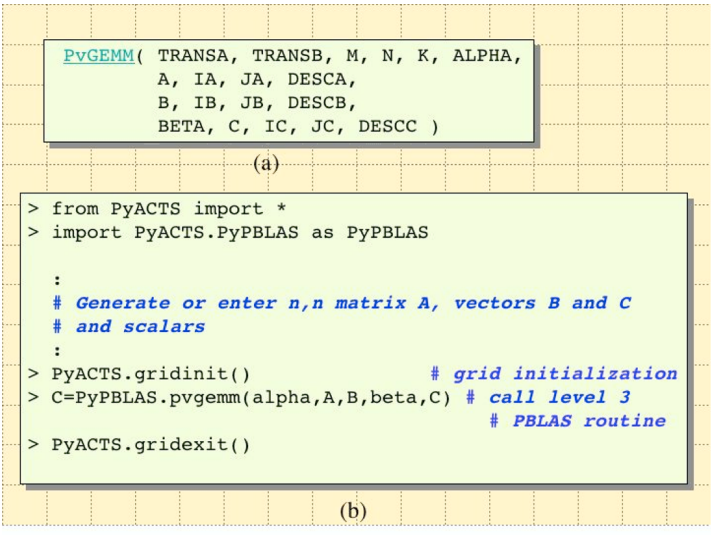


Fig. 1. The generic BLAS 3 Fortran and C call is shown in panel (a), and panel (b) shows the simplified PyACTS, specifically PyPBLAS, version of the same call

In PyACTS, the interoperability between ACTS tools is managed via a collection of routines for conversions of data representations between the different ACTS tools. For instance, this allows a user to convert a matrix from SuperLU into a PETSc matrix in an easier manner.

Currently, we have developed an interface to ScaLAPACK and SuperLU, PyScaLAPACK [15] and PySuperLU, respectively. In addition, we have designed a modular implementation of PyACTS that is shown in Figure 2. This design allows for easily handling of different versions of the same package and also the interoperability with other Python interfaces from other ACTS tool developers. For instance, PETSc and SUNDIALS provide their own Python extensions. Trilinos provides PyTrilinos [23], with Python extensions to provide access to most of the Trilinos functionality. TAU [24], a performance profiling and tuning tool in the ACTS Collection, can also profile programs written in Python. Thus, the PyACTS modular structure still allows for integration of existing PyACTS functionality with the ones being developed by other ACTS tool developers. As shown also in Figure 2, a user wanting to use PyACTS needs to have installed: MPI, BLAS, BLACS, ScaLAPACK, Python 2.1 or later, and NumPy

4 Some Examples of Applications Using PyACTS Modules

In this section we briefly present results from two parallel implementations of the Conjugate Gradient (CG) algorithm. The first implementation written in

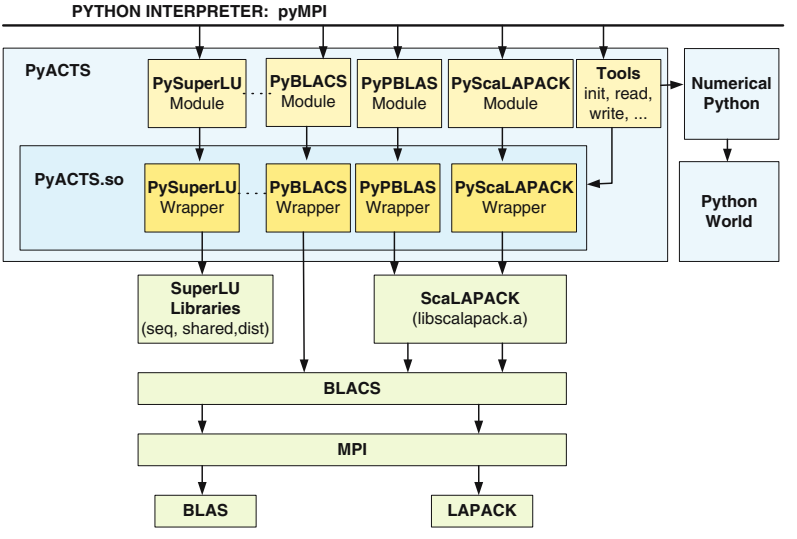
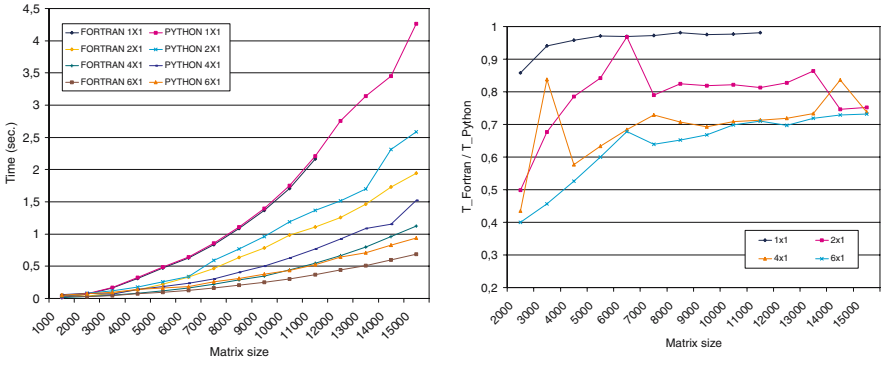


Fig. 2. Modular Structure of the PyACTS Project

pure Fortran and the second one in Python using PyACTS modules. There are many implementations of the CG for high performance computing, including robust and scalable implementations that are offered in the ACTS Collection. Furthermore, there is extensive literature on the subject that discusses in detail its derivation, applications and performance issues of the CG Algorithm. Thus, our goal here is not to provide a better version of the CG or extend the literature on the subject, but rather to illustrate with an example and performance results the low overhead of PyACTS. We have chosen this example because of the multiple calls to PBLAS routines and parallel manipulation of vectors and matrices.

In the PyACTS code, we have used the PyPBLAS module, and in the Fortran code we call the counterpart PBLAS routines directly. The experiments were perform in a Linux cluster with 6 2.0GHz Intel processors and 512Mbytes memory per processor. The BLAS level routines were previously optimized with ATLAS [25].

The different curves correspond to different processor grids for the two code implementations. First, we observe a marginal difference in the timings between the PyACTS and Fortran versions. Because Python automatically provides a more efficient memory management mechanism than Fortran (i.e., a Fortran code without any special memory management nor memory optimization schemes) in some cases we were able to run the parallel PyACTS implementation with matrix sizes and processes grids that were not possible with Fortran due to memory limitations. For instance see 1×1 configuration for matrix sizes over 11,000 in Figure 3(a). Figure 3(b) shows the relationship between the execution time of the Fortran code and Python based one ($Time\_FortranCode \div Time\_PyACTSCode$)



(a) Actual Run Times (b) Relative Run Times

Fig. 3. Comparing a Fortran vs a Python implementation of the Conjugate Gradient algorithm. We use PyACTS modules for the Python implementation.

A number of other examples have been performed with both PyScaLAPACK and PyPBLAS routines [26], and the results have consistently displayed a marginal difference between the PyACTS and Fortran code implementations in different computer platforms.

Additionally, PyScaLAPACK has already been used inside scientific applications [26]. In particular, it has been used to provided parallel functionality to a sequential Python-based package called PyClimate. PyClimate [27] provides support to common tasks during the analysis of climate variability data. It provides functions that range from simple IO operations and operations with COARDS-compliant netCDF files to Empirical Orthogonal Function (EOF) analysis, Canonical Correlation Analysis (CCA) and Singular Value Decomposition (SVD) analysis of coupled data sets, some linear digital filters, kernel based probability-density function estimation and access to DCDFLIB.C library from Python. PyClimate uses functionality available in LAPACK.

5 Conclusions and Future Work

Although, early experiments with PyACTS have shown a low overhead induced by the Python-based interface, PyACTS is not yet intended for large production runs in high-end system, rather it is a didactical tool for generating a first prototype of the application code. It helps the user to become familiar with a particular interface and also access in an interoperable manner other ACTS tools interface without having to learn it in great detail. We envision that the popularity of high-level programing languages, and their portability to many computer system will in the future enable technology that will make this high-level programming languages to scale. Thus, PyACTS may also scale to thousands of processors.

We are currently working on a PyACTS *scribe* that will allow to write out the Fortran and C language equivalent functions of the High-Level PyACTS routines.

Therefore, a user that prototypes an application using PyACTS will be able to get the exact Fortran or C calling interface sequence in order to produce a code that can be compiled and used for production runs in a large number of system.
In the future, we will be working closely with other ACTS tool developers and integrating more functionality to PyACTS.

Acknowledgments. This work was partially supported by the Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract No. DE-AC03-76SF00098", the Spanish Ministry of Science and Education under grant number TIN2005-093070-C02-02, and by Universidad de Alicante under grant number VIGROB-020.

References

1. Drummond, L.A., Marques, O.A: An Overview of the Advanced CompuTational Software (ACTS) Collection. ACM Transactions on Mathematical Software 31(3), 282–301 (2005), http://doi.acm.org/10.1145/1089014.1089016
2. Drummond, L.A., Hernández, V., Marques, O., Román, J.E., Vidal, V.: A Survey of High-Quality Computational Libraries and Their Impact in Science and Engineering Applications. In: Ganter, B., Godin, R. (eds.) ICFCA 2005. LNCS (LNAI), vol. 3403, pp. 37–50. Springer, Heidelberg (2005)
3. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J.W., Dhillon, I., Dongarra, J.J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK User's Guide. SIAM, Philadelphia (1997)
4. Demmel, J.W., Gilbert, J.R., Li, X.: SuperLU User's Guide. University of California, Berkeley (2003)
5. Balay, S., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M., Curfman McInnes, L., Smith, B.F., Zhang, H.: PETSc's home page (2007), http://www.mcs.anl.gov/petsc
6. Korvola, T.: PyPETSc (2005), http://www.elisanet.fi/tkorvola/hacks/''
7. Balay, S.K., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. Modern Software Tools in Scientific Computing. In: Arge, E., Bruaset, A.M., Langtangen, H.P. (eds.) Modern Software Tools in Scientific Computing, pp. 163–202. Birkhauser Press (1997)
8. Cohen, S.D., Hindmarsh, A.C.: CVODE User Guide. Technical Report UCRL-MA-118618, Lawrence Livermore National Laboratory (1994)
9. Byrne, G.D., Hindmarsh, A.C.: User documentation for PVODE, an ODE solver for parallel computers. Technical Report UCRL-ID-130884, Lawrence Livermore National Laboratory (1998)
10. Hindmarsh, A.C., Serban, R.: User Documentation for CVODES, An ODE Solver with Sensitivity Analysis Capabilities. Technical Report UCRL-MA-148813, Lawrence Livermore National Laboratory (2002)
11. Taylor, A.G., Hindmarsh, A.C.: User Documentation for KINSOL, A nonlinear solver for sequential and parallel computers. Technical Report UCRL-ID-131185, Lawrence Livermore National Laboratory (1998)
12. Hindmarsh, A.C., Taylor, A.G.: User Documentation for IDA, a Differential-Algebraic Equation Solver for Sequential and Parallel Computers. Technical Report UCRL-MA-136910, Lawrence Livermore National Laboratory (1999)

13. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An Overview of the Trilinos Project. ACM TOMS Vol 31:3 (2004) 127
14. Sala, M., Spotz, W., Heroux, M.: PyTrilinos: High-performance distributed-memory solvers for Python. ACM TOMS (2006) (submitted)
15. Galiano, V., Drummond, L.A., Migallón, V., Penadés, J.: High Level User Interfaces for High Performance Libraries in Linear Algebra: PyBLACS and PyPBLAS. In: Proceedings from 12th International Linear Algebra Society Conference, University of Regina, Regina, Saskatchewan, Canada (2005)
16. Drummond, L.A., Galiano, V., Migallón, V., Penadés, J.: Improving ease of use in BLACS and PBLAS with Python. In: Joubert, G., Nagel, W., Peters, F., Plata, O., Tirado, P., Zapata, E. (eds.) Parallel Computing: Current & Future Issues of High-End Computing, Proceedings of the International Conference ParCo 2005, vol. 33, NIC series (2006) ISBN 3-00-017352-8
17. Kang, N., Drummond, L.A.: A first prototype of PyACTS. Technical Report LBNL-53849, Lawrence Berkeley National Laboratory (2003)
18. Marques, O.A., Drummond, L.A.: The ACTS Information Center (2007), `http://acts.nersc.gov`
19. van Rossum, F.D.J.G.: An Introduction to Python. Network Theory Ltd (2003)
20. Miller, P.: PyMPI - An introduction to parallel Python using MPI (2002), `http://www.llnl.gov/computing/develop/python/pyMPI.pdf`
21. Peterson, P.: F2py users guide and reference manual (2005), `http://cens.ioc.ee/projects/f2py2e/`
22. Ascher, D., Dubois, P.F., Hinsen, K., Hugunin, J., Oliphant, T.: Numerical Python. Lawrence Livermore National Laboratory, Livermore, CA 94566, UCRL- MA-128569 (2001), `http://numpy.sourceforge.net`
23. Sala, M.: Distributed Sparse Linear Algebra with PyTrilinos. Technical Report SAND2005-3835, Sandia National Laboratories (2005)
24. Shende, S., Malony, A.D.: The tau parallel performance system. International Journal of High Performance Computing Applications 20, 287–311 (2006)
25. Whaley, R.C., Petitet, A., Dongarra, J.: Automated empirical optimizations of software and the atlas project. Parallel Computing 27, 3–25 (2001)
26. Drummond, L.A., Galiano, V., Marques, O.A., Migallón, V., Penadés, J.: PyACTS: A High-Level Framework for Fast Development of High Performance Applications. In: Lecture Notes in Computer Science. vol. 4395, pp. 417–425 (2007)
27. Saenz, J., Zubillaga, J., Fernández, J.: Geophysical data analysis using Python. Computers and Geosciences 28/4, 457–465 (2002)