Exporting Ada Software to Python and Julia

Jan Verschelde
University of Illinois at Chicago, Department of Mathematics, Statistics, and Computer Science, 851 S. Morgan St. (m/c 249), Chicago, IL 60607-7045; email: janv@uic.edu, http://www.math.uic.edu/∼jan

Abstract

The objective is to demonstrate the making of Ada software available to Python and Julia programmers using GPRbuild. GPRbuild is the project manager of the GNAT toolchain. With GPRbuild the making of shared object files is fully automated and the software can be readily used in Python and Julia. The application is the build process of PHCpack, a free and open source software package to solve polynomial systems by homotopy continuation methods, written mainly in Ada, with components in C++, available at github at `https://github.com/janverschelde/PHCpack`.

1 Language Agnostic Computing

This paper describes interface development from the perspective of an Ada programmer, aimed to export the functionality of a software package to Python [1] and Julia [2] computational environments, available through Jupyter notebooks [3]. The Jupyter notebook is the interface to SageMath [4], a free open source system for mathematical computing.

In order to export all functionality the interface passes through C, which may be regarded as a least common multiple of programming languages, as Ada, Python, and Julia share enough common ground to enable language agnostic computing, as Jupyter stands for Julia, Python, R, and many others.

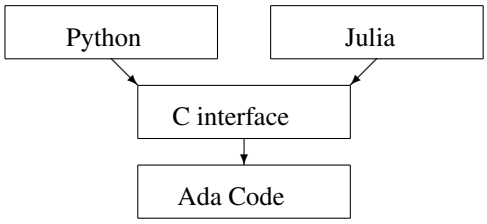

Figure 1: C as the least common multiple language.

The main point is to automate building with GPRbuild.

2 GPRbuild and Interface Development

The mixed language development is supported by GPRbuild, the project manager of the gnu-ada compiler GNAT. The build process, defined via library projects, results in shared object files (with the extension `.so` on Linux, `.dll` on Windows, and `.dylib` on Mac OS X). These shared object files can be called directly from a Python script or a Julia program.

In the C interface layer, the control is passed to a C program. The C program passes input data to some Ada procedure, calls an exported procedure, and extracts the output data via another call to an Ada procedure. The most basic and versatile manner to pass data is via a plain sequence of characters of 32-bit integers. As the *hello world* for this interface, consider the swapping of characters in a string.

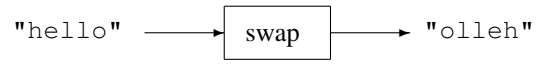

Figure 2: Swapping characters via an interface package.

The interface package as shown in Figure 3 exports a procedure to pass the input data, the `DoIt` procedure to compute the output data, and then a third function to return the output.

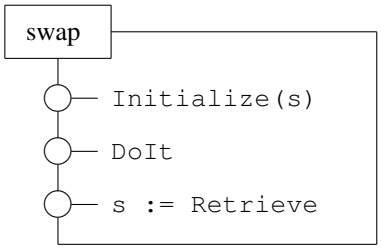

Figure 3: An interface package to swap characters in a string.

Then the C program calls the function `call_swap`, declared in Ada as below.

```
with C_Integer_Arrays;
 use C_Integer_Arrays;

function call_swap
  ( jobnbr : integer;
    sizedata : integer;
    swapdata : C_intarrs.Pointer;
    verbose : integer ) return integer;
```

where the package `C_Integer_Arrays` defines `C_Integer_Array` as an array of C integers, of type `Interfaces.C.int`. The package contains

```
  package C_intarrs is
    new Interfaces.C.Pointers
      (Interfaces.C.size_T,
       Interfaces.C.int,
       C_Integer_Array,0);
```

Observe that the `void` idiom of C is avoided. The details of this introductory project are posted at `github.com/janverschelde/ExportAdaGPRbuild`.

The C code to test takes a string `word`, converts the string into an array of 32-bit integers, and then calls the Ada code:

```
sizeword = strlen(word);

for(int idx = 0; idx < sizeword; idx++)
   dataword[idx] = (int) word[idx];

adainit();
fail = _ada_call_swap(0,sizeword,dataword,1);
fail = _ada_call_swap(1,sizeword,dataword,1);
fail = _ada_call_swap(2,sizeword,dataword,1);
adafinal();

for(int idx = 0; idx < sizeword; idx++)
   word[idx] = (char) dataword[idx];
```

The contents of the file `demo.gpr` defines the build of the C test program.

```
project Demo is

    for Languages use ("Ada", "C");

    for Source_Dirs use ("src");

    for Main use
    (
        "hello_world.adb",
        "main.adb",
        "test_call_swap.c"
    );

    for Object_Dir use "obj";

    for Exec_Dir use "bin";

end Demo;
```

To make a shared object file, a `library project` is defined. Below are the essentials of the instructions to make the `libdemo` as a shared object.

```
for Library_Dir use "lib";
for Library_Name use "demo";
for Library_Kind use "dynamic";
for Library_Auto_Init use "true";
for Library_Interface use
(
   "hello_world", "main", "swap",
   "call_swap", "c_integer_arrays"
);
for Library_Standalone use "encapsulated";

package Compiler is

  for Switches ("call_swap.adb") use ("-c");

end Compiler;

package Binder is

  -- use "-Lada" for adainit and adafinal
  for Default_Switches ("Ada")
     use ("-n", "-Lada");

end Binder;
```

Julia has the function `ccall()` to execute compiled C code. The Julia code below calls the `call_swap` procedure.

```
LIBRARY = "../Ada/lib/libdemo"

word = [Cint('h'), Cint('e'), Cint('l'),
        Cint('l'), Cint('o')]
println(word)
ptr2word = pointer(word, 1)
p = ccall((:_ada_call_swap, LIBRARY), Cint,
           (Cint, Cint, Ref{Cint}, Cint),
            0, 5, ptr2word, 1)
p = ccall((:_ada_call_swap, LIBRARY), Cint,
           (Cint, Cint, Ref{Cint}, Cint),
            1, 5, ptr2word, 1)
p = ccall((:_ada_call_swap, LIBRARY), Cint,
           (Cint, Cint, Ref{Cint}, Cint),
            2, 5, ptr2word, 1)
println(word)
```

The string `"hello"` is represented by `Int32[104, 101, 108, 108, 111]`. The last `println(word)` shows `Int32[111, 108, 108, 101, 104]`.

To extend Python code, an extension module must be defined in C or C++. The `setup.py` script has the list `extra_objects` to define the location of the compiled Ada code and the location of the Ada runtime libraries. The shared object made running `python setup.py build_ext` can then be directly imported in a Python session. The making of this extension can be done without makefiles.

3 Building PHCpack

As a demonstration to a large scale project, GPRbuild is applied to make share objects for PHCpack, a free and open source software package to solve polynomial systems with homotopy continuation. The python interface to PHCpack is phcpy [5]. Written mainly in Ada, PHCpack contains MixedVol [6] and DEMiCs [7] to count bounds on the number of isolated solutions fast. For MixedVol, a translation into Ada was made. The package DEMiCs is written in C++ and incorporated into PHCpack as such. As described in [8], the code for multiple double precision is provided by QDlib [9] and CAMPARY [10].

A Julia interface is under development. From the `Julia` folder of the PHCpack source distribution, running the Julia program `version.jl` at the command prompt:

```
$ julia version.jl
-> in use_c2phc4c.Handle_Jobs ...
PHCv2.4.85 released 2021-06-30
$
```

The `ccall()` uses the `libPHCpack` shared object, made with GPRbuild.

Acknowledgements

Supported by the National Science Foundation under grant DMS 1854513.

The author thanks Dirk Craeynest and Fernando Oleo Blanco for the organization of the Ada Devroom at FOSDEM 2022.

References

[1] F. Pérez, B. Granger, and J. Hunter, "Python: An ecosystem for scientific computing," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 12–21, 2011.

[2] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[3] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. D. Team, "Jupyter Notebooks—a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents, and Agendas* (F. Loizides and B. Schmidt, eds.), pp. 87–90, IOS Press, 2016.

[4] W. Stein, "Sage: Creating a viable free open source alternative to Magma, Maple, Mathematica, and MATLAB," in *Foundations of Computational Mathematics, Budapest 2011* (F. Cucker, T. Krick, A. Pinkus, and A. Szanto, eds.), vol. 403 of *London Mathematical Society Lecture Note Series*, pp. 230–238, Cambridge University Press, 2012.

[5] J. Otto, A. Forbes, and J. Verschelde, "Solving polynomial systems with phcpy," in *Proceedings of the 18th Python in Science Conference*, pp. 563–582, 2019.

[6] T. Gao, T. Y. Li, and M. Wu, "Algorithm 846: MixedVol: a software package for mixed-volume computation," *ACM Trans. Math. Softw.*, vol. 31, no. 4, pp. 555–560, 2005.

[7] T. Mizutani and A. Takeda, "DEMiCs: A software package for computing the mixed volume via dynamic enumeration of all mixed cells," in *Software for Algebraic Geometry* (M. Stillman, N. Takayama, and J. Verschelde, eds.), vol. 148 of *The IMA Volumes in Mathematics and its Applications*, pp. 59–79, Springer-Verlag, 2008.

[8] J. Verschelde, "Parallel software to offset the cost of higher precision," *ACM SIGAda Ada Letters*, vol. 40, no. 2, pp. 59–64, 2020.

[9] Y. Hida, X. S. Li, and D. H. Bailey, "Algorithms for quad-double precision floating point arithmetic," in *15th IEEE Symposium on Computer Arithmetic (Arith-15 2001)*, pp. 155–162, IEEE Computer Society, 2001.

[10] M. Joldes, J.-M. Muller, V. Popescu, and T. W., "CAMPARY: Cuda Multiple precision arithmetic library and applications," in *Mathematical Software – ICMS 2016, the 5th International Conference on Mathematical Software*, pp. 232–240, Springer-Verlag, 2016.