Mixed language programming - C++11 and Fortran 2008

My programming ramblings

Posted on May 11, 2012 by Paul

The code for this tutorial is on GitHub: https://github.com/sol-prog/mix_fortran_cpp.

Different programming languages have different strengths and while you can express any imaginable algorithm in C++11 sometimes you need to interface your code with legacy codes written in Fortran, or you want to use modern Fortran for his rich matrix operations. This post will present some simple examples of mixed Fortran 2008 and C++11 code, as a side note it is entirely possible to write these examples in pure Fortran or C++, coding every operations from scratch in C++ or using matrix libraries like Eigen.

Mixing Fortran with C (or C++) was a painful experience in the past and the exact mechanism was compiler dependent. Starting with Fortran 2003 you can use a standardized mechanism for interoperability with C, calling Fortran from C and vice versa can be done using the *iso_c_binding* Fortran module.

Let's start by writting a C++11 code that creates two N by N arrays filled with random numbers. In this post we use the column-major order storage (Fortran order) and not the typical row-major storage used in C or C++ for multidimensional arrays.

```
#include <iostream>
#include <vector>
#include <random>
#include <ctime>

using namespace std;

//Fill a vector with random numbers in the range [lower, upper]
void rnd_fill(vector<double> &V, double lower, double upper) {

    //use system clock to create a random seed
    size_t seed (clock());

    //use the default random engine and an uniform distribution
    default_random_engine eng(seed);
    uniform_real_distribution<double> distr(lower, upper);

    for( auto &elem : V){
        elem = distr(eng);
    }
}

int main() {
    size_t N = 3;
    vector<double> A(N * N), B(N * N);

    //Fill A and B with random numbers in the range [0,1]:
    rnd_fill(A, 0.0, 1.0);
    rnd_fill(B, 0.0, 1.0);

    return 0;
}
```

The *iso_c_binding* Fortran module defines numerical types compatible with C and C++ numerical types, e.g. a C *int* is defined as *c_int* and a C *double* as *c_double*. We are going to implement a Fortran subroutine, callable from C, that multiplies two matrices filled with doubles. For matrix multiplication we are going to use a Fortran intrinsic function named *matmul*:

```
subroutine matrix_multiply(A, rows_A, cols_A, B, rows_B, cols_B, C, rows_C, cols_C) bind(c)
    use iso_c_binding
    integer(c_int) :: rows_A, cols_A, rows_B, cols_B, rows_C, cols_C
    real(c_double) :: A(rows_A, cols_A), B(rows_B, cols_B), C(rows_C, cols_C)
```

```
5
6     C = matmul(A, B)
7 end subroutine matrix_multiply
```

Please note the use of *bind(c)* on line 1 and *iso_c_binding* module on line 2, the actual matrix multiplication is done on line 6.

In order to be able to use the above Fortran subroutine in C we need to translate Fortran's subroutine definition to a C function:

```
1 subroutine matrix_multiply(A, rows_A, cols_A, B, rows_B, cols_B, C, rows_C, cols_C) bind(c)
```

became:

```
1 void matrix_multiply(double *A, int *rows_A, int *cols_A, double *B, int *rows_B, int *cols_B, double
```

Please note that every variable is declared as a pointer, this is because Fortran uses call by reference by default.

What about a C++ definition for *matrix_multiply* ? We can use the C definition wrapped in an *extern "C"* declaration:

```
1 extern "C" {
2     void matrix_multiply(double *A, int *rows_A, int *cols_A, double *B, int *rows_B, int *cols_B, do
3 }
```

Now, we can modify our original C++ code in order to use *matrix_multiply* for multiplying A and B:

```
 1 #include <iostream>
 2 #include <vector>
 3 #include <random>
 4 #include <ctime>
 5
 6 using namespace std;
 7
 8 //Fortran subroutine definition "translated" to C++
 9 extern "C" {
10     void matrix_multiply(double *A, int *rows_A, int *cols_A, double *B, int *rows_B, int *cols_B, d
11 }
12
13 //Fill a vector with random numbers in the range [lower, upper]
14 void rnd_fill(vector<double> &V, double lower, double upper) {
15
16     //use system clock to create a random seed
17     size_t seed (clock());
18
19     //use the default random engine and an uniform distribution
20     default_random_engine eng(seed);
21     uniform_real_distribution<double> distr(lower, upper);
22
23     for( auto &elem : V){
24         elem = distr(eng);
25     }
26 }
27
28 //Print matrix V(rows, cols) storage in column-major format
29 void print_matrix(vector<double> const &V, int rows, int cols) {
30
31     for(int i = 0; i < rows; ++i){
32         for(int j = 0; j < cols; ++j){
33             std::cout << V[j * rows + i] << " ";
34         }
35         std::cout << std::endl;
36     }
37     std::cout << std::endl;
38 }
39
40 int main() {
41     size_t N = 3;
42     vector<double> A(N * N), B(N * N), C(N * N);
43
44     //Fill A and B with random numbers in the range [0,1]:
```

```
45     rnd_fill(A, 0.0, 1.0);
46     rnd_fill(B, 0.0, 1.0);
47
48     //Multiply matrices A and B, save the result in C
49     int sz = N;
50     matrix_multiply(&A[0], &sz, &sz, &B[0], &sz, &sz, &C[0], &sz, &sz);
51
52     //print A, B, C on the standard output
53     print_matrix(A, sz, sz);
54     print_matrix(B, sz, sz);
55     print_matrix(C, sz, sz);
56
57     return 0;
58 }
```

Please note the syntax used at line 50 for sending a C++ STL vector to our Fortran subroutine by reference.

The above codes can be compiled with gcc-4.7 on Linux and Mac or with Visual Studio and Intel Fortran on Windows.

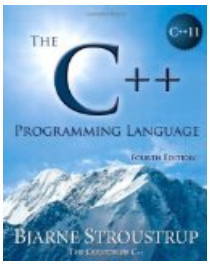
If you save the Fortran code in *fortran_matrix_multiply.f90* and C++ in *cpp_main_1.cpp* you can compile and link them on Unix:

```
 1 sols-MacBook-Pro:mix_fortran_cpp sol$ gfortran-4.7 -c fortran_matrix_multiply.f90
 2 sols-MacBook-Pro:mix_fortran_cpp sol$ g++-4.7 -c -std=c++11 cpp_main_1.cpp
 3 sols-MacBook-Pro:mix_fortran_cpp sol$ gfortran-4.7 fortran_matrix_multiply.o cpp_main_1.o -lstdc++ -
 4 sols-MacBook-Pro:mix_fortran_cpp sol$ ./mix_example.out
 5 0.411975 0.886183 0.041323
 6 0.130066 0.576865 0.507137
 7 0.26862 0.938682 0.385561
 8
 9 0.464278 0.317101 0.3179
10 0.799268 0.0544081 0.744739
11 0.0202931 0.0940126 0.447147
12
13 0.900407 0.182738 0.809419
14 0.531748 0.120308 0.697726
15 0.882796 0.172499 0.956869
16
17 sols-MacBook-Pro:mix_fortran_cpp sol$
```

On Windows open an Intel Fortran command prompt and write these two lines for compiling and linking the codes:

```
1 ifort -c fortran_matrix_multiply.f90
2 cl fortran_matrix_multiply.obj cpp_main_1.cpp.cpp
```

If you are interested in learning more about the new C++11 syntax I would recommend reading The C++ Programming Language by Bjarne Stroustrup.



or, Professional C++ by M. Gregoire, N. A. Solter, S. J. Kleper 4th edition:

If you need to brush your Fortran knowledge a good book is Modern Fortran Explained by M. Metcalf, J. Reid and M. Cohen: