DYNAMIC LIBRARIES FOR MATLAB

Michal Dobeš
Department of Computer Science, Faculty of Science,
Palacky University Olomouc, Czech Republic

Abstract

Matlab enables to call external functions created in different programming languages like C, C++ or Fortran. A part of a program code or a function can be written in another programming language, for example C++. It is useful especially, if the code is already optimized for speed. This paper introduces how to create a DLL containing the function that can be called from Matlab. Dynamic libraries were successfully used in our iris recognition project.

Introduction

Matlab offers many functions and libraries – toolboxes to develop applications. Sometimes there is a need to include a source code written in other programming language. In this case Matlab enables to call external functions created in different programming languages. Communication with Java programs is simple – Java Virtual Machine is installed during the Matlab installation. You can run and access Java objects within the Matlab environment. (Java version 1.3.1 is installed with Matlab 6.5.)
Communication with a subroutine (or a function) written in C, C++ or Fortran is slightly different. You must create a Matlab Executable file (MEX-file). On Windows, it is created as a dynamic library (DLL) file and it contains the desired routine (function). It can be called from Matlab as if it were a built in function. Matlab functions are optimized for performance, and usually work effectively. In some cases a code written in C++ can be performed more effectively, especially if the code contains loops that were optimized for speed. The other reason can be that the code was already written and was debugged in C or C++.
We used Microsoft Visual C++, and Windows 2000, so the following text refers to this configuration.

How to create a DLL from a C/C++ code

The following steps must be performed to create a Matlab Executable file:

1. Setup appropriate compiler.
2. Write the desired function (say, in C++) according to the specific rules described later.
3. Compile the function into a Matlab executable, i.e. MEX file.

The first step is easy if a version 6 or above of Matlab is used. You can use a C++ compiler which is ANSI C and is able to create dynamically linked libraries. A Matlab built in compiler Lcc is also available. If there is more compilers installed on your computer it is possible to choose what compiler will be used to compile a source code into Matlab executable. To set up the required compiler you type: “mex –setup” from the Matlab prompt and choose the appropriate options.

Then a C/C++ code is to be prepared so that it can be compiled into a MEX file. The name of an executable file must be the same as the name by which it is called from Matlab (similarly like “.m” files). Unlike .m files that are platform independent, MEX files are platform dependent and have a different extension: “.dll” for Windows, “.mexglx” for linux.

The C/C++ source must contain a header "mex.h". (See the example of a program). A must contain a function which is called "mexFunction" and which acts as an interface to Matlab. The mexFunction enables to pass the input and output arguments. (Functions prefixed with mex perform the communication in the MATLAB environment).

The mexFunction contains four arguments:
nlhs – the number of left hand side arguments (number of expected output mxArrays)
plhs – the pointer to the output array (originally set to the array of NULL pointers)
nrhs – the number of right hand side input arrays (number of expected input mxArrays)
prhs – the pointer to the right hand side input array.

Variables, vectors, matrices, and other objects are stored as the only object type - the MATLAB array. In C, the MATLAB array is declared as an mxArray. Routines that manipulate with Matlab arrays have a mx prefix and allow you to create and access the mxArrays. The type and number of the arguments passed through the mexFunction should be checked. For example, you use mxIsDouble function that returns Boolean true if the expected type is double, i.e.: isDouble = mxIsDouble(prhs[0]).
To acquire the pointer and the size of the input argument(s) you use mxGet functions. In this example, the pointer to the first input argument is of a type double:
double *in1;
in1 = mxGetPr(prhs[0]);
rows1 = mxGetM;
cols1 = mxGetN;

To assign and pass the output value(s) you need to create the output array. Suppose you have one output argument – a matrix of the size m x n and type double, you use:
double *out;
plhs[0] = mxCreateDoubleMatrix(m, n, mxReal);
and get the pointer to the beginning of the array where the output matrix is stored:
out = mxGetPr(plhs[0]);
You specify the order of an element in the Matlab array rather then a row and column to access the values in the created matrix. The elements are stored column by column in the array, and they are counted from zero, i.e. first column, second column ..., see the "Example of a source code". For more details about mex and mx functions, see the "On line manuals for Matlab" [Mat02].

Example of a source code

A function takes three arguments, and performs 2D bilinear interpolation according to [Ver91] (similar to 'interp2' in Matlab which is used when resizing images).

```
// Example of a bilinear interpolation as a MEX file
#include "mex.h"

void mexFunction( int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[] )
{
double *in1,*in2, *in3, *out;
int rows1, cols1, rows2, cols2;

// Check number of arguments
if (nrhs != 3) {
	mexErrMsgTxt("Bilinear requires three input arguments.");
} else if (nlhs > 1) {
	mexErrMsgTxt("One output argument required");
}

// check the type of arguments
   if (mxIsDouble(prhs[0]) && mxIsDouble(prhs[1]) && mxIsDouble(prhs[2]))
   {
         rows1 = mxGetM(prhs[0]); // number of rows
         cols1 = mxGetN(prhs[0]); // number of collumns
         rows2 = mxGetM(prhs[1]); // ...
         cols2 = mxGetN(prhs[1]); // ...

         in1 = mxGetPr(prhs[0]);  // get a pointers to the first input argument
         in2 = mxGetPr(prhs[1]);  // ...
         in3 = mxGetPr(prhs[2]);  // ...

         // create the output
         plhs[0] = mxCreateDoubleMatrix(rows2, cols2, mxREAL);
         // get a pointer to the beginning
         out = mxGetPr(plhs[0]);

         // compute the bilinear interpolation ...
         double p,q,a,b,c,d;
         int position;

         for (int i=0; i<rows2; i++)        {
                 for (int j=0; j<cols2; j++){

                         // elements of a matrix in the array are stored collumn by column so
                         position = j*rows2+i;

                         /*  Compute bilinear interpolation [Ver91]  + respect the border points ...
                                 ...
                         */
                         // assign the output value:
                         *(out+position) = a*p + b*q + c*p*q + d;

                 }; //for j
         }; //for i
   }
   else { mexErrMsgTxt("Arguments must be of type double.");
};
return;
} ;
```

Translation of a source code into a MEX file is performed from the Matlab environment by typing: “mex bilinear.cpp”. The resulting file is stored in a bilinear.dll file. The function bilinear is then called from the Matlab environment in the same way as an .m file, i.e. bilinear(a, u, v).

Conclusion

Functions written in C/C++ may be optimized for speed, especially when loops are used in the computation. In this case the function translated into a .dll may run faster then a code written as an .m file. DLL files were successfully used in the iris recognition project [Dob02, Dob03]. In some cases [Dar99, Dar00] the speed of the function called from the .dll was three times faster then the .m file.

References

[Dob03] Dobeš, M. Rozpoznávání obrazu: neurčitost a informace. Matematika-fyzika-informatika. Prometheus s.r.o ve spolupráci s Jednotou českých matematiků a fyziků a s podporou MŠMT, Praha 2003, Vol. 13, No. 1, p. 42-49. ISSN - 1210-1761

[Dar00] Darbellay, G.A., and Tichavský, P., Independent component analysis through direct estimation of the mutual information. ICA'2000 Proc., Second International Workshop on Independent Component Analysis and Blind Signal Separation, Helsinki, Finland, 19.-22.6. 2000, pp. 69–75.

[Dar99] Darbellay, G.A., and Vajda, I., Estimation of the information by an addaptive partitioning of the observation space. *IEEE Trans. Information Theory*, 45(4):1315-1321.

[Dob02] Dobeš, M. and Machala, L. 2002. The database of Iris Images: `http://www.inf.upol.cz/Iris`

[Mat02] On-line manuals: Matlab The Language of Technical Computing, Notes for Release 13, Math Works 2002.

[Sed92] Sedgevick, R. Algorithms in C++, Princeton University, Addison-Wesley 1992

[Ver91] Vernon, D. Machine Vision, Automated Visual Inspection and Robot Vision, Prentice Hall International (UK) Ltd., 1991.

Contact Information

Computer Science Department, Faculty of Science, Palacky University Olomouc,
Tomkova 40, 77900, Olomouc, Czech Republic.
E-mail: michal.dobes@upol.cz