The MATLAB Compiler Suite: M-Files to C/C++ Executable Programs

Ray Bachnak
Texas A&M University-Corpus Christi

Roger Lee
Naval Air Systems Command

Abstract

MATLAB has been recognized as the premier program for numerical computations and data visualization. Due to the fact that MATLAB is an interpreted language, M-files execute slower than compiled programs written in other languages. Furthermore, M-files require the presence of MATLAB to run. Recently, The MathWorks introduced a set of tools to automatically convert M-files into C/C++ source code, which can then be compiled to produce stand-alone executable code. This paper discusses the features of the Compiler, describes the conversion process, and presents test results that compare execution times of M-files and C/C++ executable applications.

Introduction

MATLAB is a powerful language that simplifies the process of solving technical problems in a variety of disciplines by including a very extensive library of predefined functions. It also offers many special-purpose toolboxes that address specific areas, such as signal processing and neural networks, and provides Graphical User Interface (GUI) tools that make it suitable for application development. Because of these capabilities, MATLAB has been recognized as the premier program for numerical computations and data visualization and has been used by the educational, industrial, and government communities all over the world.

Due to the fact that MATLAB is an interpreted language, MATLAB files (M-files) execute slower than compiled programs written in other languages, such as C, C++, and Fortran. In addition, M-files require the presence of MATLAB to run. To address these weaknesses, The MathWorks recently introduced a set of tools that automatically convert M-files into C/C++ source code, which can then be compiled to produce C/C++ executable programs that can run even if MATLAB is not installed on the system. These C/C++ executable programs are referred to as stand-alone applications. The tools required for generating stand-alone applications are: MATLAB, the MATLAB Compiler, the MATLAB C/C++ Math Library, and a C/C++ compiler. The MATLAB C/C++ Graphics Library is also required in order to create applications that make use of Handle Graphics® functions. This paper describes the important features of the Compiler and its associated tools, states the benefits of converting M-files to stand-alone applications, and presents examples that illustrate the steps involved in the conversion process.

The MATLAB® Compiler

The MathWorks uses the phrase “MATLAB Compiler,” with a capital letter C, to refer to the product that translates M-files to C or C++ source code [1]. The word "compiler" with a lowercase c is used to refer to the C or C++ compiler. The MATLAB® Compiler version 2.1 may be used to take M-files as input and produce the following outputs: C source code for building MEX-files, C or C++ source code for combining with other modules to form stand-alone applications, C code S-functions for use with Simulink®, or code for C shared libraries and C++ static libraries. The four main reasons to compile M-files are: (1) To speed up the execution of programs, (2) To create stand-alone applications, (3) To create C shared libraries or C++ static libraries, and (4) To hide proprietary algorithms. The current Compiler supports most of the functionality of MATLAB, however, it cannot compile script M-files, function M-files that call a script, and function M-files that use objects.

The MATLAB® C/C++ Graphics Library

The C/C++ Graphics Library is a collection of graphics routines that make the MATLAB plotting and visualization capabilities available to stand-alone C and C++ applications [4]. The library contains more than 100 routines, including MATLAB 6.0 built-in graphics functions, such as surf, plot, get, and set, and some commonly used M-file graphics functions, such as newplot, gcf, gca, gco, and gcbf. Using the graphics library with the Compiler and the C/C++ Math Library makes it possible to compile M-files that include lines, text, meshes, and polygons, as well as graphical user interface (GUI) components such as menus, push buttons, and dialog boxes.

The C/C++ Graphics Library does not support some MATLAB features, including MATLAB objects, MATLAB Java objects, plotedit command, and propedit command. In addition, the graphics library supports only a subset of print command switches to specify device drivers. The C/C++ Graphics Library is available for PCs running Microsoft Windows or Linux, Sun, HP, SGI, and Compaq Alpha UNIX platforms. The process of creating C/C++ applications is similar for both PC and UNIX machines. To create a graphics application, the Compiler is used along with the Graphics Library bundle file. Bundle files are ASCII text files that contain Compiler command line options and arguments. For example, to compile an M-file called test1.m into a C application, the following command is entered the MATLAB prompt.

```
mcc -B sgl test1
```

This command invokes the Compiler, using the -B flag to specify the bundle file used to create C stand-alone graphics applications, sgl. The first time the Compiler is run to create a stand-alone graphics application, it creates a subdirectory, named \bin, in the current working directory. The command for a C++ application is mcc –B sglcpp fname.

Supported Compilers

On Unix workstations, the following compilers are supported with some limitations [4]: The GNU C compiler, gcc, (except on HP and SGI64), the system's native ANSI C compiler on all UNIX platforms, the system's native C++ compiler on all UNIX platforms (except Linux), and the GNU C++ compiler, g++, on Linux. On PC workstations, the following ANSI C and C++ PC Compilers are supported: Lcc C version 2.4 (included with MATLAB), Watcom C/C++ versions 10.6 and 11.0, Borland C++ versions 5.0, 5.2, 5.3, 5.4, and 5.5, and Microsoft Visual C++ (MSVC) versions 5.0 and 6.0. Selecting a C/C++ compiler is accomplished by typing "mbuild -setup <RET>" then following the instructions.

Creating C/C++ Executable Code

A MathWorks utility, mbuild, provides an easy way to specify an options file that can be used to set the compiler and linker settings, change compilers or compiler settings, switch between C and C++ development, and build an application. The Compiler (mcc) automatically invokes mbuild under certain conditions. In particular, mcc -m or mcc –p invokes mbuild to perform compilation and linking. To prevent mcc from invoking mbuild automatically, the -c option can be used. For example, mcc -mc filename. On systems where there is exactly one C or C++ compiler, the mbuild utility automatically configures itself for the appropriate compiler. On systems where there is more than one C or C++ compiler, the mbuild utility lets the user select which of the compilers to use. Once a C or C++ compiler is selected, that compiler becomes the default compiler. The user may later specify another compiler by following the same procedure.

When you use the Compiler to compile an M-file, it generates C or C++ code, a header file, and a wrapper file. The C or C++ code and the header file are independent of the final target type and target platform. That is, the C or C++ code and header file are identical no matter what the desired final output (i.e. MEX-functions, C/C++ applications, or libraries). The wrapper file, however, provides the code necessary to support the output executable type. Consider the M-file called "PolyVal" shown in Fig. 1.

```
%***************************************************************
% This program evaluates the polynomial y=x^4 + 2x^3 - x^2 + 4x - 5  at  x=5, 6
%***************************************************************
function y=PolyValue(poly,x)
poly=[1 2 -1 4 -5];
x=[5, 6];
y=polyval(poly, x)
```

Fig. 1 M-file to compute the value of a polynomial

This program evaluates a polynomial at two given points. Typing PolyVal at the MATLAB prompt yields

```
ans =

     865       1711
```

To create an executable C file, the mcc command is used at the MATLAB prompt as follows

```
mcc -m PolyValue
```

This mcc command generates the following files:

```
PolyValue.h
PolyValue_main.c
bin
PolyValue.c
PolyValue.exe
```

PolyValue.h contains the public information.
PolyValue_main.c contain .EXE function interface (wrapper).
bin is a directory containing the MATLAB menu bar and toolbar figure files.
PolyValue.c is the C source code.
PolyValue.exe is the executable application.

Note that when executing the mcc command to link files and libraries, mcc actually calls the mbuild script to perform the functions. To run the C application, PolyValue, invoke it by typing its name on the MS-DOS command line. However, MATLAB offers a quick way to shell out to DOS by using the bang (!) function. The exclamation point character, "!", is a shell escape and indicates that the rest of the input line is a command to the operating system. It is used to invoke utilities or run other programs without quitting from MATLAB. At the MATLAB prompt type

```
!PolyValue
```

The application should run and display the same exact results as before. After quitting the program, the operating system returns control to MATLAB.

Inputting From the Keyboard and Reading Data Files

It is often necessary to create interactive programs that allow the user to enter input from the keyboard. Furthermore, in many applications, the input must come from previously generated data files stored in binary or MATLAB format. Consider the M-file program of Fig. 2. This program, ReadDataFile, loads input data "signal" from a mat-file then calls a function, DispImg, to display the imag of the data, "signal".

```
%********************************************************
% This programs loads data from an input file and displays the image      *
%********************************************************
fname = input('Enter input file name (without extension [.mat]): ', 's');
tic
%load input data from matlab file *.mat
disp('STARTING: Load input data ...');
eval(['load ' fname]);
cuttail=0;                %as defined in function Dispimg
dispimg(signal,'LIN',cuttail,0);  %call dispimg
toc                %stop timer
```

Fig. 2 Loading a data file and displaying image

This program can be run at the MATLAB prompt by typing ReadDataFile as follows.

```
>>ReadDataFile <enter>
>>Enter input file name (without extension [.mat]): test3 <enter>
```

As stated earlier, the Compiler does not support the INPUT function to manipulate workspace environment. Therefore, the Compiler can not correctly compile the program of Fig. 2. To get around this restriction, one can manually edit the C code generated by the Compiler. This approach, however, is inefficient and not practical. The C/C++ Graphics Library offers an input dialog box MATLAB function, INPUTDLG, which can be used to automatically solve this problem. The above example may be modified as shown in Fig. 3.

```
%********************************************************
% This programs loads data from an input file and displays the image      *
%********************************************************
function ReadDataFile2
% prompt for input file name
prompt={'Enter input file name (without extension [.mat]): '};
tic                %start timer
title2='Input File name';
lines=1;
defaultfn={'test250'};
fname1 = inputdlg(prompt, title2, lines, defaultfn);
fname=fname1{1}     %extracting the first element in the array
signal=loadinput(fname);  %call loadinput to read file
cuttail=0;                %as defined in function Dispimg
dispimg(signal,'LIN',cuttail,0);  %call dispimg
toc                %stop timer
```

Fig. 3 Loading a data file and displaying image

The modified program consists of a main function, ReadDataFile2, which calls two functions, LoadInput and DispImg. Creating a stand-alone application by writing the source code in more than one M-file is very practical. This approach allows the programmer to take advantage of MATLAB's interpretive development environment. After getting the M-file version of the program to work properly, the code is compiled and converted into a C application. We show in Fig. 4 the function that reads the input data file as requested by ReadDataFile2, LoadInput.

```
%************************************
%This function reads an input data file (.mat)    *
%************************************
function signal=loadinput(fname1)
% Reading input signal
load(fname1, 'signal');
```

Fig. 4 Function to read an input data file

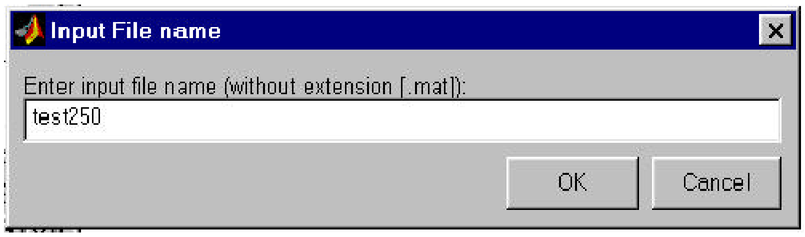
Since ReadDataFile2 uses a graphics library function, the graphics library must be included when compiling the file as follows

```
mcc -B sgl ReadDataFile2 dispimg loadinput
```

This command creates an executable file, ReadDataFile2.exe, which can be run at the MATLAB prompt by typing

```
!ReadDataFile2
```

Running the program produces the following input box


Input File name

Enter input file name (without extension [.mat]):

test250

Instead of the default “test250”, the user can type in a desired file name. After clicking on OK, the program produces the same exact results as before (except a different execution time).

Creating and Testing a Stand-Alone Real Application

The examples presented so far require little time to execute. To illustrate the benefits of the Compiler and its associated libraries, a program that implements the Range Migration Algorithm for Synthetic Aperture Radar (SAR) image formation was used.

The program, which is not shown here, consists of a main function (rma8c) and six subfunctions. The executable code is produced by typing

```
mcc -B sgl rma8c loadinput dispimg localmax stolt_int2 resol3db invpaint
```

The program was run at the MATLAB prompt by typing

```
!rma8c
```

To compare the execution speed of M-files, MEX files, and EXE files, rma8c was run on a 1GHz machine with these three types. The upper part of Table 1 summarizes the results.

Table 1. Comparison of execution times

			Execution time (seconds)			
Machine	File	Run #	M-File	MEX-file	.EXE C	Ratio (.EXE/M-file
1 GHz PC, 128 MB RAM	Rma8c	1	734.24	397.43	328.40	0.447
		2	739.15	405.51	331.26	0.447
1 GHz PC, 128 MB RAM	Rma8cs	1	688.55	339.99	306.48	0.445
		2	690.31	338.67	305.39	0.442

A shorter version of Rma8c, Rma8cs, was created by removing code that generates several intermediate graphs. Rma8cs was run on the same machine and the results are shown in the lower part of Table 1.

Summary of Commands

The following summarizes the use the MATLAB Compiler to generate some of the most common results. Note that the -g option may be added to any of these commands for debugging purposes.

Converting from MATLAB to MEX Files
The command to convert an M-file into C and create the corresponding MEX file that runs at the MATLAB prompt is: mcc –x mainfn sub1fn sub2fn

Converting from MATLAB to EXE Files (C compiler)
The command to convert an M-file into C and to create a corresponding stand-alone executable file that can be run without MATLAB is: mcc –m mainfn sub1fn sub2fn

The command to convert an M-file that contains Handle Graphics functions into C and to create a corresponding stand-alone executable file that can be run without MATLAB is: mcc –B sgl mainfn sub1fn sub2fn

Converting from MATLAB to EXE Files (C++ compiler)
The command to convert an M-file into C++ and to create a corresponding stand-alone executable file that can be run without MATLAB is: mcc –p mainfn sub1fn sub2fn

The command to convert an M-file that contains Handle Graphics functions into C++ and to create a corresponding stand-alone executable file that can be run without MATLAB is: mcc –B sglcpp mainfn sub1fn sub2fn

Converting from MATLAB to C Code
The command to produce a C file from an M-file or function is: mcc –mc mainfn sub1fn sub2fn. The command to produce a EXE file from a C file: mbuild mainfn sub1fn sub2fn.

Conclusion

This paper discusses the major features of the MATLAB Compiler suite and presents examples that illustrate the steps involved in converting M-files to C/C++ executable programs. Test results show that execution time of .EXE files is less than 50% of that of M-files. While the Compiler offers an easy way to convert M-files to stand-alone applications, the programmer should be aware that it has some major limitations.

Bibliography

1. The MATLAB Compiler User's Guide Version 2.1 (264 pages)
2. The MATLAB C Math Library User's Guide Version 2.1 (332 pages)
3. The MATLAB C Math Library Reference Version 2.1 (429 pages)
4. The MATLAB C/C++ Graphics Library Version 2.0 (52 pages)

Biography

RAFIC BACHNAK
Rafic (Ray) Bachnak is an Associate Professor of Engineering Technology at Texas A&M University-Corpus Christi. He received his B.S., M.S., and Ph.D. degrees in Electrical and Computer Engineering from Ohio University in 1983, 1984, and 1989, respectively. Dr. Bachnak was previously on the faculty of Franklin University and Northwestern State University.

ROGER LEE
Roger (Ru-Ying) Lee is a senior scientist in the RF sensors division, Avionics Department of Naval Air Warfare Center at Patuxent River, Maryland. He received his MS degree in EE from Penn State Univ. and Ph.D. in Mathematics from University of Penn. His area of research is in signal processing and image processing, in particular, in the Synthetic Aperture Radar.