Compiling UMFPACK

Evgenii B. Rudnyi, 2008, (c) All rights reserved

http://MatrixProgramming.com

Introduction

UMFPACK is Unsymmetric MultiFrontal direct solver for sparse matrices developed by Prof Timothy A. Davis:

http://www.cise.ufl.edu/research/sparse/**umfpack**/

The goal of this chapter is consider how to compile **UMFPACK** under gcc. I will use gcc under Cygwin but the procedure should be the same on Linux. Additionally I will describe how to use the **UMFPACK** library compiled with gcc under Cygwin from within Microsoft Visual Studio.

The plan is as follows. First we compile **UMFPACK** without BLAS. This is a good starting point for those who are new to Unix environment. The next step is to compile **UMFPACK** with the optimised BLAS in order to reach the best efficiency of the library (I will employ ATLAS). The two final steps are for people working on Windows - to compile **UMFPACK** under MinGW and to use this library from within Microsoft Visual Studio.

Fetching and unpacking

UMFPACK depends on **AMD** and UFConfig. Hence it is necessary to download and unpack three archives from the **UMFPACK** site. I will use `wget` to get archives and `tar/gz` to unpack them.

```
$ wget http://www.cise.ufl.edu/research/sparse/umfpack/UMFPACK-5.2.0.tar.gz

$ wget http://www.cise.ufl.edu/research/sparse/UFconfig/UFconfig-3.1.0.tar.gz

$ wget http://www.cise.ufl.edu/research/sparse/amd/AMD-2.2.0.tar.gz
```

Let us check that we have the files

```
$ ls
AMD-2.2.0.tar.gz UFconfig-3.1.0.tar.gz UMFPACK-5.2.0.tar.gz
```

and then unpack them

```
$ tar zxvf UMFPACK-5.2.0.tar.gz
```

Use TAB for name completion. That is, type `tar zxvf UM` and press TAB.

```
$ tar zxvf UFconfig-3.1.0.tar.gz

$ tar zxvf AMD-2.2.0.tar.gz
```

Explore the directories and files. Short information is in `UMFPACK/README.txt` and the documentation is in `UMFPACK/Doc`. Pay attention to `UMFPACK/Makefile`. This is the main makefile that will be used to build the library.

There are different configuration settings. They are made by editing file `UFconfig/UFconfig.mk`. The variables defined in this file will be included in all makefiles.

Compiling without BLAS

Let us start by disabling the use of BLAS. The performance will suffer but this is the simplest way to start working. In this case, we need just to specify a C compiler in `UFconfig.mk`. There are some Fortran files in the **UMFPACK** distribution but by default they are not used. For `gcc` it suffers to change the next lines

```
CC = gcc
CFLAGS = -O3
```

The lines below are not to use BLAS. First it is necessary to comment out variables BLAS and LAPACK

```
#BLAS = -lblas -lgfortran -lgfortranbegin
#LAPACK = -llapack
```

Second is to modify

```
UMFPACK_CONFIG = -DNBLAS
```

Finally on Windows it is good to add `*.exe` to the variable

```
CLEAN = *.o *.obj *.ln *.bb *.bbg *.da *.tcov *.gcov gmon.out *.bak *.d *.exe
```

On Windows `gcc` adds `.exe` to binaries and this setting will clean them. Otherwise later on you may need to clean them manually.

The modified file is UFconfig.mk.noblas. You can just replace with it `UFconfig/UFconfig.mk`.

Now we go to the directory **UMFPACK** and run make

```
$ cd UMFPACK

$ make
```

Now make reads `Makefile` in this directory and executes it. It builds the **AMD** library in `AMD/Lib`, the **UMFPACK** library in `UMFPACK/Lib`, and then demos in `UMFPACK/Demo`. After that it runs demos and compares output with the output included with the library. One sees some differences but they are not essential.

That's it. Now you have `AMD/Lib/libamd.a`, `UMFPACK/Lib/libumfpack.a` and compiled demos in `UMFPACK/Demo`, that you can use as starting points on how to use **UMFPACK** in your code.

Compiling with ATLAS

UMFPACK uses calls to BLAS to reach the maximum efficiency. I will use ATLAS (you can find precompiled ATLAS with Cygwin at lib.tar.gz) and assume that it is located at `$HOME/lib/atlas`

```
$ ls $HOME/lib/atlas
libatlas.a libblas.a libcblas.a libf77blas.a liblapack.a
```

If you put it in another location, please modify the path to ATLAS below accordingly.

Let us first change the lines that told **UMFPACK** not to use BLAS and see what happens. Change in `UFconfig.mk` the line with `UMFPACK_CONFIG` to

```
UMFPACK_CONFIG =
```

or just comment it out. Now we have to delete previously compiled object files

```
$ make purge
```

and compile **UMFPACK** again without `-DNBLAS`

```
$ make
```

What happens is that make compiles the **AMD** and **UMFPACK** libraries but fails to link demos. There should be linking errors

```
gcc -O3 -I../Include -I../../AMD/Include -I../../UFconfig -o umfpack_di_demo umfpack_di_demo.c ../Lib/libumfpack.a ../../AMD/Lib/libamd.a -lm
../Lib/libumfpack.a(umf_di_blas3_update.o):umf_blas3_update.c:(.text+0xe8): undefined reference to `_dtrsm_'
../Lib/libumfpack.a(umf_di_blas3_update.o):umf_blas3_update.c:(.text+0x184): undefined reference to `_dgemm_'
../Lib/libumfpack.a(umf_di_blas3_update.o):umf_blas3_update.c:(.text+0x1ff): undefined reference to `_dger_'
../Lib/libumfpack.a(umf_di_local_search.o):umf_local_search.c:(.text+0x4e4): undefined reference to `_dgemv_'
../Lib/libumfpack.a(umf_di_local_search.o):umf_local_search.c:(.text+0x6b0): undefined reference to `_dtrsv_'
```

Here we see a list of functions that now have been inserted in the **UMFPACK** library and which the linker could not find. However, the libraries **AMD** and **UMFPACK** by themselves are done and one can already use them.

There is a useful tool `nm` that allows us to see the symbols defined in the libraries. The command

```
$ nm Lib/libumfpack.a | grep dgemm
U _dgemm_
U _dgemm_
```

confirms us that the symbol `_dgemm_` has been used in the **UMFPACK** library but is not defined there (symbol `U`). On the other hand, we can see that this function is defined in `libf77blas.a` (symbol `T`)

```
$ nm $HOME/lib/atlas/libf77blas.a | grep _dgemm_
U _atl_f77wrap_dgemm__
00000000 T _dgemm_
00000000 T _atl_f77wrap_dgemm__
```

This means that we have to tell `make` to use ATLAS libraries when it compiles demos. To this end it is necessary to define correctly the BLAS variable that we commented out previously

```
BLAS = -L$(HOME)/lib/atlas -lf77blas -latlas -lg2c
```

`-L` defines the path to the libraries and you may need to modify it appropriately. The path should be absolute or relative to `UMFPACK/Demo`, as `gcc` will be executed in this directory. **UMFPACK** uses Fortran interface to BLAS that is defined in `libf77blas.a` and the implementation by itself is in `libatlas.a`. `libf77blas.a` has been compiled with `g77` and as such it needs `libg2c.a`.

Now

```
$ make
```

should build demos correctly. Note that this time it does not build the libraries again.

The modified `UFconfig.mk` set to use ATLAS is here (UFconfig.mk.atlas) but it may be necessary to change `-L` in `BLAS` to the right location.

Compiling under MinGW

Cygwin is a tool to port Unix applications to Windows. As such, it emulates Unix API on Windows and the final application is linked to `cygwin1.dll` that implements the interface. One can see it with `cygcheck`, for example

```
$ cygcheck Demo/umfpack_simple.exe
Demo/umfpack_simple.exe
C:\cygwin\bin\cygwin1.dll
C:\WINDOWS\system32\ADVAPI32.DLL
C:\WINDOWS\system32\ntdll.dll
C:\WINDOWS\system32\KERNEL32.dll
C:\WINDOWS\system32\RPCRT4.dll
C:\WINDOWS\system32\Secur32.dll
```

If we would like to use **UMFPACK** with Microsoft Visual Studio, we need to make sure that the compiled libraries do not call Unix API. This is possible if one uses the flag `-mno-cygwin`, which forces gcc to use MinGW.

Add `-mno-cygwin` to `CFLAGS` in `UFconfig.mk`

```
CFLAGS = -O3 -mno-cygwin
```

Note that `CFLAGS` is defined two times and it is necessary to add this to the second definition or to comment the second definition out. The file with the change can be found here (UFconfig.mk.mingw).

Now

```
$ make purge
```

```
$ make
```

and the libraries compiled with `-mno-cygwin` are ready. Note that in this case the differences in output files will be much bigger. The first reason is that with `-mno-cygwin` the application writes the end of line as `CR LF` and in the output supplied with **UMFPACK** the end of line is just `LF`. This can be circumvented with `-b` for `diff`

```
$ diff -b my_umfpack_di_demo.out umfpack_di_demo.out
```

Still, there will be more differences as the formatting of numbers is different anyway.

Using UMFPACK with Microsoft Visual Studio

Microsoft Visual C compiler can be called from the command line and this is possible directly from within the Cygwin environment. To this end, it is necessary to modify the path and to define several environment variables required by `cl`. You can look at Using Microsoft Visual C under Gygwin to see how I have done it under `tsch`.

When everything is done correctly, the command `cl` should produce something like below

```
$ cl
Microsoft (R) 32-Bit C/C++-Optimierungscompiler Version 14.00.50727.762 für 80x86
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Syntax: cl [ Option... ] Dateiname... [ /link Linkeroption... ]
```

The Microsoft linker recognizes the libraries compiled by `gcc` under Cygwin. What we need to do is to just rename them from `*.a` to `*.lib`.

Let us take as an example `umfpack_simple.c` from `UMPFACK/Demo`. Make a directory `ms` in the directory where we have started

```
$ cd ..

$ ls
AMD UFconfig UMFPACK AMD-2.2.0.tar.gz
UFconfig-3.1.0.tar.gz UMFPACK-5.2.0.tar.gz

$ mkdir ms

$ cd ms
```

and copy this file here. Then copy and rename `libamd.a`, `libumfpack.a`, `libf77blas.a`, `libatlas.a` by changing the extension from `*.a` to `*.lib`. Additionally we need `libg2c.a` for `libf77blas.a` and `libgcc.a`, as `libg2c.a` uses some symbols from it. You should take these two libraries from `/lib/gcc/i686-pc-mingw32/3.4.4` (not from `/lib/gcc/i686-pc-cygwin/3.4.4/`).

At the end you should have the next files

```
$ ls
libamd.lib libf77blas.lib libgcc.lib umfpack_simple.c
libatlas.lib libg2c.lib libumfpack.lib
```

Now the command

```
$ cl -MD -I../UMFPACK/Include -I../AMD/Include -I../UFconfig umfpack_simple.c libumfpack.lib libamd.lib libf77blas.lib libatlas.lib libg2c.lib
libgcc.lib
```

does the job. It compiles `umfpack_simple.c` and links the object file with the libraries. The flags `-I` tells `cl` where to find the headers from **UMFPACK**. You may need to modify the path if you have made this directory in another place. Interestingly enough that `cl` understands correctly the direct slash as the sigh for directory. `-MD` is the regime with which the MinGW libraries are working best. In this case actually it is possible to compile without it but there will some warnings from the linker. If you understand the command, you can make also this settings directly in GUI.

Finally the question how I have found that it is necessary to add `libgcc.lib`. The answer is simple. Try to compile without it. Then you immediately see some symbols missing and simple manipulations with nm shows that they are in `libgcc.a`.

By editing makefiles in UMPFACK it is possible to compile **AMD** and **UMFPACK** directly with `cl`. This should be relatively simple. What will be more challenging is to compile ATLAS with `cl`. Alternatively you can use the optimised Intel BLAS.

Discussion

Please post your comments, questions, suggestions to the discussion group at http://groups.google.com/group/matrixprogramming.