

OpenFOAM on Windows

Porting Requirements and Experiences

Petr Vita

`petr.vita@mu-leoben.at`

Montanuniversität Leoben

June 9, 2007

Outline

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of `wmake` Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Outline

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of `wmake` Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Outline

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of `wmake` Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Outline

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of `wmake` Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Overview

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of `wmake` Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Motivation

- How it all started by us?
- You are facing Windows station and you want to get OpenFOAM working on it, because
 - you are developing applications that have to be delivered under Windows.
 - its OS is not a valid excuse to let its processor idle.
 - there exists a group of users who can use only Windows.
 - it is a challenge to get it working.
 - your Boss ordered you to do it!
 - ...

Overview

1 Basics

Motivation

Windows vs Linux

Cygwin

2 Porting Step-by-Step

Preparing your Workbench

Modification of wmake Build System

Building of OpenFOAM

How to Compile Your Own Application?

3 Unported and Future Stuff

4 Acknowledgements and Links

File System Differences

- Speaking about Linux we can expect as a standard filesystem ext2fs or ext3fs, and we most probably face NTFS on Windows OS.
- Linux file systems ext2fs and ext3fs are
 - case-sensitive, i.e. hello represent HelLo two files,
 - and limited to 255 characters for a filename and unlimited for path length.
- Windows NTFS filesystem is
 - case-insensitive,
 - limited to filenames with 254 characters plus dot symbol,
 - and limited to 32.767 characters with each path component up to 255 characters.

Linux API vs Win32API

- Linux adheres to the POSIX standard and provides Linux API.
- Window OS adheres to some proprietary guidelines and provides Win32API.
- Linux API and Win32API are not compatible—different process model, different OS layering, etc.
- OpenFOAM is target to Linux primarily, heavily using the Linux API. Thus we have to
 - either rewrite Linux API dependent code to conform Win32API,
 - or use some emulation into Win32API.

Windows vs. Linux Dynamic Modules System

- **Linux dynamic modules are named shared object files (*.so).**
 - A shared object file (*.so) contains code and also *names of functions and data*, that it expects to find in the program, so called *dangling references*.
 - All these references are changed to point to the actual locations in the program, when the file is joined to the program in the runtime.
- **Windows OS dynamic modules are dynamic-link library files (*.dll).**
 - Dynamic-link library file (*.dll) has no dangling references, access to functions or data goes through a lookup table.
 - An import library is generated during linking of a dynamic library to reassure following compilation targets about DLL code existence.
 - DLL code already uses the DLL lookup table, and the lookup table is modified at runtime to point to the functions and data.

Development Tools

- Windows OS opposite to Linux distributions does not contain development tools.
- What worse, there do not exist native GNU development tools OpenFOAM is using—GCC, Make, Flex, Bison, etc.
- There exist development tools native to Windows OS like MSVC++ etc., but these are not compatible with an OpenFOAM building system and come with extra costs.
- Some of libraries OpenFOAM depends on exist in a native form for the Windows platform, e.g. JDK, some do not.
- In order to overcome these shortages, we have to
 - either cross-compile to the Windows platform,
 - or look for some emulation that allows us to use native Linux development tools on Windows OS,
 - and find the way how to port missing libraries onto the Windows platform.

Overview

1 Basics

Motivation

Windows vs Linux

Cygwin

2 Porting Step-by-Step

Preparing your Workbench

Modification of wmake Build System

Building of OpenFOAM

How to Compile Your Own Application?

3 Unported and Future Stuff

4 Acknowledgements and Links

Cygwin

- GNU + Cygnus + Windows = Cygwin
- What is Cygwin?
 - Cygwin is a Linux-like environment for Windows OS. It consists of two parts–
 - a `cygwin1.dll` which acts as a Linux API emulation layer providing substantial Linux API functionality,
 - a collection of tools which provide Linux look-and-feel, inclusive development tools.
 - Cygwin currently works with all recent, commercially released x86 32- and 64-bit versions of Windows, with the exception of Windows CE
- What is not Cygwin?
 - Cygwin is not a way to run native Linux apps on Windows OS, you have to rebuild your application from source.
 - Cygwin is not a way to magically make native Windows apps aware of UNIX functionality, like signals, ptys, etc.

Why We Speak about Cygwin?

- **Cygwin is a cure for the most of our problems,**
 - Cygwin will allows us to port Linux API dependent code into Win32API without an additional work.
 - Cygwin brings with all development tools we need.
 - There exist Cygwin versions of the most libraries OpenFOAM depends on.
 - Cygwin allows us to compile missing tools and libraries.
- **but for a price!**
 - Inserting an emulation layer between OS and user code will introduce a performance loss.
 - While Cygwin tries to emulate Linux API as far as possible, there are still some incompatibilities hidden.
- **When we speak about porting of OpenFOAM on Windows, we actually mean porting of OpenFOAM into Cygwin.**

Overview

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of `wmake` Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Installing Cygwin

- First of all you will have to install and prepare Cygwin environment.
- Installation of Cygwin is very trivial and it is done with a setup program you can download directly from <http://cygwin.com/>.
- OpenFOAM porting will need some extra tools and packages to those default one, do not forget them.
 - Make
 - GNU GCC
 - binutils
 - Bison
 - Flex
 - Xorg-X11-devel
 - w32api
 - sed

Compiling Compiler

- You will need for a successful building of OpenFOAM a self-compiled GNU GCC compiler of a major version 4.
 - Unfortunately the latest Cygwin distribution, Cygwin 1.5.24, contains only GNU GCC 3.4.4
 - The Cygwin compiler has a hidden bug, GCC bug 24196, in its string implementation.
- How you compile the compiler?
 - 1 Download sources of GNU GCC from <http://gcc.gnu.org/>
 - GNU GCC 4.1.2 compiles without problems, GNU GCC 4.2.0 does not so far.
 - 2 Read the documentation twice before you start.
 - 3 Compile the compiler following the instructions.
 - Use same configuration settings as the provided Cygwin GNU GCC compiler, you can help yourself with `g++ -v`.
 - Add a configuration flag `--enable-fully-dynamic-string`.
 - 4 Place the compiled compiler into the OpenFOAM source tree into directory `cygwin`.

Unpacking Source Distribution

- Direct unpacking of OpenFOAM sources is not possible due to case-sensitive filenames and directories.
- There are two basic ways how you can deal with this issue.
 - You either make your Windows filesystem case-sensitive, it did not work for us,
 - with the help of a Microsoft Services for Unix package,
 - or use Cygwin managed mounts,
 - or you have to rename clashing files and directories
 - with the help of scripts,
 - or manually.
- We have used the manual way.
 - 1 Unpack sources onto Linux filesystem exported with Samba.
 - 2 Unpack sources onto Windows filesystem.
 - 3 Use `diff -brief dir1 dir2` to get the list of missing files.
 - 4 Systematically rename and copy missing things, e.g. `*_.*`, `*_.`
 - 5 Search and replace all references to renamed files in the code.

Overview

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of wmake Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Modification of `wmake` Build System

- `wmake` system is central to the OpenFOAM building as it allows a platform dependent compilation.
- You have to modify it, in order
 - to compile Windows dynamic-link libraries,
 - to work with import libraries, that are specific to Windows platform,
 - to be able to compile cyclic dependencies between dynamic-link libraries.
- Following components have to be modified
 - `wmkdep` has to process import dependencies.
 - Makefile and MakefileFiles need platform specific compilation rules and switches.
 - `rules/cygwin` and `rules/cygwinGcc4` have to be defined.
 - Some scripts—`bashrc` etc.—have to be modified to integrate Cygwin modifications into `wmake` system, so it can co-exist in the main distribution of OpenFOAM

Famous Cyclic Dependency Problem

- There exist DLLs that depend on each other in OpenFOAM, i.e. `Pstream` and `OpenFOAM`.
 - This was probably the reason why porting of OpenFOAM on Windows took such a long time.
- Windows are using import libraries to reassure linked targets about existence of the code they depend on.
- If one DLL file needs another one to compile and vice versa, we do not have import libraries at the hand, when we compile the first DLL of such a cycle. Instead, we need to provide the linker with an object code of files, that will be linked into a yet non-existent DLL-file, to reassure him about future code existence.

Compilation Example Live

dll.cpp: Dynamic-Link Library code

```
1 double myMain();  
  
3 double myDll() {  
    return myMain();  
5 }
```

main.cpp: Main program

```
1 #include <iostream>  
  
3 int myDll();  
  
5 int myMain() {  
    return 1;  
7 }  
  
9 int main() {  
    std::cout << "myDll() = " << myDll() << std::endl;  
11    return 0;  
}
```

How to Compile It?

Compilation Commands

```
g++ -c dll.cpp
2 g++ -c main.cpp -o import_main.o
g++ -shared -o dll.dll -Wl,--out-implib,dll.lib -Wl,-no-undefined
4 -Wl,--enable-runtime-pseudo-reloc dll.o import\_main.o
6 g++ -c main.cpp
g++ -o main.exe main.o -L./ -ldll
```

- The whole magic of Windows compilation is shown here.
 - A compilation of object files and *an import object file*.
 - Linking of the first dynamic-link library in a dependency cycle.
 - An import library creation to reassure about DLL's code.
 - An inclusion of `dll.o` code into DLL.
 - Use of import object file, to reassure our DLL about the code that do not exist yet.
 - Linking of an executable that depends on DLL.

Overview

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of `wmake` Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

Building of OpenFOAM

- Having the workbench and modified wmake system ready, you can compile whole OpenFOAM. You have to enter one directory after another and use wmake commands.
 - There is no difference against Linux distribution, compilation commands stay exactly the same, i.e. `wmake libso dir`, `wclean dir`, `wmake dir`.
- During OpenFOAM compilation following problems regularly emerge.
 - Name clashes in `lnInclude` directory.
 - Clashes between the OpenFOAM code and STD C++ Library.
 - `Make/files` file has to be extended with import object files if cyclic dependency is present.
 - `Make/options` file has to be extended by libraries our code depend on.

Hidden Evil in lnInclude Directory

- OpenFOAM links during its compilation all headers hidden deep in the source tree for a given compilation target into lnInclude directory.
 - This is comfortable for lazy programmers, who do not need to write path into each #include directive,
 - but brings our headaches, as case-sensitive filenames separated on different logical levels suddenly meet at one location.
- How to deal with this?
 - Either let a script to find these files and to do work for you,
 - or do it manually as we did.
 - ① `wmLnInclude` utility will report all clashing filenames during the compilatin.
 - ② Systematically rename clashing filenames, e.g. `*_.*`.
 - ③ Search and replace all references to renamed files in the code.

Clashes between OpenFOAM Code and STD C++ Library

- OpenFOAM contains some code that mimics STD C++ Library functionality and what worse, this code shares case-sensitive filenames with it, e.g. `Map.H` vs. `map.h`.
 - We guess, that this is an artefact from those wild times, when STD C++ Library was not standard yet and its portability was lacking.
- What to do with these particular files?
 - ① Localize these special files first.
 - Either list all the headers of STD C++ Library and look up case-sensitive equivalents in the OpenFOAM source tree,
 - or pin-point them during compilation through an obscure compilation errors.
 - ② Systematically rename found filenames, e.g. `*.hh`.
 - ③ Search and replace all references to renamed files in the code.

Overview

① Basics

Motivation

Windows vs Linux

Cygwin

② Porting Step-by-Step

Preparing your Workbench

Modification of wmake Build System

Building of OpenFOAM

How to Compile Your Own Application?

③ Unported and Future Stuff

④ Acknowledgements and Links

How to Compile Your Own Application?

- Once you have visited all directories in OpenFOAM source tree and fixed all local problems, you are nearly done.
- What stays for now, are just your own applications.
 - Be assured, that there is no difference in their compilation to what you have done so far.
 - In the most cases they will directly compile without your help.
 - The most common problem will be just a missing library information in Make/options file.
 - Only in the rare cases you can meet some case-sensitivity problem in `lnInclude` directory or with STD C++ Library.
- Finished? Congratulations! You have OpenFOAM on Windows!
 - You got an option to deliver your applications on your customers now.
 - `cygcheck` is your friend.

Unported and Future Stuff

- Some parts of OpenFOAM are not ported yet.
 - Parallel computing–OpenMPI, LAM MPI library.
 - FoamX click-me–application.
 - FoamX was ported to Cygwin in OpenFOAM 1.3.
- Surprisingly the future porting work should concentrate on the missing features.
- Porting is now a tedious, frustrating and rather demotivating work, that has to be done for each OpenFOAM release from start.
 - Group of OpenFOAM users on Windows is quite small.
 - The main distribution of OpenFOAM did not include any changes made so far.
 - No reflection was taken on the case-sensitive filenames.
 - Number of case-sensitive files even grew in OpenFOAM 1.4.

Acknowledgements

Thanks for listening!

Questions?

- Acknowledgements

- Brooks Moses (*bmoses@stanford.edu*)
 - Done a tremendous amount of work with renaming of clashing files, scripting and updating of Cygwin port from version 1.2 to 1.3.
- Takuya Oshima (*ohshima@wa2.so-net.ne.jp*)
 - Ported FoamX in OpenFOAM 1.3.
- Ron W. Cresswell (*roncresswell@comcen.com.au*)
 - Seriously contributed into discussion regarding porting.
- Niklas Nordin (*niklas.nordin@scania.com*)
 - Seriously contributed into discussion regarding porting.
- Bernhard Gschaider (*bgschaid@ice-sf.at*)
 - Had to listen to my laments over porting and provided me with a steady stream of anti-Czech jokes.

Links

- **OpenFOAM Forum**
 - (<http://openfoam.cfd-online.com/cgi-bin/forum/discus.cgi>)
- **Windows vs. Unix: Linking dynamic load modules**
 - (<http://xenophilia.org/winvunix.html>)
- **Microsoft Developer Network**
 - (<http://msdn.microsoft.com/library/>)
- **GNU GCC**
 - (<http://gcc.gnu.org/>)
- **Cygwin**
 - (<http://cygwin.com/>)