Annex A: Using the Software

This annex provide guidance for compiling and running the support software, which can be downloaded from the book site, http://booksite.elsevier.com/9780128037614/. Before going into the details about libraries and examples, a few words about software installation may be useful.

A.1 LIBRARIES REQUIRED

Pthreads and Windows are native multithreading libraries in Unix-Linux or Windows systems. They are automatically integrated in the operating system. OpenMP is also automatically integrated in the C-C++ compilers (Intel, GCC, VisualC++, etc.). TBB, extensively used in this book, requires separate installation.

Installing and configuring TBB are very simple operations. First, the archive file corresponding to the latest release for Unix or for Windows systems is downloaded from the TBB web site [18]. TBB is installed by expanding the release archive file in some installation directory. Then, the file `/bin/tbbvars.sh` is modified: the `export TBBROOT` line must be completed with the full path to the TBB installation directory.

This is all about installation. Every time a new shell is started in which TBB is used, the `tbbvars.sh` script must be executed (sourced), passing a command line argument corresponding to the platform architecture (ia32, ia64, etc.) and, in Windows systems, the Visual Studio version. This script sets the paths to libraries and include files. Running the script with no command line options lists the set of options available.

At this point, TBB is ready to run. The switch `link tbb.lib` in Windows or `-ltbb` in Linux has to be included in the linker, but this is automatically done in the makefiles provided with the codes.

A.2 SOFTWARE ORGANIZATION

The overall software organization is described in Figure A.1. There are two root directories, `/msvs` for Windows platforms and `/linux` for Unix-Linux platforms. Here is the typical content and organization of the different library or chapter examples subdirectories:

- */vath_xxx*: library directory:
  - */include*: library include files.
  - */src*: library source files.
  - */test*: sources for library tests.
  - Makefile to build library and tests.
  - Data files (*.dat) required by some of the tests.
- */chxx*: chapter examples directory.
  - */include*: additional include files required by the examples.
  - */src*: examples sources.
  - Makefile to build he examples.

- */linux*
  - */vath_pth :* Pthreads implementation of "vath"
  - */vath_std :* C++11 implementation of "vath"
  - */ch3:* chapter 3 examples
  - */ch4:* chapter 4 examples
  - ...
- */msvs*
  - */vath_win* : Windows implementation of "vath"
  - */vath_std :* C++11 implementation of "vath"
  - */ch3*: chapter 3 examples
  - */ch4*: chapter 4 examples
  - ...

FIGURE A.1

Directory organization of the support software. Each directory under /msvs or /linux includes the corresponding makefile.

- Data files (*.dat) required by the examples.
- Notes: a text file with instructions for compilation and execution of the chapter examples, and eventual comments on the chapter content.

There are obvious redundancies in this code organization. For example, the source, include, and test files for the vath_std implementation of the vath library are identical in the /msvs and /linux directories. Only the makefiles are different. But we have deliberately tried to avoid universal but incomprehensible makefiles, and settled for simple makefiles that can easily be modified by users if needed. Therefore, all the relevant files are reproduced whenever needed.

The same comment applies to the chapter examples. There are, mainly in the early chapters, code examples that target specifically Pthreads, Windows threads, or the C++11 library, and the corresponding files are at the right places. However, most of the examples are portable, and the source files are identical for Windows or Unix-Linux platforms.

A.2.1 BUILDING AND TESTING THE VATH LIBRARY

Each library version directory includes the appropriate makefile for compilation of the library and the compilation and execution of the test codes, as well as data files needed for some tests. Source, include files, and test codes are in the /src, /include, and /test subdirectories, respectively.

To build and test the libraries (in Windows, replace make by nmake):

- Run make libva to build the library.
- Run make all to compile the test codes.
- Run make execall to run all the test codes (which can, of course, also be run individually).

The default compilers are Visual Studio 2013 in Windows and GCC (4.8 or higher) in Linux-Unix. These compilers are fully C++11 compliant, and support all the implementations of the library. The native implementations should work with older compilers. Setting the INTEL_ENV environment variable, when compiling in Linux platforms, selects the Intel compilers. However, Intel compilers are not yet fully C++11 compatible, so they cannot be used to build the C++11 implementation of the library.

A.2.2 COMPILING AND RUNNING THE EXAMPLES

Examples are compiled and run from the corresponding chn directory. Sources are in the /src subdirectory, and the /include sub_directory contains additional include files needed specifically for the chapter examples. Each /chn directory contains eventual data files needed to run the examples, as well as a makefile. The "notes" file in each chn directory provides detailed instructions for the compilation and execution of the chapter examples.

Filename conventions. While most of the examples depend only on portable libraries (OpenMP, TBB, and vath), those in the early chapters target specific libraries (Pthreads, Windows, or C++11). For this reason, a naming convention has been adopted for the sources, ending the source name with a trailing _P, _W, or _S (for "standard") when they run only on Pthreads, Windows, or C++11 environments, respectively. Filenames without these trailing indicators are portable.

Selecting the library. When compiling and running a portable chapter example, different library implementations can be chosen by setting environment variables. This is needed both to set the path to the include files and to the correct version of the library. By default, the native implementations are chosen: Pthreads in Unix-Linux and Windows threads in Windows. Setting instead the CPP11_ENV environment selects the portable C++11 implementation.

- To set in Linux, export CPP11_ENV = 1 or any nonempty string.
- To set in Windows, set CPP11_ENV = 1 or any nonempty string.
- To unset in Linux, export CPP11_ENV = (empty string).
- To unset in Windows, set CPP11_ENV = (empty string).

Running OpenMP codes. Compiling and running OpenMP codes requires an extra switch in the compiler and linker, otherwise the OpenMP directives are ignored. This is accomplished by setting the OPENMP_ENV environment variable.

Running codes using TBB. The tbbvars.sh script must be executed every time a new shell is open. This is sufficient, the makefiles incorporate the -ltbb switch in codes using TBB.

Several examples deliberately mix libraries or programming environments to exhibit interoperability. In those cases, it may happen that more than one environment variable needs to be set. Such is the case, for example, when OpenMP codes use synchronization utilities provided by the C++11 implementation of the vath library. Many OpenMP codes are linked to the vath library just to use the portable CpuTimer class that measures execution times.

All the Windows examples are console applications with command line compilation. Providing a Visual Studio project for each one of them is overkill. A Visual Studio project will eventually be provided on the book web site for the more complex examples of the later chapters. Remember that accessing the CL compiler and the libraries from the command line requires the execution of a varsargs script every time a new console is opened.

A.3 VATH CLASSES

A complete list of all the utility classes provided by the vath library can be found in the vath_status.pdf document, available in the root directory of the software release in the book Web site. This document also summarizes the (few) portability limitations of some of the vath classes, resulting from missing services in the basic libraries or missing features in the different compilers.

The vath_status.pdf document will be updated in the future, if and when improved versions of the library become available.