SANDIA REPORT

SAND2013-7294
Unlimited Release
Printed August 2013

Building Guide: How to build *Xyce*™ from source code

Eric R. Keiter, Thomas V. Russo, Richard L. Schiek, Peter E. Sholander, Heidi K. Thornquist, Ting Mei, Jason C. Verley, David G. Baur

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2013-7294
Unlimited Release
Printed August 2013

Building Guide:
How to build *Xyce*™ from source code

Eric R. Keiter, Thomas V. Russo, Richard L. Schiek,
Peter E. Sholander, Heidi K. Thornquist, Ting Mei, Jason C. Verley
Electrical Models and Simulation
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1177

Abstract

While **Xyce** uses the Autoconf and Automake system to configure builds, it is often necessary to perform more than the customary "./configure" builds many open source users have come to expect. This document describes the steps needed to get **Xyce** built on a number of common platforms.

Contents

1. Introduction **9**

Target Audience 9

Overview 9

Prerequisites 10

2. Installing Required Libraries **12**

Installing Prerequisite Libraries From System Packages 12

Linux 12

Red Hat Enterprise Linux (RHEL) and CentOS 12

Ubuntu and Debian Linux 13

Mac OS X 13

FreeBSD 14

Cygwin 15

3. Installing Trilinos **17**

Introduction 17

Building Trilinos for Serial **Xyce** 18

Building Trilinos for Parallel **Xyce** 20

4. Building Xyce **22**

Configure options 22
Specifying compilers and helper programs 22
Specifying compiler flags 23
Specifying additional library search directories 23
Specifying additional include search directories 23
Example configure invocations for selected systems 24
Serial Builds 24
Red Hat Linux 24
Mac OS X 24
FreeBSD 25
Cygwin 26
Parallel Builds 26
Red Hat Linux 26
Mac OS X 27
FreeBSD 27
Cygwin 28

Appendix

Building UMFPACK and AMD 29
Building ParMETIS 29

List of Figures

3.1 cmake invocation for configuring serial Trilinos 19

3.2 cmake invocation for configuring parallel Trilinos 21

1. Introduction

Target Audience

This guide is intended for those interested in building **Xyce** from the source code. It is recommended that anyone intending to build **Xyce** be familiar with editing text files and compiling programs on their target computer. Prior experience using command line tools like `configure`, `cmake` and `make` is recommended. For information on using Xyce please see the Xyce Users' Guide [1].

Overview

Installing **Xyce** from source requires the following general steps:

- Install a complete compiler suite
- Install all prerequisite libraries from packages or from source
- Build Trilinos from source and install it
- Build **Xyce**

Most libraries that **Xyce** depends on are available through system package managers; they are also available in source form, which you can compile yourself. Since the majority of the work in installing **Xyce** involves installing the prerequisite libraries, most of this document is focused on those prerequisites.

It is important to note that both Trilinos and **Xyce** make use of out-of-source builds. This means the executables and/or libraries are produced in a separate directory structure from the source code. Therefore, care must be taken to not conflate the two directory trees by properly specifying the proper source directory and, possibly, the target directory, in the configure scripts.

Prerequisites

Building **Xyce** requires a computer with a modern C++ compiler. Some of the libraries that **Xyce** uses also require a Fortran compiler. The **Xyce** development team has successfully used the Gnu Compiler Collection (`gcc`), the Intel Compiler Collection and Clang. Other compilers may work, but we have not tested any others. For building and running Xyce in parallel, an MPI compiler front-end is needed, such as Open-MPI or MPICH. These tools are sometimes tuned for specific computing clusters, but they may also be downloaded from `http://www.open-mpi.org` and `http://www.mpich.org`.

Xyce depends on many external libraries to supply enhanced parsing, mathematical algorithms and parallel communication. Some of the required libraries may already be present on your system, and some are optional; but many will have to be built or installed prior to building **Xyce**.

The required libraries for **Xyce** are listed in Table 1.1. All of the libraries are either freely available, or have equivalents that are open source. Note that if one is building a serial version of **Xyce**, i.e. a version that does not use MPI, then the ParMETIS library is not needed and Trilinos must be built with MPI disabled. Likewise, if one is building a parallel version of **Xyce**, then one must build ParMETIS and Trilinos with MPI enabled.

Table 1.1. Libraries Required for Building Xyce.

Name	Version	Notes and Download URL
blas	-	May be included with compiler, or may require separate package installation. One may also use ATLAS: `http://math-atlas.sourceforge.net`
lapack	-	May be included with compiler, or may require separate package installation. One may also use ATLAS, or download from: `http://www.netlib.org`
bison	2.3-2.7	Required for the chemical reaction parser. Download from: `http://www.gnu.org`
flex	2.5.34 or later	Required for the chemical reaction parser. Download from: `http://www.gnu.org`
FFT package		Required for Fourier and HB analysis. **Xyce** can use either the Intel Math Library or FFTW3. See: `http://software.intel.com` or `http://www.fftw.org`
UMFPACK	4.1-5.2	May be installed as part of SuiteSparse package. If otherwise unavailable, download from: `http://www.cise.ufl.edu/research/sparse/umfpack`
AMD	1.0 or greater	May be installed as part of SuiteSparse package. If otherwise unavailable, download from: `http://www.cise.ufl.edu/research/sparse/amd`
ParMETIS	3.1 or later	This is required for MPI parallel builds of Xyce and it must be compiled with an MPI compiler or use MPI libraries during linking. Download from: `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`
Trilinos	11.2.4	**Xyce** requires Trilinos for many linear algebra and solver services. When building the serial version of **Xyce** one must build Trilinos without MPI support enabled. When building Trilinos for the parallel version of **Xyce**, then Trilinos must be built with MPI support enabled. Download from: `http://trilinos.sandia.gov`

2. Installing Required Libraries

Installing Prerequisite Libraries From System Packages

Except for Trilinos, and sometimes ParMETIS, the libraries required for building Xyce are available in popular system package repositories, and as such, are easily installed. The **Xyce** team recommends using these prepackaged libraries as much as possible.

Here we enumerate the packages known to exist on systems the **Xyce** team has built on. Other systems probably have similarly named packages.

Where a system does not provide a package for a given library, you will have to build that library from source. Guidelines for building required libraries from source are given in the Appendix.

Note that at this time no system provides packages for Trilinos, so it will always need to be built from source.

Linux

Red Hat Enterprise Linux (RHEL) and CentOS

These systems have all the prerequisite libraries available in the RPM repository except for ParMETIS. The following packages are required:

- a version of gcc, g++ and gfortran
- blas-devel
- blas
- cmake (needed for Trilinos)
- lapack-devel

- lapack
- bison
- flex
- fftw-devel
- fftw
- suitesparse-devel
- suitesparse

Note that on versions of these operating systems older than RHEL6/CentOS6, the fftw and suitesparse libraries might require use of an additional repository. On RHEL6 and CentOS 6 they require no special additions.

Install these packages using `sudo yum install` on each package name. In most cases, the -devel package will also cause the other package without "-devel" to be installed, so it is not strictly necessary to list all of them above.

If you are building the parallel version of **Xyce** you will also need the openmpi and openmpi-devel packages.

ParMETIS is not available in packages on these systems, so if you are building the parallel version of **Xyce**, you will have to build it from source.

Ubuntu and Debian Linux

All required packages are available in these systems' default repositories. See the list given for RHEL in Section 2. Note that in Debian-based systems, development packages are usually called "-dev" instead of "-devel", but otherwise the names will be the same as for RHEL and CentOS.

Install these packages using `sudo apt-get install`, or a graphical package manager.

Mac OS X

Building **Xyce** on Mac OS X will require that you obtain the XCode package from Apple, and you must also install the "XCode Command Line Tools".

In addition to XCode, you will also require a package manager such as MacPorts (`http://www.macports.org`) or Fink (`http://fink.thetis.ig42.org/`). Install one of these package managers according to the instructions provided on its web site.

For Fink, install the following packages:

- a version of gcc (for example gcc48)
- fftw
- suitesparse
- cmake (for Trilinos)
- openmpi (only for parallel installations)

For MacPorts, install

- a version of gcc (for example gcc48)
- fftw-3
- SuiteSparse
- cmake (for Trilinos)
- openmpi (only for parallel installations)

It is not necessary to install blas and lapack, as XCode has those available in a system library. bison and flex are provided by the XCode command line tools.

Install these packages in fink using `fink install <packagename>`.

If you are using MacPorts then use `sudo port install <portname>`.

If you are building the parallel version of **Xyce** you will also need to build ParMETIS from source.

FreeBSD

All libraries required by **Xyce**, including ParMETIS, but not including Trilinos, are available in the FreeBSD ports system.

For the serial version of **Xyce**, install the following ports:

- lang/gcc46 or later
- math/blas

- math/lapack
- math/suitesparse
- math/fftw3
- devel/bison
- devel/cmake (needed for Trilinos)
- textproc/flex

Install each of these using the standard FreeBSD ports system. For example, to install GCC 4.6, run the following commands as root:

```
cd /usr/ports/lang/gcc46
make install clean
```

If you are building the parallel version of **Xyce**, you also require:

- net/openmpi
- math/parmetis

One important point to note is that, as of this writing, the FreeBSD ParMETIS port does not install the "metis.h" file that goes along with the "libmetis.a" file it installs. After installing math/parmetis with "make install" but before doing any "make clean", install the "metis.h" file by hand:

```
cp work/parmetis-4.0/metis/include/metis.h /usr/local/include/parmetis
```

Cygwin

For Windows systems, the **Xyce** project team uses the Cygwin (`http://www.cygwin.com/`) system to provide a Unix-like environment with Unix-style tools. While it is possible to build **Xyce** using native tools, the team does not support that approach.

Install Cygwin according to its instructions. You will then need to install a series of packages in addition to the base Cygwin packages. Add:

- gcc-core

- gcc-g++
- gcc-fortran
- make
- cmake (needed for configuring Trilinos)
- lapack, liblapack0 and liblapack-devel
- libsuitesparseconfig-devel
- libumfpack-devel
- bison
- flex
- fftw3 and libfftw3-devel
- libtool

Many of these packages will automatically install additional packages that they require.

The **Xyce** project team has never attempted to build the parallel version of **Xyce** on Windows.

3. Installing Trilinos

Introduction

Trilinos is a very complex set of interdependent packages for a wide range of numeric computational problems. This section will provide an overview of building Trilinos for **Xyce**. However, for detailed questions on Trilinos or its build system please see the Trilinos getting started page at `http://trilinos.sandia.gov/getting_started.html` for places to start your inquiry.

This section assumes you have installed all prerequisite libraries from packages according to the instructions in the previous section, or have built missing libraries from source as described in the Appendix.

First, download and unpack the Trilinos source code. Because of the size and complexity of Trilinos, it is recommended to do an out-of-source build. For example, in the Trilinos source code directory (say `trilinos-11.x-Source`), you could create two "build" directories, called `trilinosSerial` and `trilinosParallel` for the serial and parallel builds, respectively.

Trilinos has many packages that can be built, only some of which are required by **Xyce**. The packages are specified using a configuration script. Trilinos requires CMake for its configuration. In the next sections we will provide examples of CMake invocations for typical serial and parallel builds of **Xyce**.

Building Trilinos involves only a few steps:

- invoke CMake using a build script specifying all the required parameters
- `make`
- `make install`

Because multiple versions of Trilinos might need to be built to support both serial and parallel builds of Xyce, we do not recommend installing Trilinos directly into system library and header directories. Our instructions for building Trilinos will therefore be set up for installation in a user-specified, non-standard location.

Building Trilinos for Serial **Xyce**

Serial **Xyce** requires a build of Trilinos that contains the NOX, LOCA, EpetraExt, BTF, Ifpack, Isorropia, AztecOO, Belos, Teuchos, Sacado and Amesos packages. Amesos must be configured to use KLU and UMFPACK. EpetraExt must be configured with BTF (block triangular factorization), experimental features, and graph reordering enabled. Trilinos must also be told where to find the UMFPACK and AMD libraries and headers.

The CMake invocation of Figure 3.1 is an example that configures a version of Trilinos with the packages and options required by a serial version of **Xyce**. There are several things to note about this figure:

- This text should be put into a file that will be used as a shell script. This file should be placed in an empty directory (the "build directory").
- Set the variable "SRCDIR" to have the *full path* of the location of the Trilinos source code.
- The "\" character must be the last character on any line where it appears. There must be no spaces following it.
- The script specifies which C, C++, and Fortran compilers to use
- Flags for passing to the compilers are specified, and in this case are all the same.
- Use the variable, "ARCHDIR," to define the location where Trilinos should be installed.
- We have assumed that AMD and UMFPACK libraries were installed in `/usr/lib` and their associated header files in `/usr/include/suitesparse`, as they would be on 32-bit Linux. If they were installed elsewhere, modify the script in the obvious way. Some common modifications are:
  - On 64-bit Linux, the libraries are in `/usr/lib64`.
  - On OS X, using fink, the libraries are in `/sw/lib` and the headers are in `/sw/include/suitesparse`
  - On OS X, using MacPorts, the libraries are in `/opt/local/lib` and the headers are in `/opt/local/include`
  - on FreeBSD the libraries are in `/usr/local`.

Once the script is created and customized, run it. If it completes without error, Trilinos is configured for building. Build with "`make`" and install with "`make install`."

```
#!/bin/bash
SRCDIR= < location of the trilinos source code >
ARCHDIR=/home/me/XyceLibs/Serial
FLAGS="-O3 -fPIC"
cmake \
-G "Unix Makefiles" \
-DCMAKE_C_COMPILER=gcc \
-DCMAKE_CXX_COMPILER=g++ \
-DCMAKE_Fortran_COMPILER=gfortran \
-DCMAKE_CXX_FLAGS="$FLAGS" \
-DCMAKE_C_FLAGS="$FLAGS" \
-DCMAKE_Fortran_FLAGS="$FLAGS" \
-DCMAKE_INSTALL_PREFIX=$ARCHDIR \
-DCMAKE_MAKE_PROGRAM="make" \
-DTrilinos_ENABLE_NOX=ON \
-DTrilinos_ENABLE_LOCA=ON \
-DTrilinos_ENABLE_EpetraExt=ON \
  -DEpetraExt_BUILD_BTF=ON \
  -DEpetraExt_BUILD_EXPERIMENTAL=ON \
  -DEpetraExt_BUILD_GRAPH_REORDERINGS=ON \
-DTrilinos_ENABLE_TrilinosCouplings=ON \
-DTrilinos_ENABLE_Ifpack=ON \
-DTrilinos_ENABLE_Isorropia=ON \
-DTrilinos_ENABLE_AztecOO=ON \
-DTrilinos_ENABLE_Belos=ON \
-DTrilinos_ENABLE_Teuchos=ON \
-DTrilinos_ENABLE_Amesos=ON \
  -DAmesos_ENABLE_KLU=ON \
  -DAmesos_ENABLE_UMFPACK=ON \
-DTrilinos_ENABLE_Sacado=ON \
-DTrilinos_ENABLE_ALL_OPTIONAL_PACKAGES=OFF \
-DTPL_ENABLE_AMD=ON \
  -DAMD_LIBRARY_DIRS="/usr/lib" \
  -DTPL_AMD_INCLUDE_DIRS="/usr/include/suitesparse" \
-DTPL_ENABLE_UMFPACK=ON \
  -DUMFPACK_LIBRARY_DIRS="/usr/lib" \
  -DTPL_UMFPACK_INCLUDE_DIRS="/usr/include/suitesparse" \
-DTPL_ENABLE_BLAS=ON \
-DTPL_ENABLE_LAPACK=ON \
$SRCDIR
```

Figure 3.1. cmake invocation for configuring serial Trilinos

Building Trilinos for Parallel **Xyce**

Building Trilinos for use in Parallel **Xyce** requires enabling one additional Trilinos package (Zoltan) and enabling ParMETIS library support. It also requires using the compiler wrappers provided by your system's OpenMPI package.

The CMake invocation of Figure 3.2 is an example invocation that would be appropriate on a Linux system that provides UMFPACK and AMD through SuiteSparse, but for which no package exists for ParMETIS. The notes in the serial build instructions also apply to Figure 3.2.

We assumed for this invocation that you have installed ParMETIS in `/home/me/XyceLibs/Parallel`, as described in the Appendix. If your system has provided ParMETIS, and it is installed in some other location, change the relevant lines of the cmake invocation (`-DParMETIS_LIBRARY_DIRS` and `-DParMETIS_INCLUDE_DIRS`).

Once the script is created and customized, run it. If it completes without error, Trilinos is configured for building. Build with "`make`" and install with "`make install`."

```
#!/bin/bash
SRCDIR= < location of the trilinos source code >
ARCHDIR=/home/me/XyceLibs/Parallel
FLAGS="-O3 -fPIC"
cmake \
-G "Unix Makefiles" \
-DCMAKE_C_COMPILER=mpicc \
-DCMAKE_CXX_COMPILER=mpic++ \
-DCMAKE_Fortran_COMPILER=mpif77 \
-DCMAKE_CXX_FLAGS="$FLAGS" \
-DCMAKE_C_FLAGS="$FLAGS" \
-DCMAKE_Fortran_FLAGS="$FLAGS" \
-DCMAKE_INSTALL_PREFIX=$ARCHDIR \
-DCMAKE_MAKE_PROGRAM="make" \
-DTrilinos_ENABLE_NOX=ON \
-DTrilinos_ENABLE_LOCA=ON \
-DTrilinos_ENABLE_EpetraExt=ON \
   -DEpetraExt_BUILD_BTF=ON \
   -DEpetraExt_BUILD_EXPERIMENTAL=ON \
   -DEpetraExt_BUILD_GRAPH_REORDERINGS=ON \
-DTrilinos_ENABLE_TrilinosCouplings=ON \
-DTrilinos_ENABLE_Ifpack=ON \
-DTrilinos_ENABLE_ShyLU=ON \
-DTrilinos_ENABLE_Isorropia=ON \
-DTrilinos_ENABLE_AztecOO=ON \
-DTrilinos_ENABLE_Belos=ON \
-DTrilinos_ENABLE_Teuchos=ON \
-DTrilinos_ENABLE_Amesos=ON \
   -DAmesos_ENABLE_KLU=ON \
   -DAmesos_ENABLE_UMFPACK=ON \
-DTrilinos_ENABLE_Sacado=ON \
-DTrilinos_ENABLE_Zoltan=ON \
-DTrilinos_ENABLE_ALL_OPTIONAL_PACKAGES=OFF \
-DTPL_ENABLE_AMD=ON \
   -DAMD_LIBRARY_DIRS="/usr/lib" \
   -DTPL_AMD_INCLUDE_DIRS="/usr/include/suitesparse" \
-DTPL_ENABLE_UMFPACK=ON \
   -DUMFPACK_LIBRARY_DIRS="/usr/lib" \
   -DTPL_UMFPACK_INCLUDE_DIRS="/usr/include/suitesparse" \
-DTPL_ENABLE_BLAS=ON \
-DTPL_ENABLE_LAPACK=ON \
-DTPL_ENABLE_ParMETIS=ON \
   -DParMETIS_LIBRARY_DIRS="$ARCHDIR/lib" \
   -DParMETIS_INCLUDE_DIRS="$ARCHDIR/include" \
-DTPL_ENABLE_MPI=ON \
-DTPL_MPI_LIBRARIES="" \
$SRCDIR
```

Figure 3.2. cmake invocation for configuring parallel Trilinos

4. Building **Xyce**

Xyce is built using the standard `configure` system. Most of the work getting **Xyce** configured involves getting `configure` to find the required libraries and headers. This is primarily accomplished by passing "LDFLAGS" and "CPPFLAGS" parameters in to `configure`. It may also be necessary to tell `configure` which compiler to use, or any special libraries it should link in.

The **Xyce** team recommends compiling **Xyce** "out-of-source." Create an empty directory and specify a full path to the **Xyce** `configure` script:

```
mkdir my_build
cd my_build
/path/to/Xyce/configure [configure options]
make
make install
```

The remaining sections below detail the configure options most likely to be required. A full list of configure's options may be obtained by typing:

```
/path/to/Xyce/configure --help
```

Configure options

Specifying compilers and helper programs

`configure` usually tries to find the C, C++, and Fortran compilers from a well-known list of standard compiler names. If your system does not have one of the standard-named compilers, or you wish to use some other compiler, you can specify them with `configure` variables. It is important to use the same compilers to compile **Xyce** that you used to compile the Trilinos libraries. Additional variables also allow you to specify particular versions of helper programs, like the Flex lexical analyzer generator. For example:

```
/path/to/Xyce/configure \
CC=strangeC \
CXX=strangeC++ \
F77=strangeF77 \
LEX=/usr/local/bin/flex
```

Here we have used the shell's continuation mechanism, and placed a backslash at the end of each line other than the last. It is important that no spaces appear after the backslashes.

Specifying compiler flags

Each compiler has a configure variable for passing it parameters. These variables are `CXXFLAGS` for C++ compiler flags, `CFLAGS` for C compiler flags, and `FFLAGS` for Fortran compiler flags:

```
/path/to/Xyce/configure \
CC=gcc48 \
CXX=g++48 \
F77=gfortran48 \
LEX=/usr/local/bin/flex \
CXXFLAGS="-O3" \
CFLAGS="-O3"
```

Specifying additional library search directories

If your libraries are installed in any directory that `configure` will not search by default, you can direct configure to search them by adding "`-L`" flags to the variable "`LDFLAGS`":

```
/path/to/Xyce/configure \
LDFLAGS="-L/some/random/directory/lib -L/home/me/archdir/ZipChip2000/lib"
```

Specifying additional include search directories

It is sometimes the case that system packages install headers in subdirectories of `/usr/include` or `/usr/local/include`. `configure` (and the C++ compiler itself) will not search for include files outside standard locations. If header files are installed in any directory that `configure` will not search by default, you can direct configure to search them by adding "`-I`" flags to the variable "`CPPFLAGS`":

```
/path/to/Xyce/configure \
CPPFLAGS="-I/home/me/archdir/ZipChip2000/include"
```

Example configure invocations for selected systems

Here we present example invocations of `configure` for selected systems. Because multiple versions of Trilinos might need to be built to supoort both serial and parallel builds of Xyce, we do not recommend installing Trilinos directly into system library and header directories. Therefore, in each example we have assumed that Trilinos (and any other hand-built libraries) have been installed into a directory called `/home/me/XyceLibs/Serial/lib`, and their associated headers into `/home/me/XyceLibs/Serial/include`. In the case of Trilinos, that can be accomplished by setting "`ARCHDIR`" to `/home/me/XyceLibs/Serial` in the CMake invocation script.

Serial Builds

Red Hat Linux

If all packages have been installed as recommended, configuring on Linux is fairly simple. This is because the package manager installs all of its packages in `/usr/lib` and `/usr/include`, and these are searched by configure and the compilers with no extra options required.

```
/path/to/Xyce/configure \
CXXFLAGS="-O3" \
LDFLAGS="-L/home/me/XyceLibs/Serial/lib" \
CPPFLAGS="-I/usr/include/suitesparse -I/home/me/XyceLibs/Serial/include"
```

Here, we have not had to specify anything more than the extra directories to search for our hand-built libraries and headers. Even the compilers are found automatically.

Mac OS X

If you have installed XCode, Fink, and all the Fink packages we have recommended, and installed Trilinos in /home/me/XyceLibs/Serial (per our example), building **Xyce** on Mac OS X in serial can be accomplished with the following `configure` invocation:

```
/path/to/Xyce/configure \
CXXFLAGS="-O3" \
LDFLAGS="-framework,Accelerate -L/sw/lib -L/Users/me/XyceLibs/Serial/lib" \
CPPFLAGS="-I/sw/include/suitesparse -I/sw/include -I/home/me/XyceLibs/Serial/include" \
CXX=g++48 \
```

```
CC=gcc48 \
F77=gfortran48
```

For a MacPorts installation, use the following:

```
/path/to/Xyce/configure \
CXXFLAGS="-O3" \
LDFLAGS="-framework,Accelerate -L/opt/local/lib -L/Users/me/XyceLibs/Serial/lib" \
CPPFLAGS="-I/opt/local/include -I/home/me/XyceLibs/Serial/include" \
CXX=g++48 \
CC=gcc48 \
F77=gfortran48
```

The `-framework,Accelerate` option in `LDFLAGS` tells configure to link in the optimized libraries provided by XCode. These libraries include BLAS and LAPACK, so when configure starts to look for those routines it will find them without having to search for libraries like libblas or liblapack. [1]

The other `LDFLAGS` and `CPPFLAGS` options tell configure to look for libraries and headers where Fink or MacPorts has installed them, and to look for Trilinos where it was installed.

While we have used `gcc48` in the above examples, the system `clang` compiler has also been used to compile **Xyce**. Note that `gfortran` is still required, since clang does not have a native Fortran compiler.

FreeBSD

If all libraries and packages have been installed as described above, serial Xyce can be configured on FreeBSD with the following `configure` invocation.

```
/path/to/Xyce/configure \
CXXFLAGS="-O3 -Wl,-rpath=/usr/local/lib/gcc46" \
LEX=/usr/local/bin/flex \
LDFLAGS="-L/usr/local/lib -L/home/me/XyceLibs/Serial/lib" \
CPPFLAGS="-I/usr/local/include -L/home/me/XyceLibs/Serial/include" \
CXX=g++46 \
CC=gcc46 \
F77=gfortran46
```

[1] While numerous **Xyce** developers have built the code with exactly the configure options shown above, it has been observed that in some combinations of package managers and XCode versions the build process has problems finding the Fortran versions of BLAS and LAPACK. When this happens, there will be a large number of link errors at the end of the build, reporting "undefined symbols for architecture" referenced by functions in Teuchos (a Trilinos package). Should this occur, one can add the `--disable-fortran_test` option to the `configure` invocation to make it use C versions of BLAS and LAPACK library functions instead.

In this example, we have used the GCC 4.6 compiler from the FreeBSD ports collection rather than the system gcc compiler (gcc and g++, versions 4.2). Because these compilers have incompatible C++ standard libraries that conflict with the system libraries, it is necessary to specify an "`rpath`" in the CXXFLAGS so that the resulting binary will always reference the correct C++ standard library. If both Trilinos and Xyce were built with the system gcc compiler (or with clang), this `rpath` would be unnecessary. GCC 4.6 and later will generally produce slightly faster executables, and should be preferred over the system GCC 4.2 compilers.

Cygwin

If all libraries and packages have been installed as described above, the following `configure` invocation will configure Xyce to be built under Cygwin:

```
/path/to/Xyce/configure \
LDFLAGS="-L/usr/local/lib -L/home/me/XyceLibs/Serial/lib" \
CPPFLAGS="-I/usr/local/include -I/home/me/XyceLibs/Serial/include" \
CXX=g++ \
CC=gcc
```

While it is tempting to let `configure` guess the C++ and C compilers, doing so causes problems in the link stage, when for some reason libtool is invoked using the C compiler as linker instead of C++ (it appears that the reason for this is that configure finds "CC" as a valid C++ compiler, but when this name is passed to libtool it is interpreted to mean "C mode"). This results in numerous inexplicable "undefined references" to standard C++ library functions.

Parallel Builds

Most of the complexity of building **Xyce** in parallel on any platform is in getting the third-party libraries built. If you have all the required packages for Trilinos built, and have properly built Trilinos using MPI compiler wrapper scripts, then configuring **Xyce** is usually just a matter of providing the same wrapper scripts for compilers and adjusting the configure line to point at the parallel versions of libraries.

In addition to naming the MPI compiler wrappers and identifying library directories, one must add "`--enable-mpi`" to the configure options.

Red Hat Linux

```
/path/to/Xyce/configure \
CXXFLAGS="-O3" \
LDFLAGS="-L/home/me/XyceLibs/Parallel/lib" \
CPPFLAGS="-I/usr/include/suitesparse -I/home/me/XyceLibs/Parallel/include" \
--enable-mpi \
CXX=mpiCC \
```

```
CC=mpicc \
F77=mpif77
```

Mac OS X

Fink:

```
/path/to/Xyce/configure \
CXXFLAGS="-O3" \
LDFLAGS="-framework,Accelerate -L/sw/lib -L/home/me/XyceLibs/Parallel/lib" \
CPPFLAGS="-I/sw/include/suitesparse -I/sw/include -I/home/me/XyceLibs/Parallel/include" \
--enable-mpi \
CXX=mpiCC \
CC=mpicc \
F77=mpif77
```

MacPorts:

```
/path/to/Xyce/configure \
CXXFLAGS="-O3" \
LDFLAGS="-framework,Accelerate -L/opt/local/lib -L/Users/me/XyceLibs/Serial/lib" \
CPPFLAGS="-I/opt/local/include -I/home/me/XyceLibs/Serial/include" \
--enable-mpi \
CXX=mpiCC \
CC=mpicc \
F77=mpif77
```

FreeBSD

```
/path/to/Xyce/configure \
CXXFLAGS="-O3 -Wl,-rpath=/usr/local/lib/gcc46" \
LEX=/usr/local/bin/flex \
LDFLAGS="-L/usr/local/lib -L/home/me/XyceLibs/Parallel/lib" \
CPPFLAGS="-I/usr/local/include -L/home/me/XyceLibs/Parallel/include" \
--enable-mpi \
CXX=/usr/local/mpi/openmpi/bin/mpiCC \
CC=/usr/local/mpi/openmpi/bin/mpicc \
F77=/usr/local/mpi/openmpi/bin/mpif77
```

Note the required "`rpath`" is still there, because the MPI wrappers will have the same libstdc++ issue that the serial g++46 compiler will.

Cygwin

The **Xyce** team has never built **Xyce** in parallel on Windows, and cannot suggest packages or build options for that system.

Appendix: Building the Prerequisite Libraries From Source

For the best instructions on building the libraries listed in Table 1.1, see the documentation included with those libraries. On some systems, these libraries may be available as pre-compiled packages, and are installable with a package management system such as `app-get`, `yum`, `fink` or `port`. Where a package is available for your system, it is usually preferable to use it.

While prebuilt packages can save one time, other considerations may require that one build from source code instead. Thus, these instructions will document how to build the individual components from the source for Umfpack, AMD, and ParMETIS.

Due to its license, ParMETIS is not available on most Linux package repositories. It will therefore be necessary to build ParMETIS from source on most Linux systems.

It is useful to keep the libraries needed for **Xyce** in a consistent location. This makes it easier to tell **Xyce**'s configure script where to find all the external libraries. Additionally, since some libraries require MPI to be enabled, it makes sense to keep separate library directories for MPI and non-MPI enabled code. Thus, we suggest you make two directories such as `XyceLibs/Serial` and `XyceLibs/Parallel`. Within these directories you should make `lib` and `include` directories. For the sake of consistency we will use `/home/me/XyceLibs` as the prototypical name for this directory in all our examples below.

Building UMFPACK and AMD

Download and unpack the UMFPACK/AMD source code. Compile it as per UMFPACK's instructions with whatever compiler you intended to use for **Xyce**. This requires hand-editing an include file that is used by the main Makefile. UMFPACK may give you the option of using its own version of BLAS or an external version if you have one. If you have an optimized version of BLAS then it is advised that you use it rather than the supplied version.

Once UMFPACK is built, you will need to copy two of the libraries it created to your `XyceLibs/Serial/lib` directory. These are libraries called `libamd.a` and `libumfpack.a` on most systems, or `libamd.lib` and `libumfpack.lib` under Windows. If you intend to build both serial and parallel versions of **Xyce** then you can copy these same libraries to both your serial and parallel `XyceLibs/Parallel/lib` directories, as they do not depend on MPI.

Building ParMETIS

ParMETIS is only required for parallel versions of **Xyce**. Thus, if you do not intend to build parallel **Xyce** then you may skip this section. Once you have downloaded and unpacked the ParMETIS source code, build it with the MPI-enabled compiler that you intend to use when building **Xyce**.

In older versions of ParMETIS (e.g. 3.1), this is accomplished by editing an appropriate "Makefile.in.XXXXX" file for your system (copy one that is similar if none is appropriate and edit it to fit), then copy that Makefile.in.XXXXX to Makefile.in, then run "make".

For the older version of ParMETIS, there is no install target in the Makefile. You will need to copy the two libraries `libmetis.a` and `libparmetis.a` to your `XyceLibs/Parallel/lib` directory, and copy `parmetis.h` and `METISLib/metis.h` to your `XyceLibs/Parallel/include` directory.

Recent versions of ParMETIS (4.0 and later) use CMake to configure. The **Xyce** developers have used two methods to install ParMETIS.

Method 1

When ParMETIS is first unpacked, one can type "`make`" for installation instructions. Following these instructions, simply type

```
make config prefix=/home/me/XyceLibs/Parallel
make
make install
```

One can specify the MPI compilers, if they are not the default `mpicc` by adding the `cc` and `cxx` options to the `make config` line. For example,

```
make config cc=openmpicc cxx=openmpicxx prefix=/home/me/XyceLibs/Parallel
```

Once ParMETIS is installed, the same procedure must be performed in the `metis` sub-directory.

Method 2

Provide the C compiler wrapper for MPI to the CMake invocation. Create an empty build directory, then execute CMake:

```
cmake /path/to/parmetis/source \
-DGKLIB_PATH=/path/to/parmetis/source/metis/GKlib \
-DMETIS_PATH=/path/to/parmetis/source/metis \
-DCMAKE_INSTALL_PREFIX=/home/me/XyceLibs/Parallel \
-DCMAKE_C_COMPILER=mpicc
```

After configuring ParMETIS 4.0 in the appropriate manner, simply type "make". You will then need to manually copy the required libraries and headers. From your build directory:

```
cp libmetis/libmetis.a /home/me/XyceLibs/Parallel/lib
cp libparmetis/libparmetis.a /home/me/XyceLibs/Parallel/lib
```

```
cp /path/to/parmetis/source/include/parmetis.h \
     /home/me/XyceLibs/Parallel/include
cp /path/to/parmetis/source/metis/include/metis.h \
     /home/me/XyceLibs/Parallel/include
```

Windows

The **Xyce** team has never built or tested **Xyce** in parallel on Windows, and has no guidance for building any of its third-party libraries for parallel in that environment.

References

[1] Eric R. Keiter, Ting Mei, Thomas V. Russo, Eric L. Rankin, Richard L. Schiek, Heidi K. Thornquist, Jason C. Verley, Deborah A. Fixel, Roger P. Pawlowski, and Keith R. Santarelli. Xyce parallel electronic simulator: User's guide, version 6.0. Technical Report SAND2013-WWWW, Sandia National Laboratories, Albuquerque, NM, 2013.

DISTRIBUTION:

1 MS 0899 Technical Library, 9536 (electronic copy)

Sandia National Laboratories