Building Calculix with Code Blocks

by Harry G. Schaeffer

Schaeffer Enterprises

April, 2020

Problem Statement

The first generation of FEA software was written using FORTRAN and targeted mainframe computers that were in use in the 1960s. Since that time a large number of applications have been developed and deployed to automate the build process, includinf the GNU “make” utility and Integrated Development Environment (IIDE) such as Micrsoft’s Visual Studio. This article describes the use of an Open Source application called Code Blocks to build the Open Source Calculix finite element application.

Background and Previous Work

This paper reflects lessons learned in over 40 years of using and developing software in the Computer Aided Engineering space. These lessons include:

1. The importance of a project plan
2. Understanding the development environment

Nastran

The NASA determined that it was in the national interest to create a software system for simulating the structural behavior of large aerospace systems. Several companies responded to the Request for Proposal and after the dust settled the contract was awarded to the Computer Science Corporation (CSC), whose technical partner was the MacNeal-Schwendler Corporaration (MSC). This was a fortunate paring of companies that understood the fledgling fields of computer science and finite elements, respectively.

The documentation was developed first and then the program was written by the combined team. Looking at the architecture of the final product in appears that CSC was responsible for the architecture of the Nastran Operating System (NOS), and MSC was responsible for the engineering mechanics and associated software. The fact that CSC had expertise in the architecture ot the Univac computer system is probable cause for the NOS being similar to OS for the Univac 1108, one of major computers at that time (1960).

At that time memory as scarce, and cards and tape drives were used for persistent storage. The Nastran team therefore decided to use tables and matrices as the primary

variables which were data structures which would then be stored on tapes. (Cards were used as the primary input media; perhaps the reason that old-timers still refer to the input data as “cards”.

Looking at the antique code in the NASTRAN-95 delivery one sees the computer science associated with mapping very large codes to very small memory, called the “core”, using a facility called “overlaying” and the partitioning of the code modules into “links”. Nastran had a table that resided in core andt mapped the Nastran Modules to overlay links.

Pretty complicated stuff: writing the program was the easy stuff. It is hard to imagine the difficulty of developing a large application using card input and managing storage on tape drives.

Calculix

This a marvelous application that performs simulations of non-linear field equations using the Finite Element Method. The development work was done at MTU Aero Engines GmbH (Muenich, Germany) in 1987 by Guido Dhondt. (As a side note, I started my career in 1958 at AiResearch of Arizona that was developing small gas turbines. At that time most simulation was done using Friedan calculators and analog computers. A very smart and able person named Dr. Monty Steele in the stress and vibration group designed and developed a structural simulation that was run on the IBM 704. I did not have the background in engineering mechanics then, and it wasn’t until working on a PhD at VPI in the early 1963 that I realized that Dr. Monty Steele probably developed the first application using what became known as the Finite Element Method.) Calculix is exactly the technology require to simulate the entire function of the jet engine, IMHO.

Integrated Development Environment (IDE)

The tools used to control the actual build of Nastran in the 1960s included a Compiler and a Linker and since the makefile utility was created by Feldma in April 1976, it certainly didn’t have make. Be that as it may be; the fact is that make has been one of the major tools used to define and automate the build process. And, it is used it the define the build of Calculix and it’s two solution libraries, ARPACK for eigenvalues; and, SPOOLES to matrix decomposition and forward-backward substitution.

For those who want to study an excellent well-documented system of make files, please read the SPOOLES documentation of it's build process.

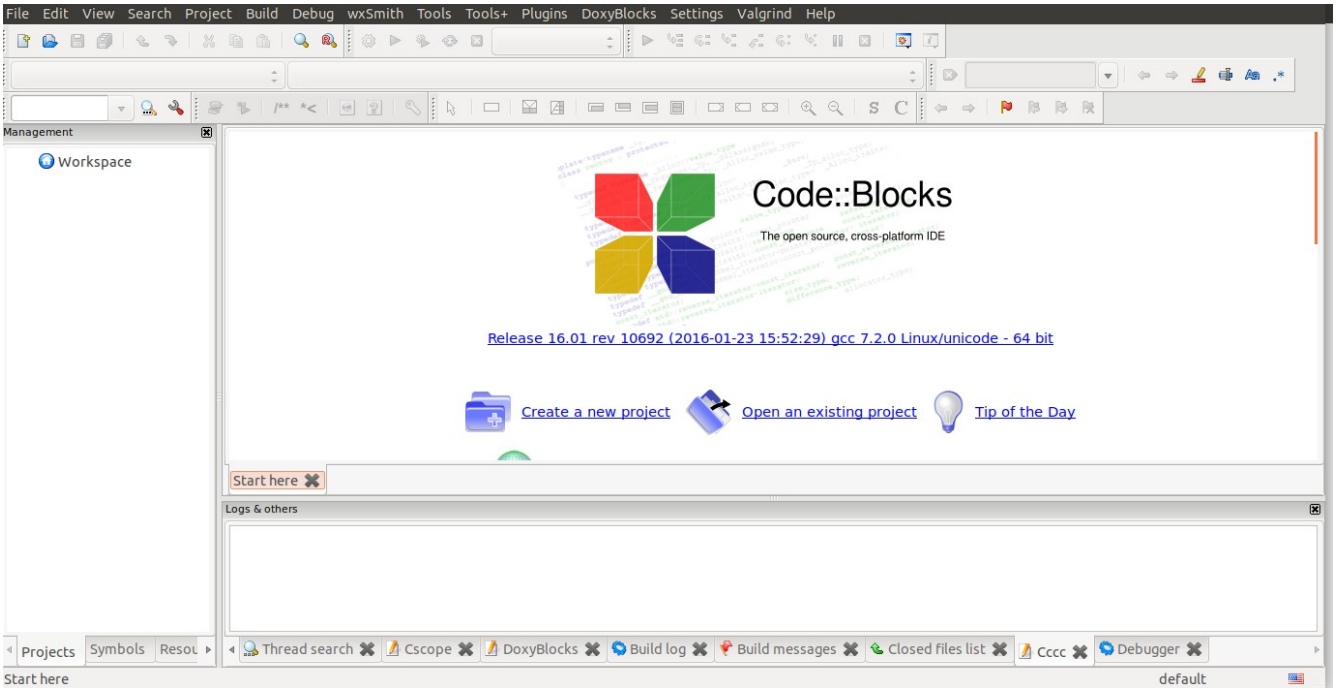
At any rate, creating a makefile system such as that used by SPOOLES is beyond the skill set for the average developer so IDE's soon appeared that mechanized this task. The Visual Studio developed by Microsoft is the best. The author has used Visual Studio 6.5 and it's companion, Visual Fortran running on Windows XP since the late 1990s. However process in the field of compiler development as evidenced by GNU gfortran and gcc; the demise of Windows XP; the emergence of Linux as an alternative to Windows; the fact that Visual Studio no longer supports Fortran; and, the addition of the Windows Subsystem for Linux, makes Code Blocks an attractive alternative.

Code Blocks (C::B)

C::B and the GNU tool set are included in Ubuntu 18.04 for sure and perhaps in earier releases. The frequently used applications used for the current discussion are shown in this screen image:

They include Code Blocks, Geany, Files, Document Viewer, and Texstudio. If these applications are not installed please do so.

Click the Code::Blocks IDE icon; after a few moments the following screen will be displayed.



So far so good. Instead of doing the “Hello World” demo Let me describe what I have learned about C::B:

1. A Workspace is a file that includes references to all projects that define an application such as Nastran or Calculix. The associated workspace definition is an XML fire that is stored in your home directory, “~/”, which expands to /home/<your_user_name>/.config/codeblocks/<target>.workspace.
2. A Project is one of one of the categories shown by the menu displayed by selecting “Create a new project”. The data defining a project is written to an XML file having the extension, “.cbp”, that is saved at a location of your choosing. Since it is convenient to save all related projects in the same directory it is wise to define a directory called project in which all project related files are saved.
3. A project will contain one or more source files and will generate on or more object files and one target that can be a static library having an extension, “.a”, a shared object file having an extension, “.so”, or an application file havint the name of the project.
4. The project will contain the following directories that contain the object tiles:
    1. /bin/debug and bin/release
    2. /obj/debug and /obj/release

5. The source files are saved in the XML file for the project.
6. If a project includes Fortran Modules, the ".mod" files generated when the module is compiled are saved in the project /obj directory. It should also be noted that the .mod files are not updated when the associated module file is recompiled. In addition, the .mod files must be identified by specifying their path in search directories in build options. Fortran Modules are a real pain to deal with. Perhaps another option such as that used in SPOOLES should be considered. That won't be an issue in the Calculix application but it is in Nastran which uses them.

Building Calculix on Linux

Important Linux Directories

There is a learning curve for those using Linux for the first time: But many find it difficult to return to Windows after get used to Linux. The directory system is different: there is no C: drive; there are only directories. Use the web to find information on "linux directories" such as that found here.

Important Linux Tools

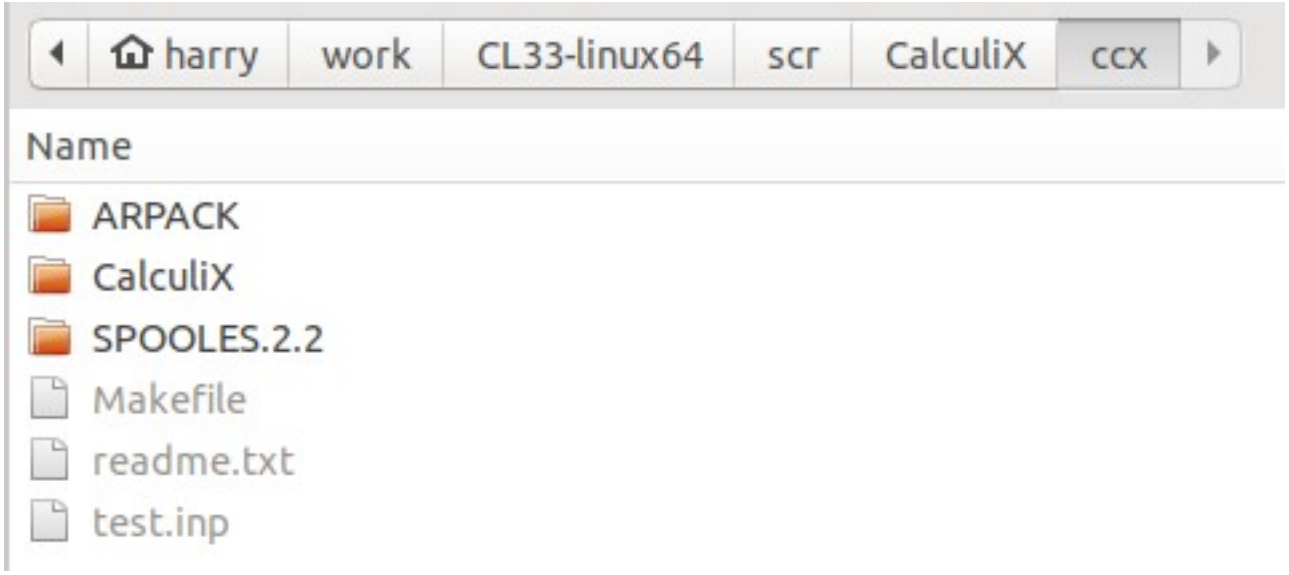
Many of the directories are restricted to root or super users so for many tasks you must identify yourself using the "sudo" command. When prefacing a command using sudo you will be asked to enter your user name.

A tutorial on linux commands is given here.

Installing Calculix

Calculix can be downloaded from a number if sites that can be found by weaching the web. After downloading a tar file will be saved in your Download directory. Now open a Terminal window in your HOME directory and create a new directory called work. Then using the File app right click the work icon and select "Open in terminal". Using this terminal window make a directory called calulix.

The next step is to copy the download file to ~/work/calculix and to extract the contents which is left to you. Now use the file manager (FM) to open, successively: calculix/src/ Calculx/ccx. The FM now displays:



Build ARPACK and SPOOLES Libraries

Both Arpack and Spooles will be built using the makefiles in their respective directories

Spooles Library

The SPOOLES.2.2 directory contains a number of sub directories including one called documentation. Now is a good time to open ‘Texstudio’ by clicking on the app icon. Click “Open” and in the application open main.tex. Then select Build & View from the Tools drop-down. The Spooles Installation manual is now displayed on the right hand window. In this window click the leftmost icon on the top to open an external viewer. Please print a copy for ready reference.

It is worth your time to read this manual since it describes the directory structure and the associated make files.

The only make file that requires modification is Make.inc in the top level spooles directory. Following the guideline in “Section 2 Makefiles” the Make.inc for Linux is:

.POSIX:

#--

#

file created 98jun18, cca, (cleve.ashcraft@boeing.com)

```
#  based on work by clay breshears (clay@turing.wes.hpc.mil)
#  (much appreciated)
#
#---------------------------------------
#
#  place your favorite compiler here
#
# for solaris
#
  CC = gcc
# CC = /usr/lang-4.0/bin/cc
#
# for sgi
#
# CC = cc
#
# for hp
#
# CC = /opt/mpi/bin/mpicc
#
#---------------------------------------
#
#  set the compiler flags
#
# OPTLEVEL =
# OPTLEVEL = -g -v
```

```
  OPTLEVEL = -O
# OPTLEVEL = -xO5 -v
# OPTLEVEL = -O3
# OPTLEVEL = -O4
# CFLAGS = -Wall -g
# CFLAGS = -Wall -pg
# CFLAGS = $(OPTLEVEL) -D_POSIX_C_SOURCE=199506L
  CFLAGS = $(OPTLEVEL)
# CFLAGS = -Wall $(OPTLEVEL)
#
#---------------------------------------
#
#  set any load flags
#
#  LDFLAGS = -Wl,+parallel -Wl,+tm,spp2000 # for hp exemplar
   LDFLAGS =
#
#-----------------------------------------------------------------
#
#  set any thread libraries
#
# THREAD_LIBS =
# THREAD_LIBS = -D_REENTRANT=199506L -lpthread
  THREAD_LIBS = -D_POSIX_C_SOURCE=199506L -lpthread
# THREAD_LIBS = -lpthread
#
```

```
#-----------------------------------------------------------------
#
#  set the purify environment (a memory monitoring tool)
#
  PURIFY =
# PURIFY = /usr/local/purify-4.0.1/purify
#
#  purify wouldn't work with the newest version of the gcc library,
#  so we had to force loading the old version
#
  PURIFY_GCC_VERSION =
# PURIFY_GCC_VERSION = -V 2.7.2
#
#-----------------------------------------------------------------
#
#  set the archive flags
#
  AR = ar
  ARFLAGS = rv
#
#-----------------------------------------------------------------
#
#  set the ranlib environment
#  (if ranlib is not needed, we echo the library name)
#
# RANLIB = ranlib
```

```
  RANLIB = echo
#
#----------------------------------------------------------------
#
#  set suffix rule *.c --> *.o
#
.c.o :
        $(PURIFY) $(CC) -c $(CFLAGS) $<
#
#----------------------------------------------------------------
#
#  set suffix rule *.c --> *.a
#
.c.a :
        $(PURIFY) $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
#
#----------------------------------------------------------------
#
#  MPI install library
#
# MPI_INSTALL_DIR =
  MPI_INSTALL_DIR = /usr/local/mpich-1.0.13
#
#----------------------------------------------------------------
```

```
#
#  MPI library path
#
# for sgi
#
# MPI_LIB_PATH =
#
# for solaris
#
  MPI_LIB_PATH = -L$(MPI_INSTALL_DIR)/lib/solaris/ch_p4
#
# for hp
#
# MPI_LIB_PATH =
#
#-----------------------------------------------------------------
#
#  MPI libraries
#
# for solaris
#
  MPI_LIBS = $(MPI_LIB_PATH) -D_REENTRANT -lmpi -lsocket -lnsl -lthread
#
# for sgi
#
# MPI_LIBS = $(MPI_LIB_PATH) -lmpi -lpthread
```

```
#
# for hp
# MPI_LIBS = -lpthread
# MPI_LIBS = $(MPI_LIB_PATH) -lpthread
#
#------------------------------------------------------------------
#
#  MPI include path
#
# MPI_INCLUDE_DIR =
  MPI_INCLUDE_DIR = -I$(MPI_INSTALL_DIR)/include
#
#------------------------------------------------------------------
```

Setting Thread Type

Following the suggestions in Section 3, make the changes noted in the MT directory.

Then, open a Terminal in the top level spooles directory and enter:

make lib

This will create spooles.a in the /src directory and spoolesMT.a in the /*MT/*src directory.

Arpack Library

Using the FM view the contents if the ARPACK directory. The Makefile in this directory includes Armake.inc which, ar line 28 defines the path, home as:

home = /home/harry/work/CL33-linux64/scr/CalculiX/ccx/ARPACK

This MUST be changed to reflect that appropriate for your directory. After doing, open a terminal in this director and enter:

make

This will create a library in the directory named libarpack_linux.a.

Build Calculix

The libraries required to build calculix have now been created. Before opening Code Blocks, add a new directory in the /work/calculix directory called, “cbprojects”. Then open C::B and right-click the workspace icon in the left pane and choose “save workspace as”. Save as “CCX.worlspace”, noting that the file is saved in your home directory as:

~/.config/CCX.workspace

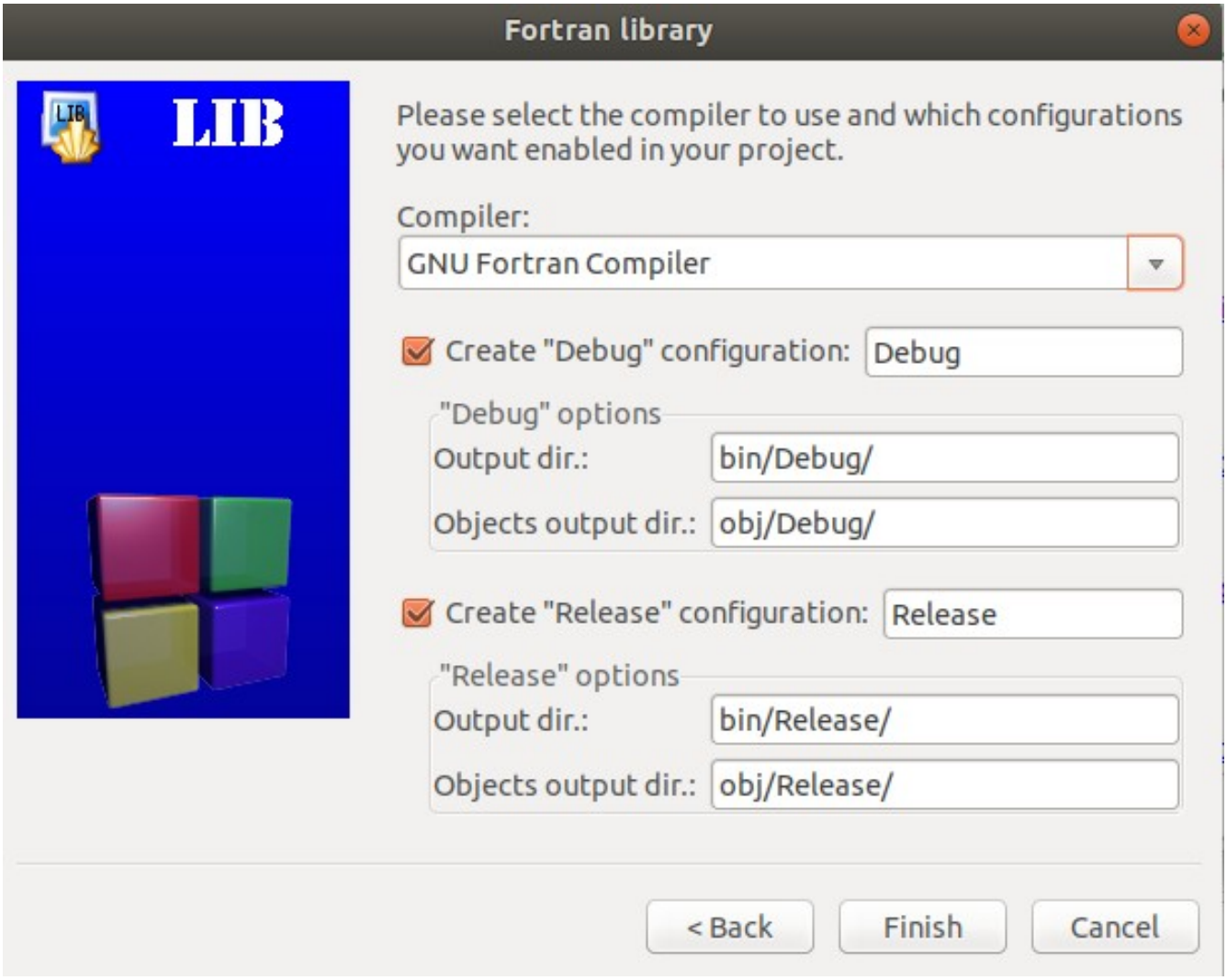
Calculix Library

The source code path is:

path = ~/work/CL33-linux64/scr/CalculiX/ccx/CalculiX/ccx_2.15/src

where the tide (~) is a shortcut for your HOME directory, and the top level Calculix directory in this article is CL33-linux64 so you should use the name chosen when you installed Caculix. Opening Makefile.inc you will note that make variable SCCXF includes all the Fortran files, SCCXC includes all the c files, and SCCXCXX includes a c++ source. Using the GNU toolset, the same compiler, gfortran, can be used to build mixed fortran, c and c++.

In C::B choose **file>new>project** and in the form that is displayed choose “Fortran Library” and click “Go”. Then in the form: set Project name = “SCCX”; select the /work/calculix/cbproject in the directory, recalling that this was created in a previous step; click next to open the Fortran Library form. Then set compiler =”**GNU Fortran Compiler**” The completer form then appears as follows:



Before selecting Finish: make sure the Debug and the Release versions are selected. C::B will then create the SCCX directory and sub directories as noted when finish is selected.

Note that the C::B form has a select box at the top that allows you to select Debug or Release build. Select Debug and before adding source files to the project: click Fortran sources then Right-Click (R-C) main.f and in drop down select **Remove file from project**.

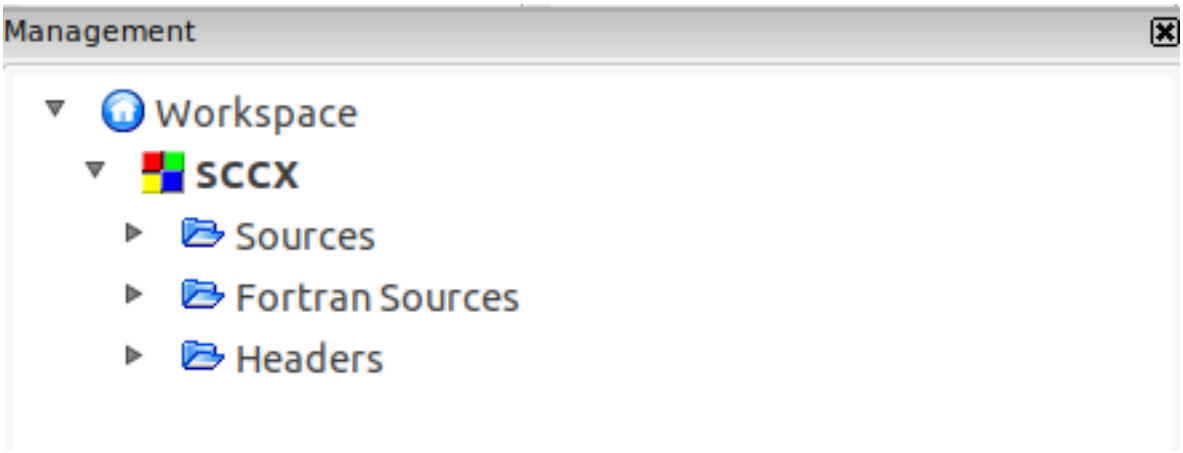
At this point:

1. The SCCX project has been defined having two versions: Debug and Release.
2. C::B has created a directory named SCCX in the cbprojects directory and associated sub directories for each version.

You now need to define the build parameters using the build options for project and add the source files files using the add files option.

Source files are added to the project by: R-C the project name and then choosing add files. In the resulting dialog, move to the Calculix src directory and at the bottom set the filter to “Fortran 77”; select all the sources (shift-Ctrl) and click open. Then repeat for c/c++ files.

The Management pane now looks like this:



Where Sources contains all c/c++ files, Fortran Sources contains all Fortran files and Headers contains the c header tile, including Calculix.h. In Sources R-C ccx-2.15.c and choose “**Remove file from project**”; and in Fortran Sources remove gauss.f and xlocal.f, which are include files that will be added in a latter step. In Headers: open Calculix.h and modify the definitions of FORTRAN and CEE macro definitions as follows:

#define FORTRAN(A,B) A##_ B

#define CEE(A,B) A##_ B

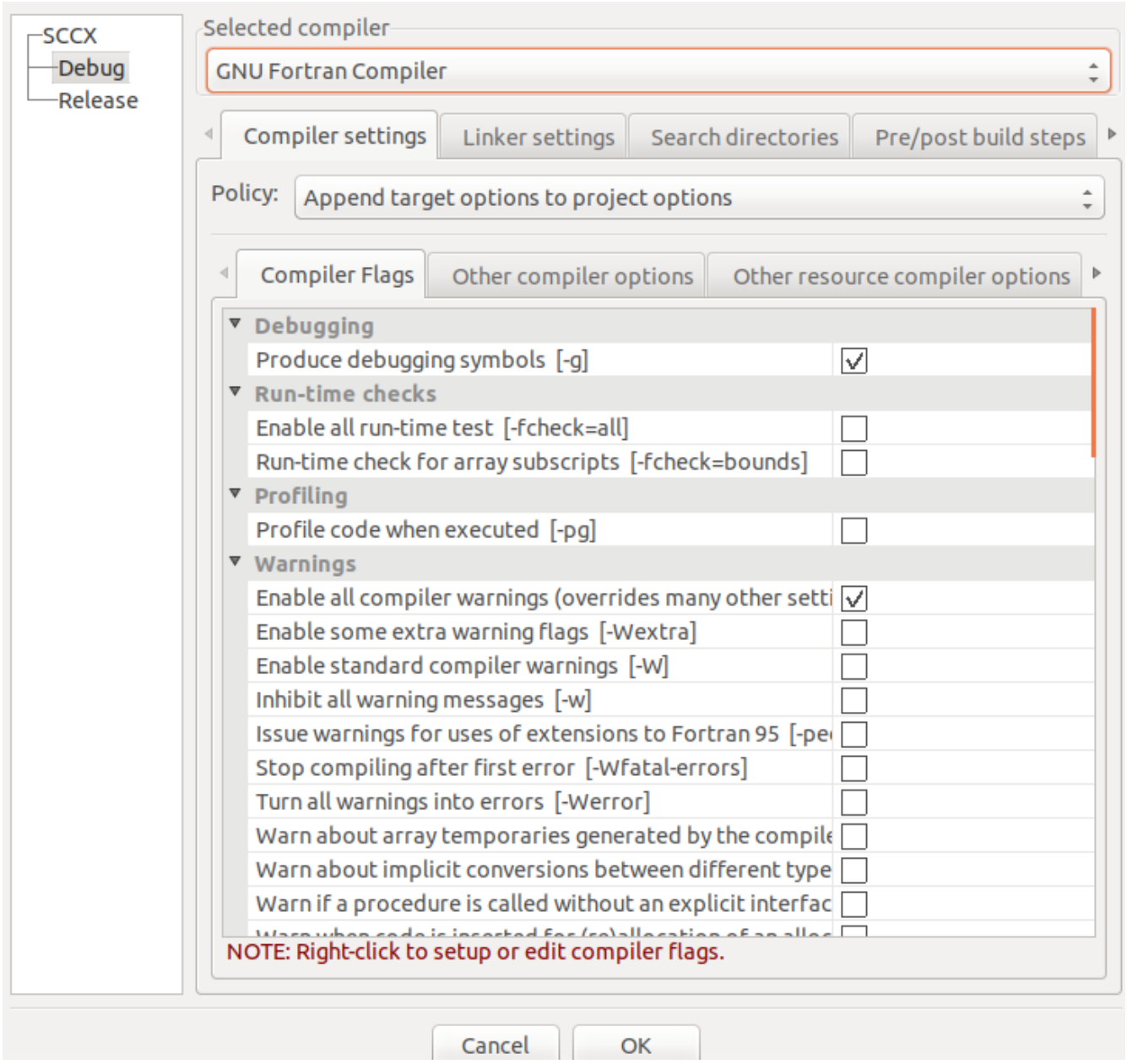
and remove other options. (Note: This an interesting way to get c and Fortran to talk together. It works so why change it)

That takes care of the files to be included in SCCX. Before adding the build information to the project open the Makefile in the /src directory: Note the following:

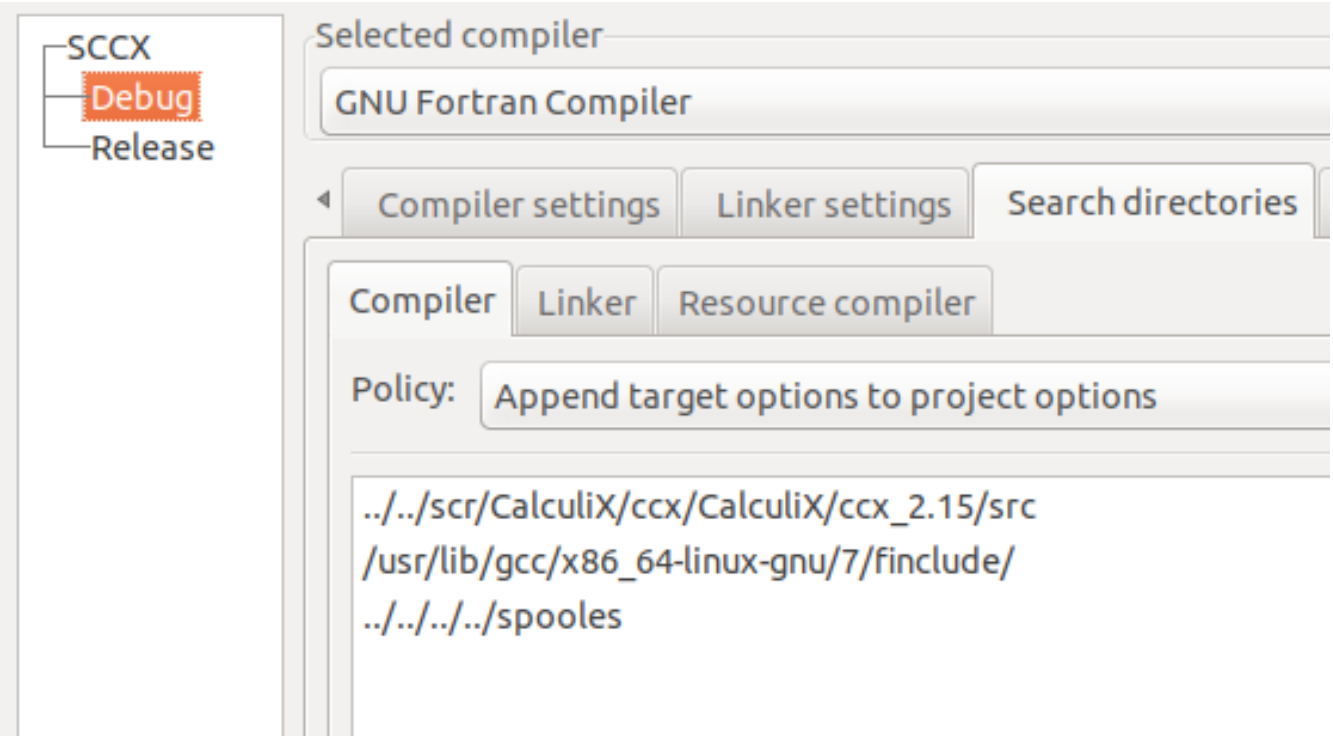
1. FC=gfortran defines the Fortran compiler that SCCX will use for both *c and *f files.

2. FFLAGS sets the optimization which for SCCX and ccx projects will be set using build options as well as the -openmp flag.
3. SCCXMAIN=ccx_2.5.c defines the main routine.
4. LIBS are:
    1. spoolesMT.a
    2. Spooles.a
    3. libarpack_linux.a
    4. libpthread.a
    5. libm.a

In C::B, define the actual build information by right_clicking on SCCX and selecting **build options** to display the following form:



The form allows one to select various compiler options: notice that the the -g options is selected. Select the **Search directory** that will be used to find the include files by clicking the tab followed by clicking the **add** button. Then select the /ccx_2.5/src directory, keep the path relative. Continue to add the spooles directory and the usr/lib/gcc/x86+65-linux-gnu/7/finclude directory. Then click OK to produce:



Finally click OK to close the build options.

Now right-click (R-C) SCCX and select build in the menu. After completing the build, notice that the Build log tab is active and the associated pane contains the following:

Output file is bin/Debug/libSCCX.a with size 16.46 MB

Output file is bin/Debug/libSCCX.a with size 16.46 MB

Process terminated with status 0 (0 minute(s), 54 second(s))

0 error(s), 0 warning(s) (0 minute(s), 54 second(s))

Build log saved as:

file:///home/harry/work/CL33-linux64/projects/SCCX/SCCX_build_log.html

Opening the build log in the browser displays the complete build history.

Build Main

Create a new Fortran Application project named “SCCXMAIN, taking care to use the gortran compiler, and that both Debug and Release versions are to built in the /work/calculix/cbproject directory. Select Debug version and click OK. Click Fortran sources and remove main.f. (C::B rather annoyingly always adds main.f!)

Now add the ccx_2.15.c and Calculix.h, as modified for SCCX, to the project.

The main driver,ccx_2.15.c, must be modified so that the spooles solver is used by adding the following after the code that conditionally sets the variable, isolver:

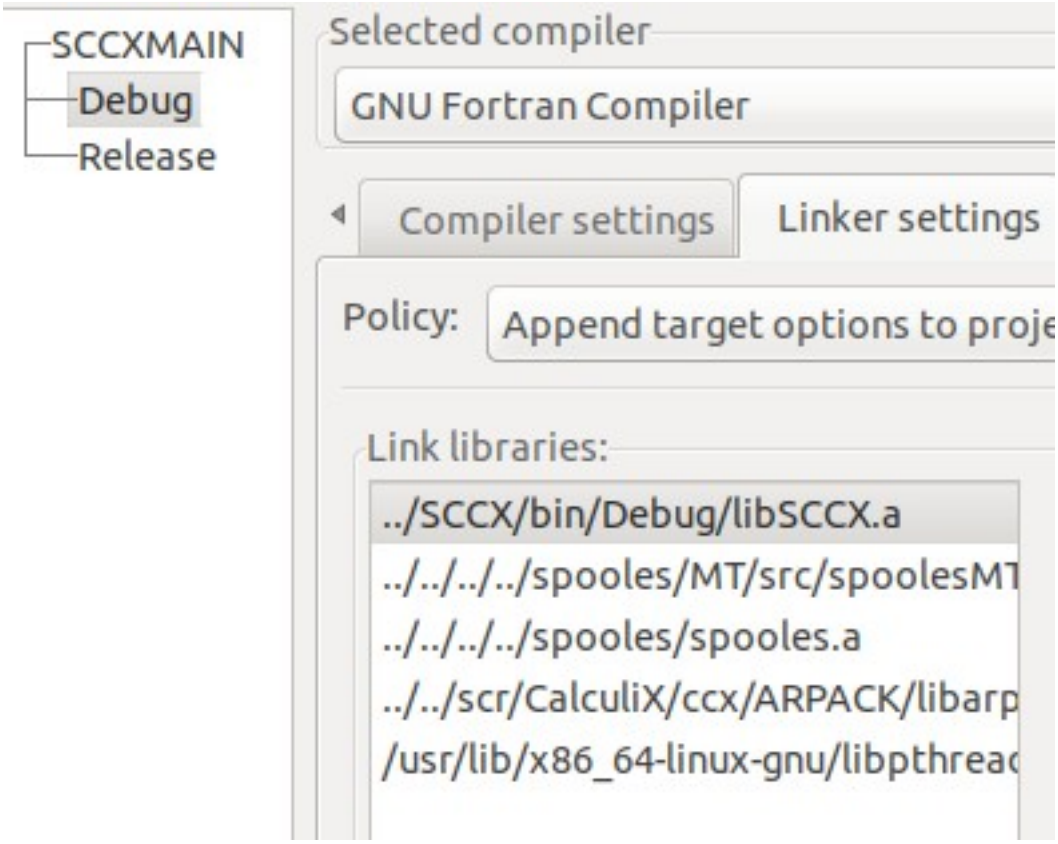
```
/*
INSERT AFTER LINE 161 TO SET SOLVER DEFAULT TO SPOOLES
*/
isolver = 0;
```

Add Libraries to SCCXMAIN

The libs are added to the build in the order defined in the Make file by:

1. Opening build options
2. Setting compiler flags -w and -fopenmp
3. Selecting Linker settings tab; and, the clicking the Add button for each library to be added
4. Adding libSCCX.a
5. Adding libspoolesMT.a (using the path to your spooles build)
6. Adding libspooles.a (using the path to your spooles build)
7. Adding libarpack_linux.a from the path to your ARPACK build
8. Adding libpthread.so where the path, /usr/lib/x86_64-linux-gnu/libpthread.so was found using the linux command, “locate libpthread.so”

The Build Options Link files now looks like this:



Build by clicking OK and return to the project menu and then select build. The Build log shows that the build succeeded and that the executable is saved in cbprojects/SCCXMAIN/bin/Debug/SCCXMAIN.

Test the Build

To test the build, run the file beam20p.inp that is in the directory ~/work/CL33-linux64/hlp/verifsamples. This can be run by:

1. Activating the SCCXMAIN projects
2. From the Projects menu, select Set program arguments opening the “Select target form”. Fill in the Program argument field to; /home/harry/work/CL33-linux64/hlp/verifsamples/beam20p > aaaa. Change the user name, harry, to your user name and pipe the results to a file called “aaaa” that will be written to projects/SCCXMAIN/bin/Debug/SCCXMAIN.
3. Click the green triangle in the C::B form

An X TERM window will open and display very little since the output is written to the aaaa file that contains the following:

****

CalculiX Version 2.15, Copyright(C) 1998-2018 Guido Dhondt
CalculiX comes with ABSOLUTELY NO WARRANTY. This is free
software, and you are welcome to redistribute it under
certain conditions, see gpl.htm

****

You are using an executable made on Sa 15. Dez 15:34:34 CET 2018
Decascading the MPC's

Determining the structure of the matrix:
number of equations
720
number of nonzero lower triangular matrix elements
37458

Using up to 1 cpu(s) for the stress calculation.

Using up to 1 cpu(s) for the symmetric stiffness/mass contributions.

Factoring the system of equations using the symmetric spooles solver
Using up to 1 cpu(s) for spooles.

Using up to 1 cpu(s) for the stress calculation.

The numbers below are estimated upper bounds

number of:

nodes: 261
elements: 32
one-dimensional elements: 0
two-dimensional elements: 0
integration points per element: 27
degrees of freedom per node: 3
layers per element: 1

distributed facial loads: 0
distributed volumetric loads: 0
concentrated loads: 9
single point constraints: 63
multiple point constraints: 1
terms in all multiple point constraints: 1
tie constraints: 0
dependent nodes tied by cyclic constraints: 0
dependent nodes in pre-tension constraints: 0

sets: 5

terms in all sets: 100

materials: 1

constants per material and temperature: 2

temperature points per material: 1

plastic data points per material: 0

orientations: 0

amplitudes: 4

data points in all amplitudes: 4

print requests: 2

transformations: 0

property cards: 0

STEP 1

Static analysis was selected

Job finished

The output is saved in the same directory as that specified for the input: you can compare the current results to those in the associated archive file beam20p.dat.ref.

Debugging Calculix

Suppose you needed to debug Calculix. In that case, set a break-point at line 107 in ccx_2.5.c by opening the file in SCCXMAIN and then clicking on the line between the line number and the line on the right . C::B then displays a red circle there as shown

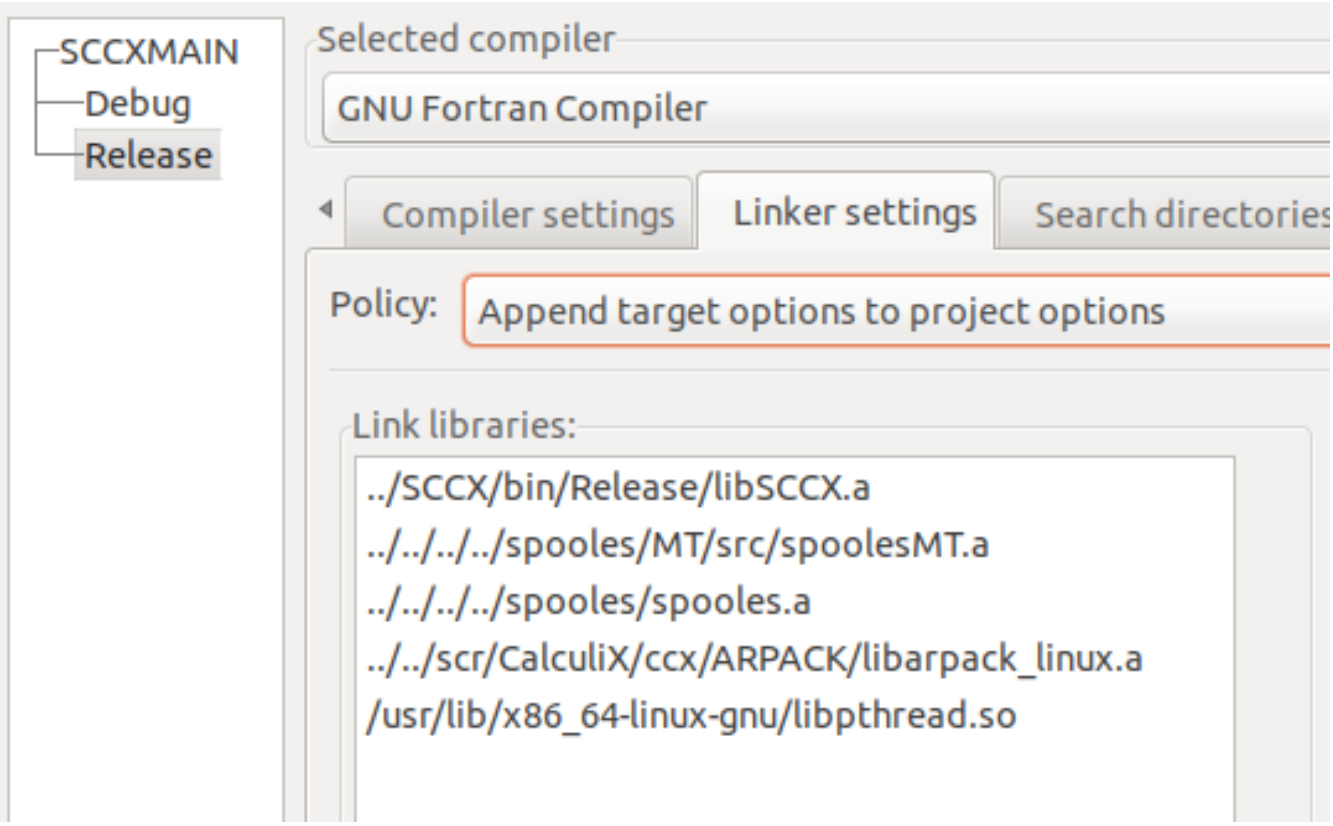
```
 96   double ctrl[36]={4.5,8.5,9.5,10.5,10.5,4.5,0.,5.5,0.,0.,0.25,0.5,0.75,0.85,
 97
 98   double fei[3],*xmodal=NULL,timepar[5],
 99       alpha,ttime=0.,qaold[2]={0.,0.},physcon[13]={0.,0.,0.,0.,0.,0.,0.,0.,0.
100
101   #ifdef CALCULIX_MPI
102   MPI_Init(&argc, &argv) ;
103   MPI_Comm_rank(MPI_COMM_WORLD, &myid) ;
104   MPI_Comm_size(MPI_COMM_WORLD, &nproc) ;
105   #endif
106
107   if(argc==1){printf("Usage: CalculiX.exe -i jobname\n");FORTRAN(stop,());}
108   else{
109     for(i=1;i<argc;i++){
110       if(strcmp1(argv[i],"-i")==0) {
111       strcpy(jobnamec,argv[i+1]);strcpy1(jobnamef,argv[i+1],132);jin++;break;
```

Now click the red triangle at the top of C::B and the program starts and stops at line 107 as denoted by the yellow triangle in the red circle. To find the value of the argc variable, in the Command window at the bottom, enter “p argc”. The GDB then evaluates the variable and displays $1=2 in the Logs & others pane.

For information about navigating the program with the debugger please go here.

Building Release Version

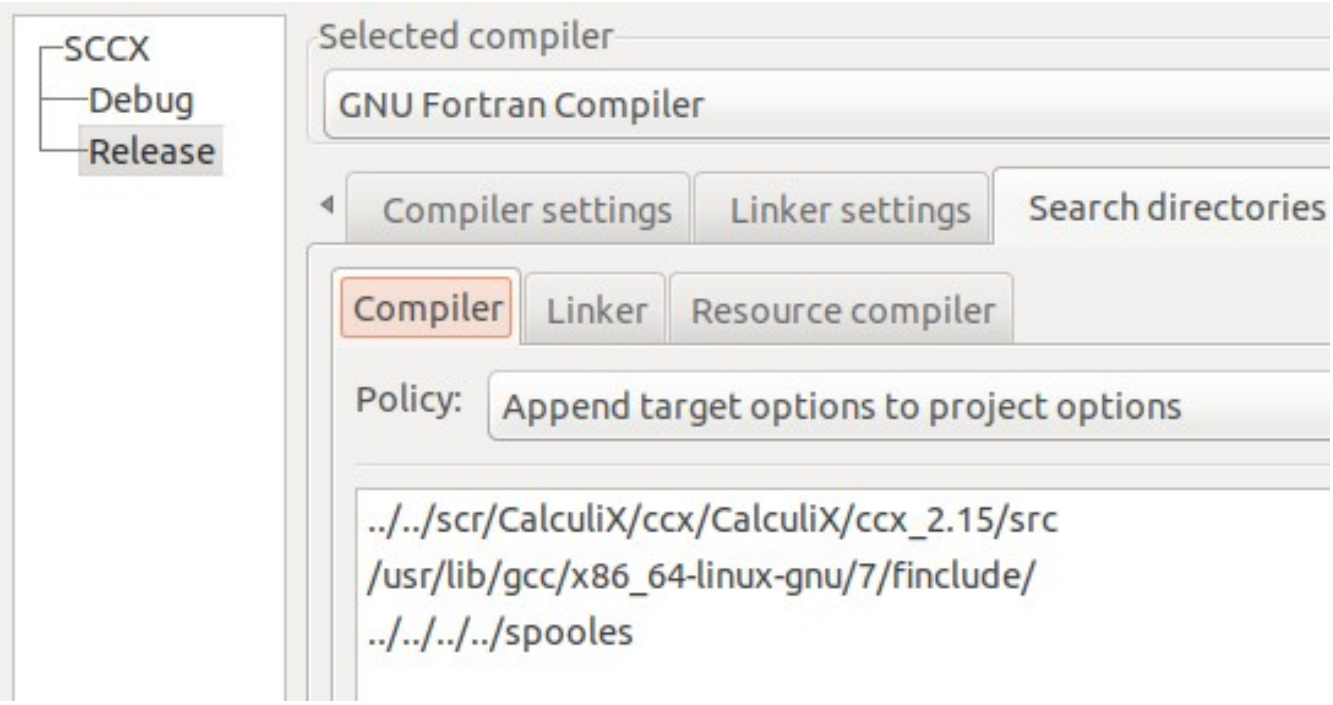
Firstly, the Release version must have been specified when the project was created. If you failed to do that the project definition file that has a cbp extension must be modified outside of C::B. However, that is beyond the scope of this article. Make sure you select both Debug and Release versions!



Assuming that you have built the Debug version, the select Release at the top of the C::B form. That done, make SCCXMAIN active and open the Build options menu and tadd link libraries as shown:

where libSCCX.a is in the /bin/Release directory and where the only other change is adding the openmp to compiler settings.

The same procedure is taken to create the Release version of SCCX library so that the Search Directories are as shown.



Now build the SCCX project and then build SCCXMAIN. The executable will now be in the /bin/Release directory. Test this build by running beam20p and verify the results are correct.

Summary

The article explores the use of the Code Block IDE to build the Calculix FEA application as an alternative to the use the Make files that are included in the release. The Ubuntu 18.04 build is used.