Crosscompilierung

. . . ein Windows-Installer mit 4 Befehlen

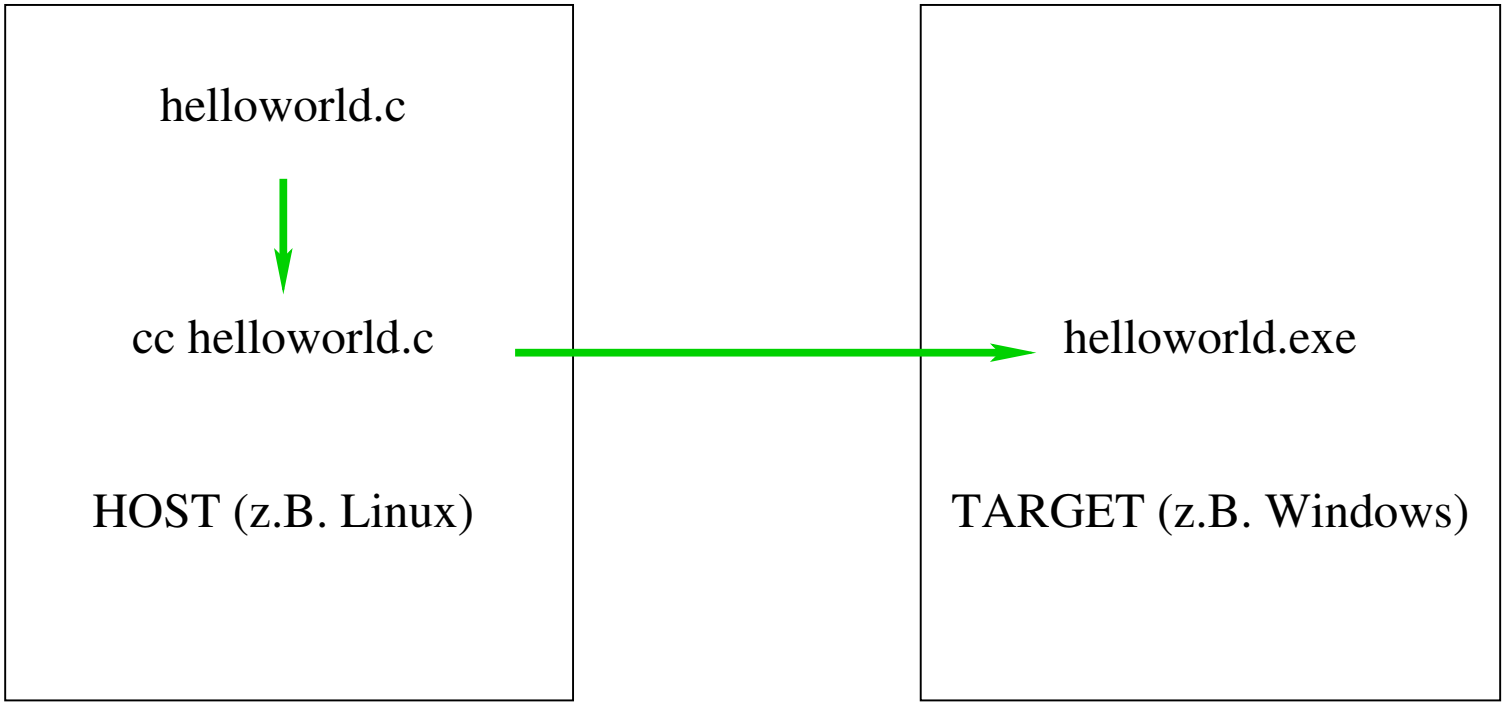
Wolfgang Dautermann

FH JOANNEUM

Linuxwochen Linz 2016

Cross-Compiler

Ein Cross-Compiler ist ein Compiler, der auf einer Plattform läuft, aber Compilate (Executables) für eine andere Plattform erzeugt.



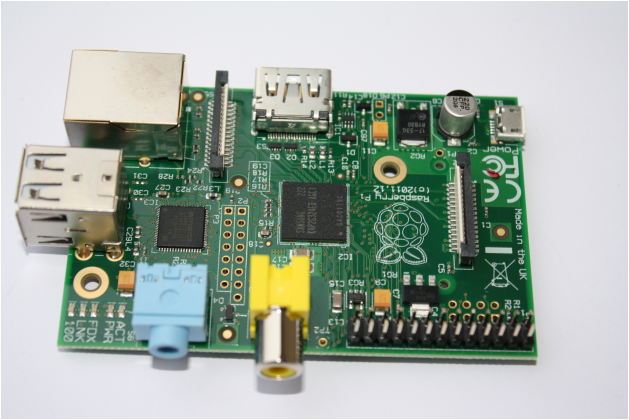
Cross-Compiler

Dabei können sich sowohl am Build-Rechner als auch auf der Zielplattform **alle** Komponenten unterscheiden:

- Betriebssystem
- CPU-Typ
- 32/64 Bit CPU (auch 16 oder 8 Bit)

Die Target-Plattform muss nicht mal real existieren!

Einsatzzwecke - Embedded Systeme



- Speicher/CPU-Ressourcen reichen für Compiler/Toolchain nicht aus
- Ein komfortable Entwicklungsumgebung läuft auf anderer Plattform
- Zeitersparnis: anderer Rechner hat mehr Ressourcen und kann schneller compilieren.

Einsatzzwecke

- Gewohnte Entwicklungsumgebung läuft nicht auf anderen Plattformen
- Build-Umgebung soll einheitlich sein
- Target-Plattform nicht verfügbar
- Softwarelizenz (Betriebssystem und/oder Entwicklungstools) nicht vorhanden
- (Noch) kein Compiler auf der Targetplattform verfügbar
- Kontinuierliche Integration – aktueller Softwarestand wird automatisch für diverse Plattformen compiliert, um Probleme früh zu erkennen.

Hello world unter Linux compilieren

hoffentlich bekannt...

Compilieren von helloworld.c

```
$ gcc -Wall -o helloworld helloworld.c
$ file helloword
helloworld: ELF 64-bit LSB executable, x86-64, [...]
```

Hello world unter Linux **für Windows** compilieren

Cross-Compiler ist installiert unter `/usr/bin/i686-w64-mingw32-gcc`.
Einfach den speziellen Compiler installieren und damit compilieren:

Compilieren von helloworld.c

```
$ apt-get install g++-mingw-w64-i686
[...]
$ i686-w64-mingw32-gcc -Wall -o helloworld.exe helloworld.c
$ file helloword.exe
helloworld.exe: PE32 executable (console) Intel 80386, for MS Windows
```

erzeugt ein Windows-Executable.

Programme unter Linux **für Windows 64 Bit** compilieren

Compilieren von helloworld.c

```
$ apt-get install g++-mingw-w64-x86-64
[...]
$ x86_64-w64-mingw32-gcc -Wall -o helloworld.exe helloworld.c
$ file helloword.exe
helloworld.exe: PE32+ executable (console) x86-64, for MS Windows
```

erzeugt ein 64 Bit Windows-Executable.

Umfangreichere Programme unter Linux **für Windows** compilieren

Die notwendigen Bibliotheken (Libraries) müssen auch für die Zielplattform verfügbar sein[1]. Ansonsten kann ganz normal compiliert werden.

Compilieren von helloworld-wx.cpp

```
$ i686-w64-mingw32-gcc -Wall -o helloworld-wx.exe helloworld-wx.cpp
     $(/usr/i686-w64-mingw32/bin/wx-config --libs --cxxflags)
$ file helloworld-wx.exe
helloworld-sdl.exe: PE32 executable for MS Windows (GUI)
Intel 80386 32-bit
```

erzeugt ein Windows-Executable.

[1] entweder selbst übersetzen oder fertig downloaden

Andere Programmiersprachen

Etliche (C, C++, ObjC, Fortran durch GCC unterstützt).

Beispiel Go

Targetangabe durch Umgebungsvariablen

```
go build helloworld.go
[...]
GOOS=windows go build helloworld.go
[...]
GOOS=linux GOARCH=arm go build helloworld.go
```

Motiviation: Windows-Installer für Maxima

- leistungsfähiges Open Source CAS
- programmiert (überwiegend) in Lisp
- Entwickler arbeiten oft nicht unter Windows
- Windows-Installer kommen nur sporadisch
- Erstellung sehr aufwendig

Installationsanweisungen für Windows-Installer

External Requirements (passt grad noch auf die Folie)

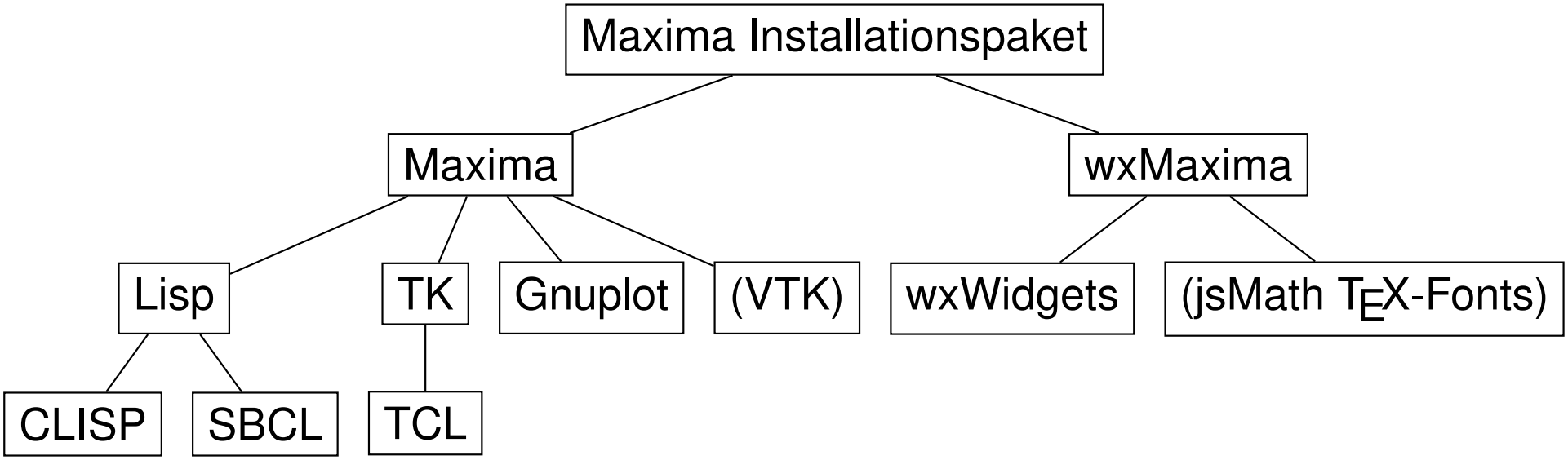
1. MSYS+MinGW, including mktemp
2. msysDTK
3. GCL
4. Tcl/Tk (8.5 or later recommended)
5. Starkit, TclKit and img.kit
6. Vtk
7. InnoSetup
8. gnuplot
9. wxMaxima
10. Microsoft HTML Help Workshop
11. Perl 5.8 (ActivePerl)
12. Python

Installationsanweisungen für Windows-Installer

Die Installation der Pakete unter Windows...

```
apt-get install MSYS MinGW msysDTK GCL tcl tk ...
```

Abhängigkeiten



Anforderungen

für einen automatisierten Maxima-Installerbau

- Skriptbar ((hoffentlich) keine manuellen Tätigkeiten)
  $\Rightarrow$ bevorzugt unter Linux/Unix
- Erweiterbar
- Automatische Downloads
- Einfache Updates
- Resultat: All-in-one Installer

Möglichkeiten

- Shellskript
- Perl / Python / Ruby / ... -skript
- Makefiles (vgl. `http://mxe.cc`)
- Higher Level Buildsystem: scons, CMake

Vereinfachungen

Build-System (z.B. Linux) $\neq$ Zielsystem (z.B. Windows)

- Quellen: Downloads, Versionsverwaltungssysteme (SVN, GIT, CVS, ...)
- Fertig compilierte Archive
- Gnu Autoconf (`./configure ; make ; make install`), ...
- Patches
- CMake als Buildsystem
- Spezialfälle

CMake - externalproject_add

Einbinden von externen Programmen: fertige Archive

Beispiel: gnuplot: fertiges ZIP-File

```
# Gnuplot (already a binary package just extract it...)
externalproject_add(gnuplot
  URL "${GNUPLOT_URL}"
  DOWNLOAD_DIR ${CMAKE_SOURCE_DIR}/downloads
  URL_MD5 ${GNUPLOT_MD5}
  CONFIGURE_COMMAND ""
  BUILD_COMMAND ""
  INSTALL_COMMAND ""
)
install(DIRECTORY ${CMAKE_BINARY_DIR}/gnuplot-prefix/src/gnuplot/
                  DESTINATION gnuplot)
```

CMake - `externalproject_add`

Einbinden von externen Programmen: `./configure ; make ; make install`

Beispiel: Compilieren von TCL

```
externalproject_add(tcl
  URL "${TCL_URL}"
  DOWNLOAD_DIR ${CMAKE_SOURCE_DIR}/downloads
  URL_MD5 ${TCL_MD5}
  CONFIGURE_COMMAND ${CMAKE_BINARY_DIR}/tcl-prefix/src/tcl/win/configure
                    --host=${HOST} --prefix=C:/maxima-${MAXIMAVERSION}
  BUILD_COMMAND $(MAKE)
  INSTALL_COMMAND $(MAKE) install
)
install(DIRECTORY
        ${CMAKE_BINARY_DIR}/tcl-prefix/src/tcl-build/C:/maxima-${MAXIMAVERSION}/
        DESTINATION .)
```

CMake - externalproject_add

Spezialfälle: Crosscompilieren von Lisp - Maxima

Verwendung von Wine

```
$ cat wine-clisp.sh.tmpl
#!/bin/sh
wine @CLISPROOT@/clisp.exe "$@"


set(CLISPROOT "${CMAKE_BINARY_DIR}/clisp-prefix/src/clisp")
configure_file("${CMAKE_SOURCE_DIR}/wine-clisp.sh.tmpl"
               "${CMAKE_BINARY_DIR}/wine-clisp.sh")
configure_file("${CMAKE_SOURCE_DIR}/wine-lisp.sh.tmpl"
               "${CMAKE_BINARY_DIR}/wine-lisp.sh")
```

CMake - externalproject_add

Spezialfälle: Crosscompilieren von Lisp - Maxima

Verwendung von Wine als Crosscompiler

```
externalproject_add(maxima
  # copy the Maxima source to a build directory
  DOWNLOAD_COMMAND rsync -av --exclude=crosscompile-windows/
                   "${CMAKE_SOURCE_DIR}/../"
                   "${CMAKE_SOURCE_DIR}/build/maxima-prefix/src/maxima"
  BUILD_IN_SOURCE 1
  CONFIGURE_COMMAND sh -c "test -x configure || ./bootstrap"
            COMMAND ./configure --host=${HOST} --enable-clisp
                        --with-clisp=${CMAKE_BINARY_DIR}/wine-clisp.sh
                        --with-clisp-runtime=${CMAKE_BINARY_DIR}/wine-lisp.sh
                        --prefix=C:/maxima-${MAXIMAVERSION}
  BUILD_COMMAND $(MAKE)
  INSTALL_COMMAND $(MAKE) -C ${CMAKE_BINARY_DIR}/maxima-prefix/src/maxima install
                          DESTDIR=${CMAKE_BINARY_DIR}/maxima-installroot/
)
```

Paketieren mit CPack

RPM, DEB Pakete, Windows Installer (NSIS)

Diverse allgemeine Variablen setzen

```
set(CPACK_GENERATOR "NSIS")
set(CPACK_PACKAGE_VERSION "${MAXIMAVERSION}")
set(CPACK_PACKAGE_DESCRIPTION_SUMMARY "Maxima")
set(CPACK_PACKAGE_VENDOR "Maxima Team")
set(CPACK_PACKAGE_DESCRIPTION_FILE "${CMAKE_SOURCE_DIR}/../README")
set(CPACK_RESOURCE_FILE_LICENSE "${CMAKE_SOURCE_DIR}/../COPYING")
set(CPACK_PACKAGE_INSTALL_DIRECTORY "maxima-${CPACK_PACKAGE_VERSION}")
set(CPACK_PACKAGE_FILE_NAME "maxima-clisp-sbcl-${CPACK_PACKAGE_VERSION}")
set(CPACK_PACKAGE_EXECUTABLES "..\\\\wxMaxima\\\\wxmaxima" "wxMaxima (GUI)")
```

Paketieren mit CPack

RPM, DEB Pakete, Windows Installer (NSIS)

NSIS spezifische Variablen setzen

```
set(CPACK_NSIS_INSTALL_ROOT "C:")
set(CPACK_NSIS_ENABLE_UNINSTALL_BEFORE_INSTALL ON)
set(CPACK_PACKAGE_ICON "${CMAKE_SOURCE_DIR}/maxima-icon.bmp")
set(CPACK_NSIS_MUI_ICON "${CMAKE_SOURCE_DIR}/../interfaces\\\\maxima-icon.ico")
set(CPACK_NSIS_MUI_UNIICON "${CPACK_NSIS_MUI_ICON}")
set(CPACK_NSIS_URL_INFO_ABOUT "http://maxima.sourceforge.net")
set(CPACK_NSIS_MENU_LINKS "bin/maxima.bat" "Maxima (command line)"
                          "http://maxima.sourceforge.net" "About Maxima"
                          "share/doc/maxima.pdf" "Maxima documentation")
```

Paketieren mit CPack

Registry-Einträge für File associations

```
# File associations:
set(CPACK_NSIS_DEFINES "!include ${CMAKE_SOURCE_DIR}\\\\FileAssociation.nsh")
set(CPACK_NSIS_EXTRA_INSTALL_COMMANDS "
\\\${registerExtension} \\\"\\\$INSTDIR\\\\wxMaxima\\\\wxmaxima.exe\\\"
                        \\\".wxm\\\" \\\"wxMaxima Document\\\"
WriteRegStr HKCR \\\".wxmx\\\\ShellNew\\\" \\\"NullFile\\\" \\\"\\\"
")
set(CPACK_NSIS_EXTRA_UNINSTALL_COMMANDS "
\\\${unregisterExtension} \\\".wxm\\\" \\\"wxMaxima Document\\\"
DeleteRegKey HKCR \\\".wxmx\\\\ShellNew\\\"
")
```

Compilieren des Installers

4 Befehle für einen Windows-Installer

```
cd crosscompile-windows/build/
cmake ..
make # könnte wegelassen werden, ev. -j nn
make package
```

Extrahieren existierender Windows-Installer

Extrahieren von Microsoft-Formaten

7z extrahiert MSI, EXE Installer

```
Auflisten des Inhalts:
$ 7z l installer.msi # oder exe

Extrahieren:
$ 7z x installer.msi # oder exe
```

cabextract extrahiert CAB-Archive

```
Extrahieren:
$ cabextract file.cab # --list file.cab zum Auflisten
```

Vielen Dank

Fragen? (hoffentlich richtige. . .) Antworten!

Vielen Dank für Ihre Aufmerksamkeit

Wolfgang Dautermann
wolfgang.dautermann [AT] fh-joanneum.at

Werbeeinschaltung :-)