Updating the Compiler?

Take Advantage of The New Development Toolchain

Andreas Jaeger
Product Manager
aj@suse.com

Programming Languages

C

C++

Fortran

And Go

Why new compiler?

Faster applications

- Support for new CPUs
- New compiler optimizations
- Improved compiler optimizations

Take advantage of new language features

Write better code due to improved diagnostics

Easier to spot problems due to improved debugging

Toolchain Module for SUSE Linux Enterprise 12

Yearly release

Deliver new Compiler and toolchain

- GCC (“GNU Compiler Collection) development tools via Toolchain Module
- GCC runtime libraries, binutils, gdb as updates for SLE Core

Versions for 2016 update of Toolchain Module for SUSE Linux Enterprise 12:

- GCC 6.2 with C, C++, Fortran support
- Binutils 2.26.1
- Gdb 7.11.1

Package build compiler (GCC 4.8) stays as default

New module is supported (latest version of SW, overlap of 6 months)

Package Hub - https://packagehub.suse.com

Contains community build software for SUSE Linux Enterprise

Including:

- Google GO compiler and runtime
- Erlang
- Glasgow Haskell Compiler ghc
- R project for statistical computing

Compare with SUSE Linux Enterprise 11

Update of toolchain with each Service Pack

Deliver GCC new development tool via SDK

Deliver GCC runtime libraries, binutils, gdb as part of SLE core

No support for newer GCC

How to use new tools?

Binutils, gdb: No change

GCC: Use “XXX-6” instead of “XXX”, for example:

```
gcc-6
g++-6
```

Note: GCC versioning scheme

Previously (until GCC 4.9):

- Major release every year: 4.0, 4.1,...
- First release: 4.8.0
- Minor releases: 4.8.1, 4.8.2

Now:

- Major release every year: 5, 6, 7
- First release: 5.1
- Minor release: 5.2, 5.3

Advantages of new toolchain

Why new compiler?

Compiler optimizations

Support for CPUs

New language features

Diagnostics and debugging

Diagnostics and debugging
Better display of wrong code

```
test.cc: In function 'int test(int, int, foo, int, int)':
test.cc:5:16: error: no match for 'operator*' (operand types are 'int' and 'foo')
   return p + q * r * s + t;
              ~~^~~
```

```
format-strings.c:3:14: warning: field width specifier '*' expects a matching 'int' argument [-Wformat=]
   printf("%*d");
            ^
```

Diagnostics and debugging
Fix-it hints

```
fixits.c: In function 'bad_deref':
fixits.c:11:13: error: 'ptr' is a pointer; did you mean to use '->'?
   return ptr.x;
             ^
             ->
```

Diagnostics and debugging
Suggestions for wrong field names

```
spellcheck-fields.cc:52:13: error: 'struct s' has no member named 'colour'; did you mean 'color'?
   return ptr->colour;
               ^~~~~~
```

Diagnostics and debugging
Check indentation

-Wmisleading-indentation (in -Wall) would have avoided CVE-2014-1266:

```
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    /* more checking here.  */
  fail:
    /* cleanups */
    return err;
```

```
sslKeyExchange.c: In function 'SSLVerifySignedServerKeyExchange':
sslKeyExchange.c:629:3: warning: this 'if' clause does not guard... [-Wmisleading-indentation]
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    ^~
sslKeyExchange.c:631:5: note: ...this statement, but the latter is misleadingly indented as if it is guarded by the 'if'
        goto fail;
        ^~~~
```

Diagnostics and debugging

New warnings

-fsanitze=thread

UndefinedBehaviorSanitizer (-fsanitize=undefined) enhanced:

- **-fsanitize=bounds-strict: Strict checking of array bounds**

New warning options:

- **-Wshift-negative-value: Left shifting a negative value.**
- **-Wshift-overflow: Left shift overflows (default).**
- **-Wtautological-compare: Self-comparison always evaluates to true or false (in -Wall).**
- **-Wnull-dereference: Compiler detects paths that trigger erroneous or undefined behavior due to dereferencing a null pointer.**
- **-Wduplicated-cond: Duplicated conditions in an if-else-if chain.**

Compiler optimizations in GCC 6

Value Range propagation Cave-at!

Value Range propagation: This pointer of C++ member function is considered non-null

This breaks non-conforming code like Chromium, KDevelop, Qt-5.

Use -fno-delete-null-pointer-checks

To identify wrong code: -fsanitize=undefined

Compiler optimizations in GCC 6

Link Time Optimization

- Robustness fixes
- C and Fortran compatibility
- Smaller LTO objects (11 %)

Interprocedural Optimization

- Function cloning more aggressive
- Improved heuristics

Thread improvements:

- OpenMP 4.5
- OpenACC 2.0a

Support for newer CPUs

X86-64:

- AVX-512 extensions for Skylake
- AMD family 17h processors (Zen core)
- Segment register

AArch64:

- LTO support
- Processor support: ARM Cortex-A35, ARM Cortex-A53, ARM Cortex-A57, Qualcomm QDF24xx, Samsung Exynos M1

Power:

- Power9 support
- Vector support

z Systems

- IBM z13 support

New language features

GNU C 11 is default (ISO C 11 plus GNU extensions)[Since GCC 5]

GNU C++ 14 is default (ISO C++ 14 plus GNU extensions)

New libstdc++

Libstdc++ contains Dual ABI:

- New ABI is C++11 conforming
- Old ABI for compatibility with previous compilers
- C++11 conforming std::list with O(1) size() function
- std::string uses small string optimization instead of copy-on-write reference counting
- SUSE Linux Enterprise Server 12 uses Old ABI by default, since system libraries are compiled using the Old ABI

Optimization Tips

To optimize for current machine: -march=native

To optimize for pool of machines: Use -march=X and select lowest common dominator or don't set it at all

Optimization flags:

-O2

-O3

-Ofast: includes -O3, -ffast-math, “unsafe” optimizations

Speed optimizations:

1) Use FDO
2) Use FDO+LTO

Feedback Driven Optimization

Use information about common paths from run-time to optimize the common case, including value profiling

Usage:

Build binary for profiling:
$ gcc -Ofast program.c -o program -fprofile-generate

Execute binary and create profile data

Recompile binary using profile:
$ gcc -Ofast program.c -o program -fprofile-use=program.gcda

Note: Profiling collection not thread-safe

Link Time Optimization

Whole-program optimization instead of single source file optimization

Files get “compiled” into .o: streaming presentation of internal data

Optimization happens at link time for the whole program

Allows interprocedural analysis across files

Link Time Optimization: Usage

Build binaries using:

```
$ gcc -Ofast -flto -c file1.c
$ gcc -Ofast -flto -c file2.c
```

Link using:

```
$ gcc -flto -o program file1.o file2.o
```

Vectorized Math Routines

Vectorizer now can use glibc’s math vector routines for vector sine etc.

Note: Needs Glibc from SLE 12 SP2, supported on x86-64 only

References

https://gcc.gnu.org/gcc-6/changes.html

https://gcc.gnu.org/gcc-6/porting.html

Outlook GCC 7

Note: GCC 7 will come out in Q2 2017 and is under development

Further fix-its

LTO debugging

Experimental C++17 support

Why new toolchain?

GDB:

- Understand new debug information by compiler

Binutils:

- Support for new toolchain
- Support for new CPUs

GDB 7.11.1 and Binutils 2.26

GDB:

- Thread signal handling improved, user gets informed about which thread triggers event
- More explicit syntax for breakpoints (“break -s main.c -li 3”)
- Bug fixes

Binutils:

- Support for compressed DWARF debug format, currently gas, ld
- IBM z13 support
- Bug fixes

Distributing Self-Compiled Software

Software build with new Toolchain Module will run on updated SLE 12 systems

Need updates for runtime libraries installed

Support

Support via the usual channel

Our GCC team needs self-contained code to reproduce issues

Concentrate on -g/-g0, -O[0123], -ffast-math, -Ofast

The Go Programming Language

Go

“Go is an open source programming language that makes it easy to build simple, reliable, and efficient software. “

Created 2007 at Google by Robert Griesemer, Rob Pike, and Ken Thompson.

Designed for writing servers.

Compiled, statically typed, with garbage collection, memory safety, concurrent features.

Why developers use Go?

Docker and docker eco-system uses Go as main language.

Faster than shell scripts

More powerful than C

Shines:

- **Netowork and Webserver**
- **Stand-alone command line programs and scripts**

Not so good for:

- **Desktop or GUI based apps**
- **System level programming**

Where does SUSE use Go?

Development of Docker

Machinery

Go 1.7 Release

Support now for all our architectures – AARCH64, PowerPC64, x86-64, and z Systems. (New: z Systems)

New compiler back end.

PackageHub

Go packages are available on the PackageHub

Compiler package is named “go”

Add PackageHub to your system via YaST

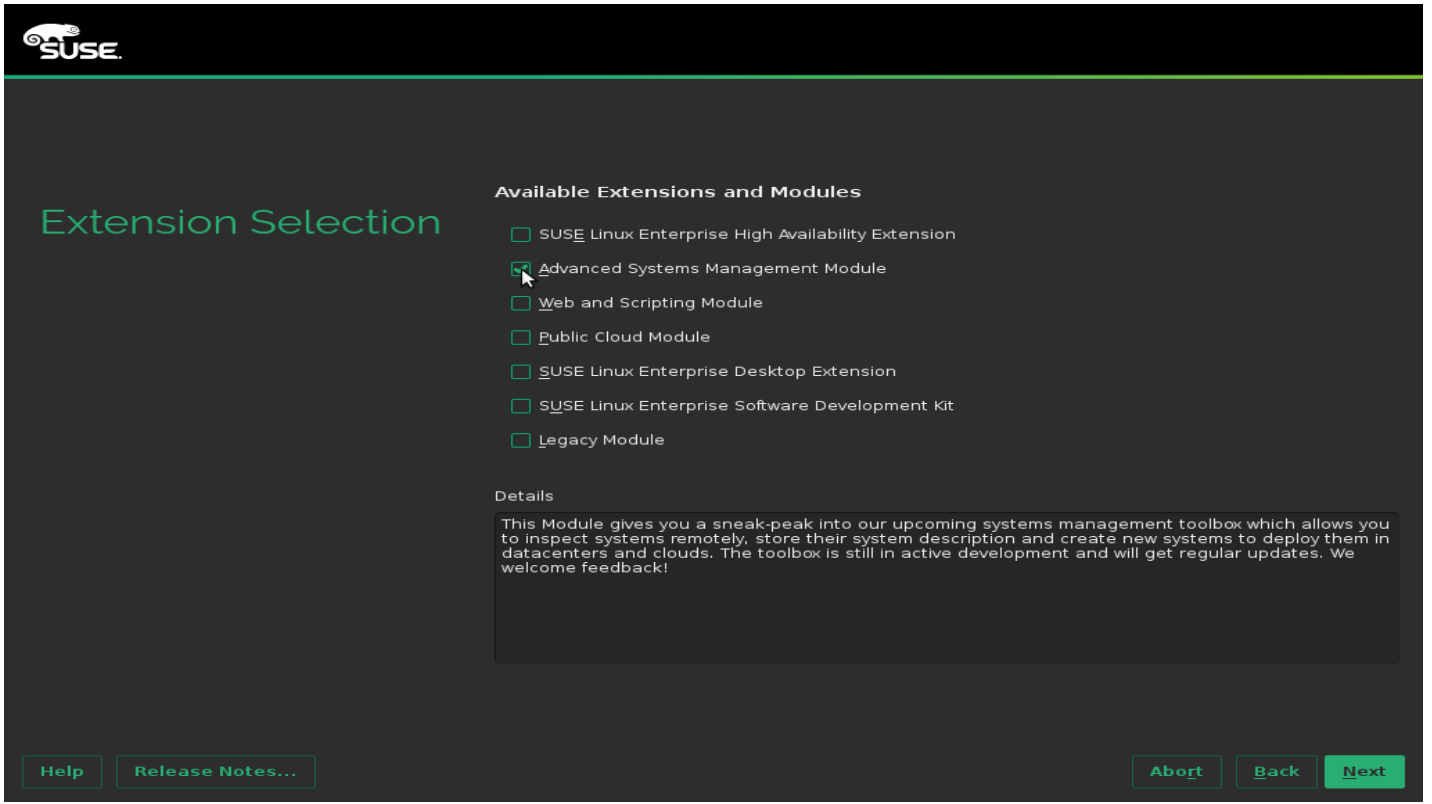
Also, available for openSUSE in devel:languages:go project

Installation

Modules - Overview

Module Name	Content (examples)	Lifecycle
Advanced Systems Management Module	The configuration management tools cfengine, puppet, salt and the new "machinery" tool	Continuous Integration
Container Module	Docker and container related functionality such as ECS integration	Continuous Integration
Legacy Module	Sendmail, old IMAP stack, old Java etc.	3 years
Public Cloud Module	Instance initialization code, command line tools for management	Continuous Integration
Toolchain Module	**GCC**	**Yearly delivery**
Web and Scripting Module	“PHP”, “Python”	3 years, 18 months overlap

Module – Software Install




SUSE®
We adapt. You succeed.