Updating the Compiler?

Take Advantage of The New Development Toolchain

Andreas Jaeger
Product Manager
aj@suse.com

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 2015 update of Toolchain Module for SUSE Linux Enterprise 12:
  - GCC 5.2 with C, C++, Fortran support
  - Binutils 2.25
  - Gdb 7.9
- Package build compiler (GCC 4.8) stays as default
- New module is supported (latest version of SW)

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-5” instead of “XXX”, for example:
  -
  -

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

Compiler optimizations in GCC 5

- Link Time Optimization
- Interprocedural Optimization
- Thread improvements:
  - OpenMP 4.0
  - Cilk Plus
  - OpenACC 2.0
-

Support for newer CPUs

- X86-64:
  - AVX-512 support
  - Intel MPX
  - Intel Broadwell and Silvermont microarchitectures
  - AMD family 15h processors (Excavator core)
- Power:
  - Power8 support

New language features

- GNU C 11 is default (ISO C 11 plus GNU extensions)
- C++ 11 finished
- C++ 14 as experimental
- Cilk Plus support

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

Diagnostics and debugging

Runtime checks using sanitizer

- -fsanitize=address: Memory error detector like out-of-bounds and use-after-free bugs.
- -fsanitze=thread
- UndefinedBehaviorSanitizer (-fsanitize=undefined) enhanced:
  - shift
  - integer-divide-by-zero
  - bound

Diagnostics and debugging
Colorizing of messages

```
$ g++ -fdiagnostics-color=always -S -Wall test.C
test.C: In function ‘int foo()’:
test.C:1:14: warning: no return statement in function returning non-void [-Wreturn-type]
 int foo () { }
              ^
test.C:2:46: error: template instantiation depth exceeds maximum of 900 (use -ftemplate-depth= to increase the maximum) instantiating ‘struct X<100>’
 template <int N> struct X { static const int value = X<N-1>::value; }; template struct X<1000>;
                                              ^
test.C:2:46:   recursively required from ‘const int X<999>::value’
test.C:2:46:   required from ‘const int X<1000>::value’
test.C:2:88:   required from here

test.C:2:46: error: incomplete type ‘X<100>’ used in nested name specifier
```

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 FTO

  2) Use FTO+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

Link Time Optimization: New in GCC 5

- Slim object files
- One definition rule: Aliasing and devirtualization optimizations
  - -Wodr – warn about violation of one definition rule
- Compile time options will be preserved for link time
  - function granularity
  - Allows special compiler flags for individual files

Interprocedural Optimizations

- Work best on LTO
- New identical code folding pass
- Improved devirtualization pass
- New comdat localization pass: allows the linker to eliminate more dead code in presence of C++ inline functions.
- Optimized virtual tables
- Elimination of write-only variables

Outlook GCC 6

- Note: GCC 6 will come out in Q2 2016 and is under development
- C++14 as default;
- First C++17 experimental features
- GPU offloading “usable”
- LTO debugging

Why new toolchain?

- GDB:
  - Understand new debug information by compiler
- Binutils:
  - Support for new toolchain
  - Support for new CPUs

GDB 7.9 and Binutils 2.25

- GDB 7.9:
  - Improved python scripting API
  - Thread signal handling improved
  - Bug fixes
- Binutils:
  - Minor new commands like “strings –data” or “strings –include-all-whitespace”
  - 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

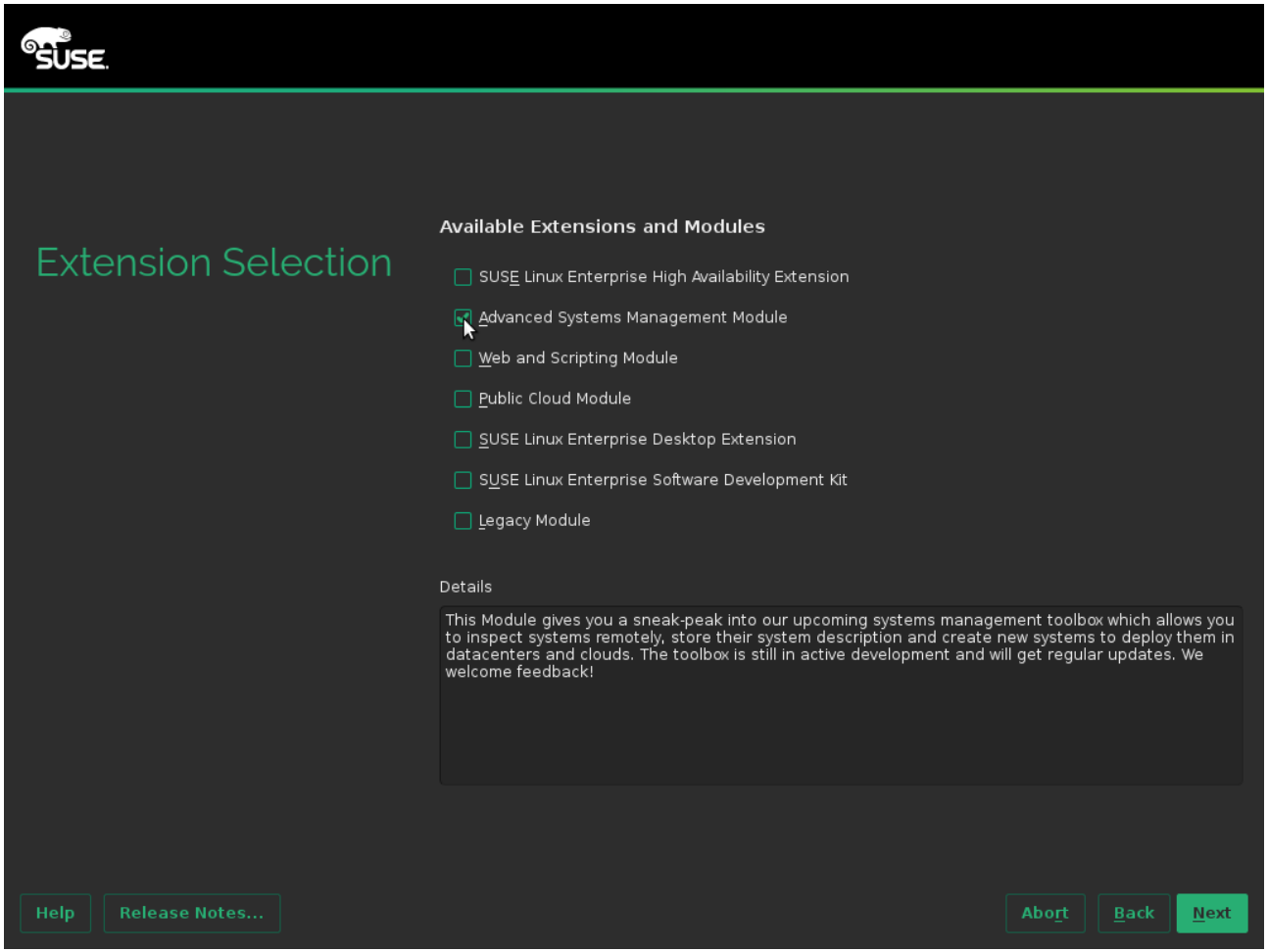
- Support via the usual channel
- Our GCC team needs self-contained code to reproduce issues
- Concentrate on -g/-g0, -O[0123], -ffast-math, -Ofast

Installation

Modules - Overview

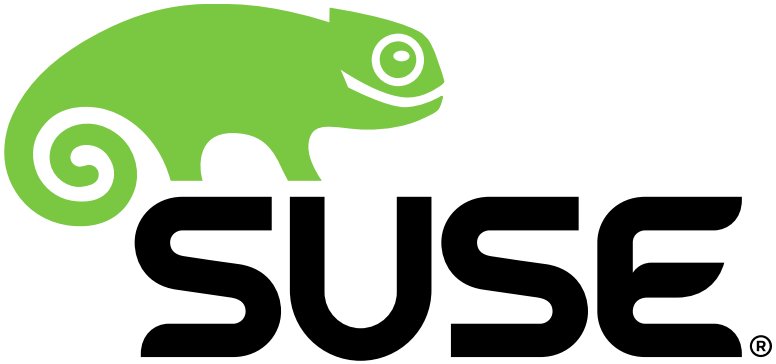
Module Name	Content (examples)	Lifecycle
Advanced Systems Management Module	The configuration management tools cfengine, puppet, salt and the new "machinery" tool	Continuous Integration
Container Module	Docker and container related functionality such as ECS integration	Continuous Integration
Legacy Module	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

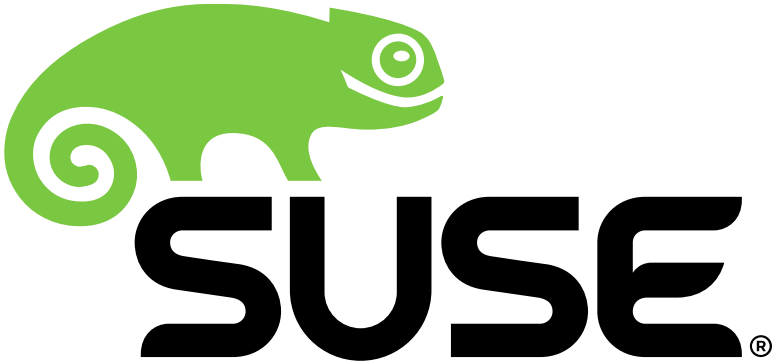
Module – Software Install



How will you use our new toolchain?

Thank you.


SUSE®
We adapt. You succeed.



Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org