Make Debian compiler agnostic
I CAN HAS PROGRAMMING LANGUAGE?
Current status:
All C, C++, Objective-C sources are being built with GCC for all supported Debian arches.
Gcc is THE FLOSS compiler for the last 25 years
Used for (pretty much) everywhere or anything
Why a new compiler ?
Because we can
No other reason ?
Because it is fun
Seriously:
Other compilers can find programming errors that gcc could not find
Code built by many compilers is more likely to be more strictly correct and more portable than code only built with gcc.
Some compilers can have advantages on some archs (ex: clang on ARM)
As we were able to do with decoupling Linux from Debian with kFreeBSD and the HURD, we're aiming to decouple GCC in Debian.
Make Debian compiler agnostic
Sylvestre Ledru
Started as an academic project
Versatile platform for compilation and virtual machine

Designed originally for the investigation of dynamic compilation techniques for static and dynamic languages
Sponsored by Apple since 2005 to replace GCC (GPL vs BSD)

Has now a strong and diverse community (academics, individuals and corporates)

Many universities/research centers are basing their research on LLVM
Clang

C, C++ & Objective-C compiler.
(no Fortran)
Based on LLVM

Default compiler for Mac OS X (Xcode)/iOS [1] and FreeBSD [2]

Sources:
Some advantages:

More recent base code (ie less legacy code)

Strong interest of material manufacturers (ARM, MIPS, Nvidia, etc)

Supposed to be faster to build code than gcc

Accept the same arguments as gcc
Example

Full build of Scilab (doc, essential tests)
~24 minutes gcc
~22 minutes clang
Some advantages (bis)
More intelligent detections

```c
int main() {
    unsigned int i = 0;
    return i < 0;
}
```

$ gcc -Wall -Werror foo.c ; echo $?
0

$ clang -Werror foo.c
foo.c:3:17: error: comparison of unsigned expression < 0 is always false
[-Werror,-Wtautological-compare]

return i < 0;
~ ^ ~

1 error generated.
Side effect

=> Brings (friendly) competition in the free compiler world.

I IS TEN NINJAS
Comparison of Diagnostics between GCC and Clang

It is often repeated that the Clang compiler produces far superior diagnostics to GCC. For example the Expressive Di indeed superior to GCC 4.2. However, that version of GCC is a few years old, and GCC has improved considerably since and add further interesting examples.¹

http://gcc.gnu.org/wiki/ClangDiagnosticsComparison
Clang vs GCC (GNU Compiler Collection)

Pro's of GCC vs clang:

- GCC supports languages that clang does not aim to, such as Java, Ada, FORTRAN, etc.
- GCC supports more targets than LLVM.
- GCC is popular and widely adopted.
- GCC does not require a C++ compiler to build it.

http://clang.llvm.org/comparison.html#gcc
Rebuild of Debian using Clang
Crappy method :

VERSION=4.7
cd /usr/bin
rm g++-$VERSION gcc-$VERSION cpp-$VERSION
ln -s clang++ g++-$VERSION
ln -s clang gcc-$VERSION
ln -s clang gcc-$VERSION
ln -s clang cpp-$VERSION
cd -
Testing the rebuild of the package.

NOT the performances (build time or execution) nor the execution of the binaries.
Rebuild with clang 3.0
February 28, 2012

15658 packages built : 1381 (8.8 %) failed.
Rebuild with clang 3.1
June 23, 2012

17710 packages built : 2137 (12.1 %) failed.
Full results published: http://clang.debian.net/

Debian Package rebuild
Rebuild of the Debian archive with clang

By Sylvestre Ledru (Debian, IRILL, Scilab Enterprises). February 28th 2012

Presentation

This document presents the result of the rebuild of the Debian archive (the compiler.

clang is now ready to build software for production (either for C, C++ or Ob
wasm-ware) and interesting more than the use-case while not servin

Done on the cloud-qa - EC2 (Amazon cloud)
Thanks to Lucas Nussbaum
Why these differences between 3.0 & 3.1?
-Werror & unused args
96 occurrences

Clang detects unused argument.

clang --param ssp-buffer-size=4 -Werror foo.c

And generates a normal warning ...

Which becomes an error with -Werror

clang: error: argument unused during compilation: '--param ssp-buffer-size=4'
96 occurrences

Fixed in clang 3.2 rc1 :
http://llvm.org/bugs/show_bug.cgi?id=9673
Security check introduced in clang 3.1
20 occurences

```c
#include <stdio.h>
void foo(void) {
    char buffer[1024];
    sprintf(buffer, "%n", 2);
}
```

$ gcc -Werror -c foo.c && echo $? 0
$ clang -Werror -c foo.c && echo $? foo.c:5:23: error: use of '%n' in format string discouraged (potentially insecure) [-Werror,-Wformat-security]
    sprintf(buffer, "%n", 2);
    ^
1 error generated.

Make Debian compiler agnostic
Sylvestre Ledru
Some of the most common errors
Unsupported options
48 occurrences

$ gcc -O9 foo.c && echo $?  
0

$ clang -O9 foo.c

**error**: invalid value '9' in '-O9'
Different default behavior
132 occurrences

```c
int foo(void) {
  return;
}
```

$ gcc -c noreturn.c; echo $?
0
# -Wall shows it as warning

$ clang -c noreturn.c
`noreturn.c:2:2: error: non-void function 'foo' should return a value [-Wreturn-type]`
    return;
    ^
1 error generated.
Different default behavior (bis)
17 occurrences

```c
void foo(void) {
    return 42;
}
```

$ gcc -c returninvoid.c; echo $?  
returninvoid.c: In function ‘foo’:  
returninvoid.c:2:2: warning: ‘return’ with a value, in function returning void [enabled by default]  
0  

$ clang -c returninvoid.c  
returninvoid.c:2:2: error: void function 'foo' should not return a value [-Wreturn-type]  
    return 42;
    ^~~  
1 error generated.
Different understanding of the C++ standard

```cpp
class address {
    protected:
        static int parseNext(int a);
};

class mailbox : public address {
    friend class mailboxField;
};

class mailboxField {
    void parse(int a) {
        address::parseNext(a);
        // will work with:
        // mailbox::parseNext(a);
    }
};
```

$ g++ -c mailboxField.cpp && echo $?  
0

$ clang++ -c mailboxField.cpp
mailboxField.cpp:17:22: error: 'parseNext' is a protected member of 'address'
    address::parseNext(a);
    ^
mailboxField.cpp:4:16: note: declared protected here
        static int parseNext(int a);
        ^
```

References:
http://llvm.org/bugs/show_bug.cgi?id=6840
http://gcc.gnu.org/bugzilla/show_bug.cgi?id=52136

November, 23th 2012

Make Debian compiler agnostic
Sylvestre Ledru
Different set of warnings with -Wall
Plenty of occurrences

```c
- plop.c -

void foo() {
    int a=1;
    if ((a == 1)) {
        return;
    }
}
```

$ gcc -Wall -Werror -c foo.cpp && echo $?
0

$ clang -Wall -Werror -c foo.cpp

```
foo.cpp:3:13: error: equality comparison with extraneous parentheses
    if ((a == 1)) {
          ^~~
foo.cpp:3:13: note: remove extraneous parentheses around the comparison to silence this warning
    if ((a == 1)) {
        ^   ~
foo.cpp:3:13: note: use '=' to turn this equality comparison into an assignment
    if ((a == 1)) {
        ~
    =
```

1 error generated.
GCC Extensions which won't be supported
25 occurrences

```
#include <vector>

void foo() {
    int N=2;
    std::vector<int> best[2][N];
}
```

```
$ g++ -c foo.cpp; echo $?  
0

$ clang++ -c foo.cpp

foo.cpp:4:29: error: variable length array of non-POD element type
'std::vector<int>'
    std::vector<int> best[2][N];

1 error generated.
```
GCC accepts stuff which should not 34 occurences

```c
-foo.cpp-

// Uncomment this line will fix the issue.

// template<typename Value_t>
// void b(Value_t value)

template<typename Value_t>
void a(Value_t value) {
    b(value);
}

template<typename Value_t>
void b(Value_t value) {
}

void foo(int y) {
    a(y);
}
```

$ g++ -c foo.cpp; echo $?  
0

$ clang++ -c foo.cpp

```
foo.cpp:6:5: error: call to function 'b' that is neither visible in the template
    b(value);
^
foo.cpp:15:5: note: in instantiation of function template specialization
    a<int>' requested here
    a(y);
^
foo.cpp:9:33: note: 'b' should be declared prior to the call site
```
GSoC 2012 work
Objective:
Update the Debian infrastructure to allow a change of compiler

Student: Alexander Pashaliyski
Mentors: Paul Tagliamonte & me
First output:

A tutorial/documentation for wanna-build setup

http://wiki.debian.org/DebianWannaBuildInfrastructureOnOneServer

http://wiki.debian.org/SetupBuildServiceForWanna-build
Setup a parallel infrastructure to the Debian build system

wanna-build

Buildd #1 patched → clang

Buildd #2 patched → clang

Buildd #3 patched → clang
Hack the Debian tools to:

- Force dpkg to export CC=/usr/bin/cc, CXX=/usr/bin/c++ and OBJC=/usr/bin/objc

- Check for hardcoded CC=gcc in debian/rules

- Set the /usr/bin/{cc,c++,objc} alternatives to {clang, clang++}
• Call a hook to sbuild after the apt-get install of the build dependencies

• Fail the build on purpose when direct usage of gcc, g++ or cpp

Published on:
• https://github.com/sylvestre/debian-clang/
Results
Publication of the build results of the packages using clang

Connected on the debian mirror (ie : updated packages)
## Debian Clang Package Auto-Building

### Builddd status for packages maintained by sylvestre@debian.org

**DDPO (sylvestre@debian.org) - Bugs**

Package(s): arpack, atlas, blas, clang, code-s

Filter by status: good (35) bad (10)

<table>
<thead>
<tr>
<th>Package</th>
<th>amd64</th>
<th>i386</th>
</tr>
</thead>
<tbody>
<tr>
<td>arpack</td>
<td>Built</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>atlas</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>blas</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>clang</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>code-saturne</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>dragnonegg</td>
<td>Built</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>fwbuilder</td>
<td>Build-Attempted</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>gl2ps</td>
<td>Built</td>
<td>Built</td>
</tr>
<tr>
<td>gluegen2</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>gtkmathview</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>guake</td>
<td>Built</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>hdf5</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>jhdf</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>lapack</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>libcgnns</td>
<td>Built</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>libjogl-java</td>
<td>Build-Attempted</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>libjogl2-java</td>
<td>Build-Attempted</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>libmatio</td>
<td>Built</td>
<td>Needs-Build</td>
</tr>
<tr>
<td>llvm-2.9</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>llvm-3.0</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
<tr>
<td>llvm-3.1</td>
<td>Build-Attempted</td>
<td>Build-Attempted</td>
</tr>
</tbody>
</table>

*November, 23th 2012*  

**Make Debian compiler agnostic**  

Sylvestre Ledru
Next steps
Update the debian policy to include something like:

*Hardcoded usage of CC or CXX (for example, CC=gcc) should be avoided and documented if necessary.*

*Debian build tools must respect the CC and CXX variables if provided. If not, they shall default to /usr/bin/cc and /usr/bin/c++*

See:

http://lists.debian.org/debian-devel/2012/08/msg00783.html
Add a lintian warning like

W: yourpackage: Hardcoded call to gcc/g++. Use /usr/bin/cc or /usr/bin/c++ instead
Should be available as a new item in the PTS
Create a repository of packages built with Clang
Future
Potential the rebuild of Debian with:

- clang+plugin. Ex: polly: cache-locality optimisation auto-parallelism and vectorization, etc

- address sanitizer (ASAN)

- scan-build: static C/C++ analyzer

```c
if (! s) return NULL;
```

```bash
7 | Taking true branch
```

```c
8 | Within the expansion of the macro 'NULL':
6 | Memory is never released; potential leak of memory pointed to by 's'
root = (ezxml_root_t)ezxml_parse_str(s, len);
```

- Intel compilers
Another GSoC project
Student : Andrej Belym
Mentor : Me
Packaging of

libc ++

libc++ is a new implementation of the C++ standard library, targeting C++0X.

libc++abi

libc++abi is a new implementation of low level support for a standard C++ library.
Clang++ is linking against libstdc++

Example:

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << " plop" << endl;
}
```

```
$ clang++ -o plop main.cpp
$ ldd plop|grep stdc
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6
(0x00007f4b50817000)
```
But Clang++ can link and run with libc++

Example:

```
main.cpp
#include <iostream>
using namespace std;
int main(){
    cout << " plop" << endl;
}
```

```
$ clang++ -stdlib=libc++ -o plop main.cpp
$ ldd plop|grep libc++
libc++.so.1 => /usr/lib/libc++.so.1 (0x00007ff0eaf1d000)
```

---

November, 23th 2012

Make Debian compiler agnostic
Sylvestre Ledru
Initial upload in Debian in July
(new snapshot upload upload yesterday ;)

No official stable release yet
Packaging of compiler-rt

A C runtime library (equivalent to libgcc_s.so)
Any questions? Remarks?
Troll? (+1)