An update on Debian with Clang

Sylvestre Ledru – sylvestre@debian.org

Current status :
All C, C++, Objective-C sources are being built with gcc for all supported Debian arches and Kernel.

gcc is THE FLOSS compiler for the last 25 years
Used for (pretty much) everywhere or anything

Why a new compiler in Debian ?

debian

- 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 then 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 are aiming to decouple gcc in Debian.

debian

LLVM/Clang

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

LLVM also provide a code representation called IR (Intermediate Representation)

See the next talk of Philipp Kaluza:
Introducing Architecture: llvm

Clang

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

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

Sources:
[1] https://developer.apple.com/technologies/tools/
[2] http://lists.freebsd.org/pipermail/freebsd-stable/2012-May/067486.html

debian

Some advantages :

More recent base code (ie less legacy code)

Strong interest of hardware manufactors (ARM, MIPS, Intel, Nvidia, etc)

Supposed to be faster to build code than gcc

Accept the same arguments as gcc

About performances :

Results presented by Google last April at the Euro LLVM conference

clang compared to gcc

Servers: 1.04
Image processing: 1.03
Video codecs: 1.00

Agregate of various internal benchmarks of Google

Source: http://www.irill.org/videos/euro-llvm-2013/carruth-hires

Libraries

OpenSSL: 1.00
Protocol Buffer: 1.12
Snappy : 1.05

Source: http://www.irill.org/videos/euro-llvm-2013/carruth-hires

About portability :

Now support of AArch64 (ARM64), R600, S390 and S390X

Improvement of MIPS/powerpc

Source: http://www.irill.org/videos/euro-llvm-2013/carruth-hires

Some advantages (bis)
More intelligent detections

– foo.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.
```

Rebuild of Debian using Clang

Crappy method :

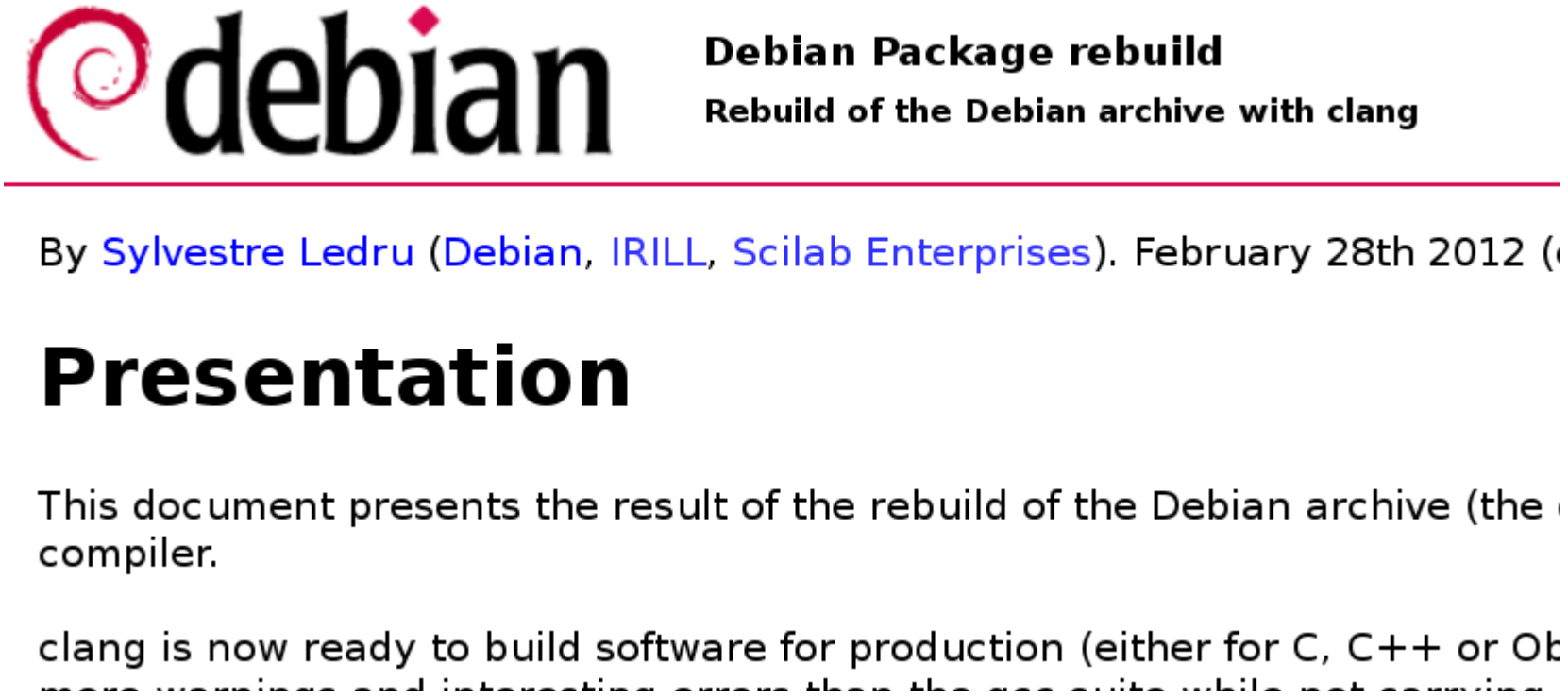
```
VERSIONS="4.8 4.7 4.6"
cd /usr/bin
for VERSION in $VERSIONS; do
 rm g++-$VERSION gcc-$VERSION cpp-$VERSION
 ln -s clang++ g++-$VERSION
 ln -s clang gcc-$VERSION
 ln -s clang cpp-$VERSION
done
```

CC=clang CXX=clang++ dpkg-buildpackage
fails to use clang in too many cases

Testing the rebuild of the package under amd64.

NOT the performances (build time or execution)
nor the execution of the binaries

Full results published:
http://clang.debian.net/



Debian Package rebuild

Rebuild of the Debian archive with clang

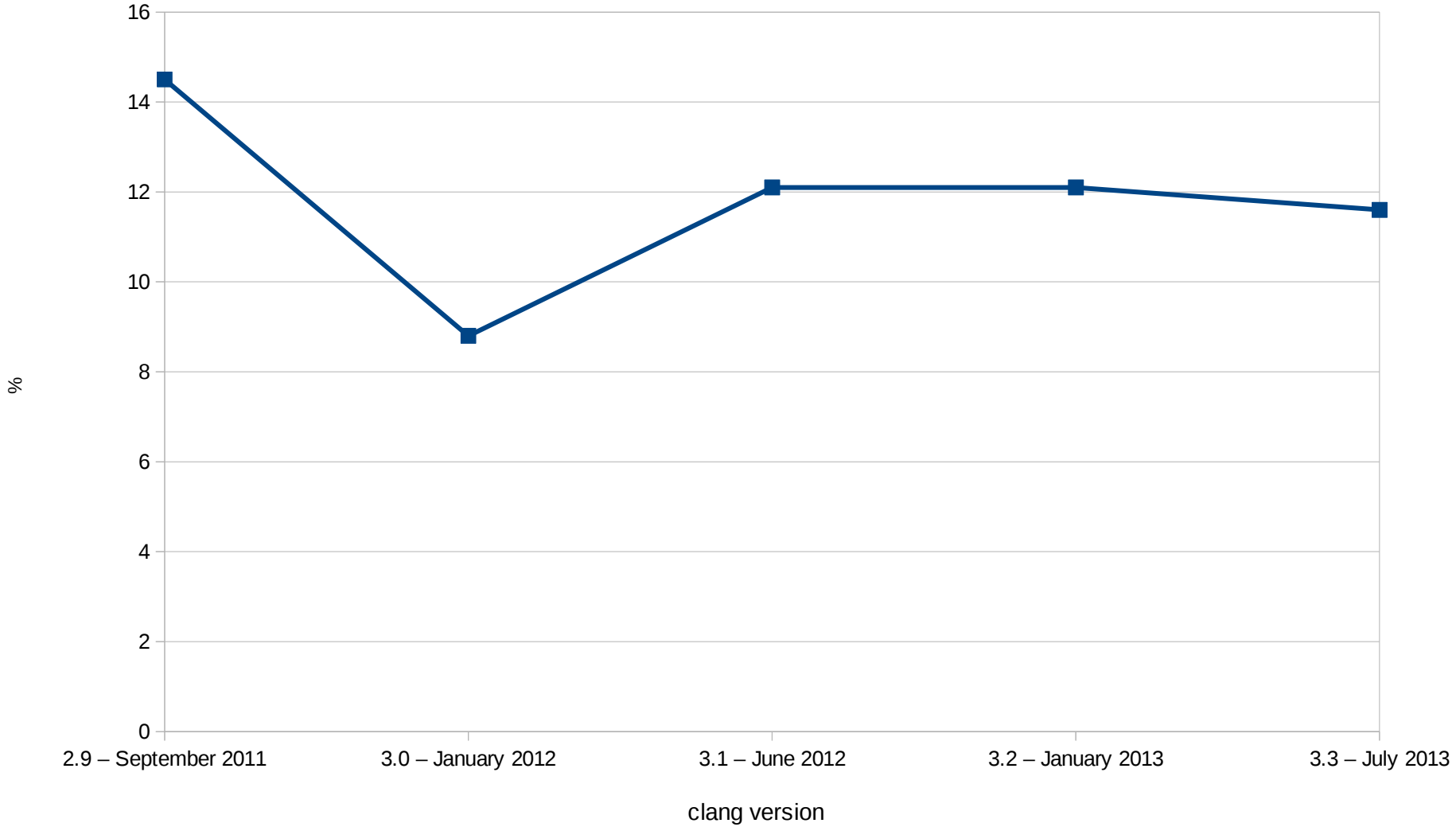
By Sylvestre Ledru (Debian, IRILL, Scilab Enterprises). February 28th 2012 (

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

Done on the cloud-qa - EC2 (Amazon cloud)
Thanks to Lucas Nussbaum and David Suarez


Percentage of failures using clang instead of gcc
16
14
12
10
8
6
4
2
0
%
2.9 – September 2011
3.0 – January 2012
3.1 – June 2012
3.2 – January 2013
3.3 – July 2013
clang version

Why these differences between 3.0 vs 3.1/3.2/3.3?

debian

Some information about *-Wall* & *-Werror* :

-Wall enables many warnings

-Werror transforms Warning to Error

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

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

$ clang -Wall -Werror foo.c && echo $?
foo.c:3:14: error: comparison of unsigned expression < 0 is always false
      [-Werror,-Wtautological-compare]
    return i < 0;
           ~ ^ ~
1 error generated.
```

Security check introduced in clang 3.1
36 occurences

```
#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.
```

Some of the most common errors

Unsupported options
49 occurrences

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

0

$ clang -O9 foo.c

error: invalid value '9' in '-O9'

Record by libdbi-drivers with -O20 \o/

Different default behavior
120 occurrences

– noreturn.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)
16 occurrences

– returninvoid.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.
```

gcc extensions which won't be supported
21 occurences

```
– foo.cpp –

#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.
```

Last rebuilds proved that clang is now ready

Remaining problems are upstream

What is *really* new since last year ?

Automatic detection of Clang error messages merged into collab-qa-tools

Started to report bugs with patches under the tag 'clang-ftbfs'

http://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=clang-ftbfs;users=pkg-llvm-team@lists.alioth.debian.org

Severity = minor

Debian Bug report logs: Bugs tagged clang-ftbfs

- Outstanding bugs -- Normal bugs; Unclassified (1 bug)
- Outstanding bugs -- Minor bugs; Patch Available (4 bugs)
- Resolved bugs -- Minor bugs (2 bugs)

Outstanding bugs -- Normal bugs; Unclassified (1 bug)

- #684508 [n| |] [aconnectgui] Use of nested functions in configure check

Outstanding bugs -- Minor bugs; Patch Available (4 bugs

- #710387 [m|+|] [xbs] xbs: FTBFS with clang instead of gcc
- #710391 [m|+|] [tcp-wrappers] tcp-wrappers: FTBFS with clang instead of gcc

We need help to report the bugs

Introduction of the llvm-toolchain packages (llvm-toolchain-3.2, llvm-toolchain-3.3, llvm-toolchain-snapshot)

Provides :

- LLVM
- Clang
- Compiler-rt
- Polly
- lldb
- cpp11-migrate (from 3.3)
- clang-format (from 3.3)

Switch from debhelper to dh
Number of line divided by 3

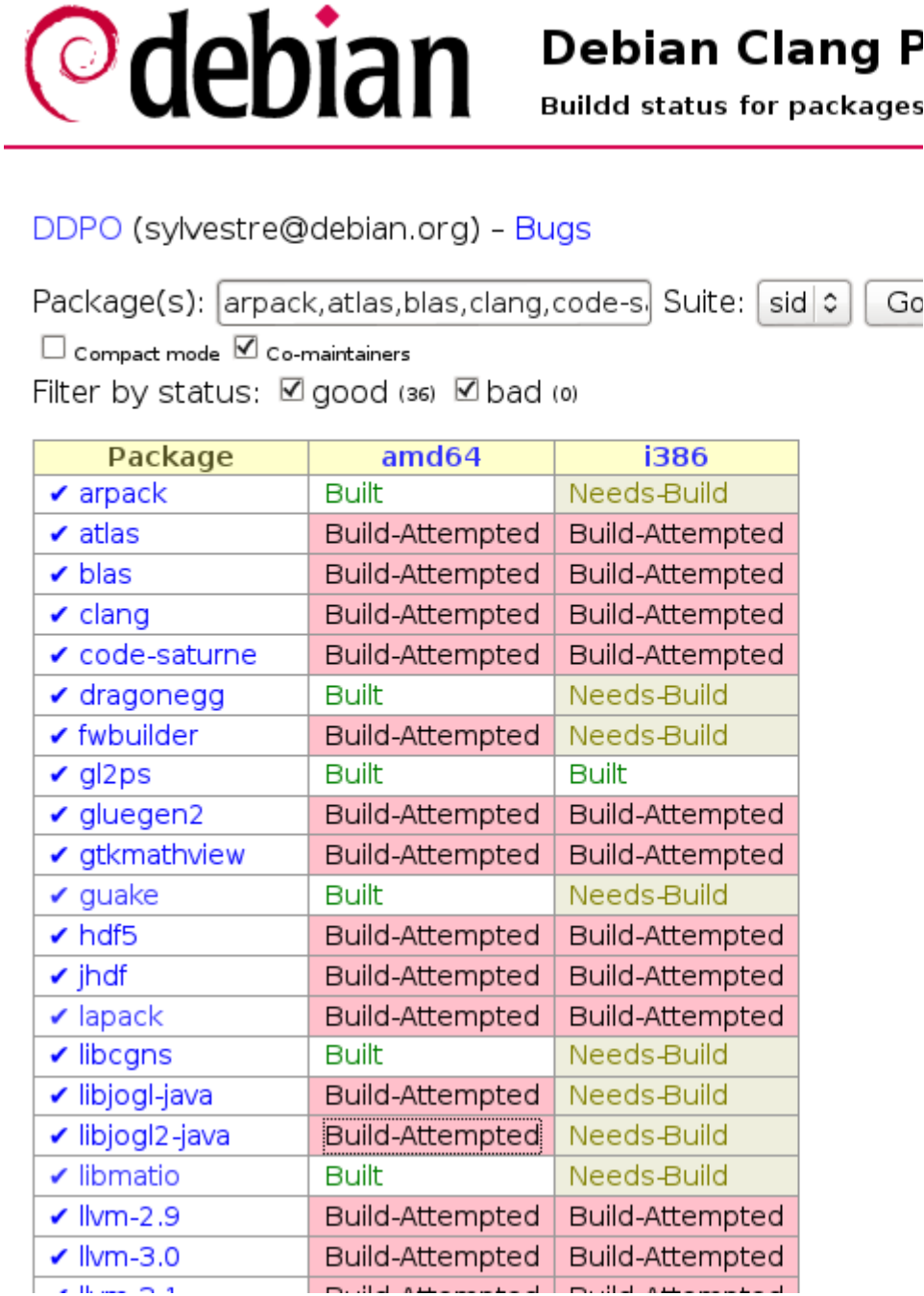
Repository with Clang built packages

deb http://clang.debian.net/repository-2013-04-07/ unstable-clang main

$ echo "deb http://clang.debian.net/repository-2013-04-07/ unstable-clang main">>/etc/apt/sources.list
$ apt-get update
$ apt-get install coreutils/unstable-clang
$ ls
$ awk

Wanna-build / buildd installed and running :

http://buildd-clang.debian.net/



But too hard to maintain and customize …

Working on debile aka Debuild.me (by Paul Tagliamonte) as part of Léo Cavaillé's Google Summer of Code

debian

debuild.me

This experimental infrastructure aims to provide a generic rebuild platform.
Normal build, custom builds (clang based) or static analyzers (coccinelle, scan-build, etc) are managed through this infrastructure.

By package		Search
By maintainer		Search

Perhaps you're looking for the last uploads

Active Jobs

Type	Arch	Suite	Assigned	Builder	Source Package
build	amd64	unstable	12 minutes ago	irill4-builder1	openmpi

Builder Status

Name	Last ping	Jobs
irill4-builder1	12 minutes ago	build
irill4-builder2	a day ago	

debian

Package info

name mercurial
version 2.7-2
uploaded by Fred the autobuilder

3/4 Remain

source jobs

debuild.me launched these jobs on the source package provided

Type	Status	Machine	Results
build *(i386)*	pending	not assigned yet	
lintian *(all)*	finished	irill4-builder1	✓ Nothing found
clanganalyzer *(all)*	finished	irill4-builder1	✗ Errors found
build *(amd64)*	finished	irill4-builder1	✓ Nothing found

Provides various workers:

- Normal builds
- scan-build
- Lintian
- Coccinelle
- Clang

Soon:

- Packaging of the debile services
- Improvements of the web interface
- Repository of clang builts packages

Implication with upstream

- Presentation at Euro LLVM 2013 in Paris

- Introduction of http://llvm.org/apt/
  2 Debian and 3 Ubuntu releases supported
  based on http://llvm-jenkins.debian.net/
  Uses the same debian/ as the one in Debian

debian

- Automatic code coverage:
http://buildd-clang.debian.net/coverage/
 76.7 % of code coverage on the whole base code

- Automatic scan-build on the llvm toolchain code:
http://buildd-clang.debian.net/scan-build/

Next steps

- Improve the debile interface
- Make sure its scale
- Provide debile to other services (a Debian PPA? Debian Mentors?)

Update the debian policy to include something like :

Hardcoded usage of CC or CXX (for example, CC=gcc) should be avoid 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

debian

Potential the rebuilds of Debian with :

- clang+plugin. Ex : polly : cache-locality optimisation auto-parallelism and vectorization, etc
- address/thread sanitizer (ASAN)
- Intel compilers (new worker?)
- Rebuild with libc++

Volunteers are welcome

Any questions ? Remarks ?