Chromium and Clang

Nico Weber and Hans Wennborg

{thakis, hans} (at) chromium.org

18th November 2011
Chromium: Overview

- Chrome is Google’s web browser
- First released 2008
- ~ 200 million active users
- Chrome is basically Chromium + branding.
Chromium: Lots of code

- ~5 million lines of code
- plus 5 million more in libraries:
  - WebKit, V8, libpng, libjpeg, ...
- 689 committers last 12 months
- Good tools are necessary.
Timeline

- Dec 2009: First patch mentioning Clang
- Apr 2010: LLVM 2.7, C++ support in alpha
- Sep 2010: Chrome builds on Linux
- Sep 2010: Chrome builds on Mac
- Sep 2010: Clang buildbot added to FYI waterfall
Timeline (contd.)

- Oct 2010: LLVM 2.8, C++ support complete
- Feb 2011: Style plugin
- Feb 2011: Clang buildbots move to main waterfall
- May 2011: ChromeOS buildbot
Timeline (contd.)

- Aug 2011: Mac bots go Clang
- Sep 2011: Mac devs are switched to Clang
- Oct 2011: Chrome 15: built with Clang on Mac
- Nov 2011: This talk.
Advantages of using Clang
Useful warnings

- Clang’s warnings are extremely useful
Useful warnings

- Clang’s warnings are extremely useful
- Look good
Useful warnings

- Clang’s warnings are extremely useful
- Look good
- Good set on by default
Useful warnings

- Clang’s warnings are extremely useful
- Look good
- Good set on by default
- Find real issues.
Useful warnings

Example: override bugs

class C {
    public:
    virtual void foo();
};

class D : public C {
    public:
    virtual void foo();
};
Useful warnings

Example: override bugs

class C {
    public:
        virtual void foo() const;
};

class D : public C {
    public:
        virtual void foo();
};
Useful warnings

Example: `-Woverloaded-virtual`

```
a.cc:8:18: warning: 'D::foo' hides overloaded virtual function [-Woverloaded-virtual]
   virtual void foo();
   ^
a.cc:3:18: note: hidden overloaded virtual function 'C::foo' declared here
   virtual void foo() const;
   ^

1 warning generated.
```
Useful warnings

Example: override specifier

```
void foo(double x) override;
```

1 error generated.

- Previously `__attribute__((override))`
- Now part of C++11 support
- Used for ~10k functions
- Stops code from breaking all the time.
Useful warnings

Example: did you mean ‘!=’?

```
a.cc:2:9: warning: using the result of an assignment as a condition without parentheses [-Wparentheses]
if (x |= y)
~~~~~~~

a.cc:2:9: note: use ’!=’ to turn this compound assignment into an inequality comparison
if (x |= y)
~~
!=
```

1 warning generated.
Useful warnings

Example: `-Wparentheses, ?:`

```
a.cc:2:16: warning: operator '?:' has lower precedence than '+'; '+' will be evaluated first
    return x + b ? y : 0;
   ~~~~~~~~~

a.cc:2:16: note: place parentheses around the '?:' expression to evaluate it first
    return x + b ? y : 0;
   ~

(a)
```

1 warning generated.

- It’s a bug every time!
Useful warnings

Example: -Wsizeof-pointer-memaccess

a.cc:8:23: warning: argument to 'sizeof' in 'memset' call is the same expression as the destination; did you mean to dereference it?
memset(s, 0, sizeof(s));
               ~

1 warning generated.
Tools

- Clang is more than a compiler
- Allows you to build your own tools.
In file included from a.cc:1:
./a.h:8:3: warning: [chromium-style] Overriding method must have "virtual" keyword.
  void foo();
^
1 warning generated.
In file included from a.cc:1:
./a.h:3:3: **warning**: [chromium-style] Complex constructor has an inlined body.
    C() {}
^
1 warning generated.
Tools

V8

- Handle<Object> for referencing gc’able memory

Handle<Object> Foo(); // Might trigger a GC.
void Bar(Object*, Object*);

Handle<Object> baz;
Bar(*Foo(), *baz);
Tools
A few rewriter attempts

- Make implicit constructor explicit
- Done using a plugin
- Fixed a few hundred instances, then gave up
- New callback mechanism, update all old call sites
- Got stuck after 4 days with arcrewrite-based system.
Other tools

AddressSanitizer (ASan)

- A fast memory error detector
- Finds use-after-free, out-of-bounds access, etc.
- Go to the talk: Ballroom Salon I/II at 4:30.
Which Clang to use

- We use Clang trunk without local patches
- Pull and test new version weekly
- Cooperating with other Clang people at Google
- When we branch for release, we branch Clang too
- Binaries: http://is.gd/chromeclang
Build numbers

Compile time (Linux)

Clang r143497, GCC 4.4.3, Chromium r108631, 8 cores, 24 GB
Build numbers

Compile time (Mac)

- Mac is also about 30% faster in Debug
- Much faster in Release.
Build numbers

Binary size (Linux, Debug)

[Graph showing size comparison between GCC and Clang for different build types]
Build numbers

Binary size (Mac)

- 10% smaller in Release.
Build numbers

Interlude: A few numbers

- 10k files @ 16 cores, $\sim 2$ s / file $\Rightarrow$ 20 min local build time
- @ 50 kB / .o, 3MB / s link $\Rightarrow$ 2.5 min
- .o file size matters!
Build numbers

Performance

[Graph showing performance scores for various benchmarks]
Build numbers

Performance

Cycle time: browser_tests
Passing thoughts

- <3 diagnostics
- <3 clang code base
- <3 the way clang is run
- <3 using clang to write own tools
- Please make it easier to write tools.
- That’s all!
- Send cakes to clang@chromium.org
- code.google.com/p/chromium/wiki/Clang