Clang & LLVM: How they can improve your life as a developer

Tilmann Scheller
Overview

- Introduction
- LLVM Overview
- Clang
- Performance
- Summary
What is LLVM?

- Mature, production-quality compiler framework
- Modular architecture
- Heavily optimizing static and dynamic compiler
- Supports all major architectures (x86, ARM, MIPS, PowerPC, ...)
- Powerful link-time optimizations (LTO)
- Permissive license (BSD-like)
LLVM sub-projects

- **Clang**
  
  C/C++/Objective C frontend and static analyzer

- **LLDB**
  
  Next generation debugger leveraging the LLVM libraries, e.g. the Clang expression parser

- **lld**
  
  Framework for creating linkers, will make Clang independent of the system linker in the future

- **Polly**
  
  Polyhedral optimizer for LLVM, e.g. high-level loop optimizations and data-locality optimizations
Which companies are contributing?

- Apple
- Google
- Samsung
- ARM
- Intel
- Qualcomm
- IBM
- Sony
- NVIDIA
- Imagination
- Cray
- AMD
- Linaro
- Azul Systems
Who is using LLVM?

- WebKit FTL JIT
- Rust
- Android (NDK, ART, RenderScript)
- Portable NativeClient (PNaCl)
- Majority of OpenCL implementations based on Clang/LLVM
- CUDA, RenderScript
- LLVM on Linux: LLVMLinux, LLVMpipe (software rasterizer in Mesa), Radeon R300-R900 drivers in Mesa
Clang users

- Default compiler on OS X
- Default compiler on FreeBSD
- Default compiler for native applications on Tizen
- Default compiler on OpenMandriva Lx starting with the next release (2015.0)
- Debian experimenting with Clang as an additional compiler (94.3% of ~20k packages successfully build with Clang 3.5)
- Android NDK ships Clang
- LLVM IR
- Scalar optimizations
- Interprocedural optimizations
- Auto-vectorizer (BB, Loop and SLP)
- Profile-guided optimizations
Compilation steps

Many steps involved in the translation from C source code to machine code:

- **Frontend:**
  - Lexing, Parsing, AST construction
  - Translation to LLVM IR

- **Middle-end**
  - Target-independent optimizations (Analyses & Transformations)

- **Backend:**
  - Translation into a DAG
  - Instruction selection: Pattern matching on the DAG
  - Instruction scheduling: Assigning an order of execution
  - Register allocation: Trying to reduce memory traffic
- The representation of the middle-end
- The majority of optimizations is done at LLVM IR level
- Low-level representation which carries type information
- RISC-like three-address code in static single assignment form with an infinite number of virtual registers
- Three different formats: bitcode (compact on-disk format), in-memory representation and textual representation (LLVM assembly language)
LLVM IR Overview

- Arithmetic: add, sub, mul, udiv, sdiv, ...
  \[
  \%tmp = \text{add i32 } \%\text{indvar}, -512
  \]
- Logical operations: shl, lshr, ashr, and, or, xor
  \[
  \%\text{shr21} = \text{ashr i32 } \%\text{mul20}, 8
  \]
- Memory access: load, store, alloca, getelementptr
  \[
  \%\text{tmp3} = \text{load i64\* } \%\text{tmp2}
  \]
- Comparison: icmp, select
  \[
  \%\text{cmp12} = \text{icmp slt i32 } \%\text{add}, 1024
  \]
- Control flow: call, ret, br, switch, ...
  \[
  \text{call void } @\text{foo(i32 } \%\text{phitmp)}
  \]
- Types: integer, floating point, vector, structure, array, ...
  \[
  \text{i32, i342, double, } <4 \times \text{float}>, \{\text{i8, } <2 \times \text{i16}>, [40 \times \text{i32}]
  \]
• Part of the backend
• Domain specific language to describe the instruction set, register file, calling conventions (TableGen)
• Pattern matcher is generated automatically
• Backend is a mix of C++ and TableGen
• Usually generates assembly code, direct machine code emission is also possible
Example

```plaintext
zx = zy = zx2 = zy2 = 0;
for (; iter < max_iter && zx2 + zy2 < 4; iter++) {
    zy = 2 * zx * zy + y;
    zx = zx2 - zy2 + x;
    zx2 = zx * zx;
    zy2 = zy * zy;
}
```
Example

\[ zx = zy = zx2 = zy2 = 0; \]
\[
\text{for} \ (; \ \text{iter} < \text{max}_\text{iter} \ \&\& \ zx2 + zy2 < 4; \ \text{iter}++) \ {\}
\[
\quad \text{zy} = 2 * zx * zy + y;
\quad zx = zx2 - zy2 + x;
\quad zx2 = zx * zx;
\quad zy2 = zy * zy;
\]

\[
\text{loop:}
\]
\[
%zy2.06 = \phi \text{ double} \ [ \ %8, \ %\text{loop} \ ], \ [ \ 0.000000e+00, \ %\text{preheader} \ ]
\quad %zx2.05 = \phi \text{ double} \ [ \ %7, \ %\text{loop} \ ], \ [ \ 0.000000e+00, \ %\text{preheader} \ ]
\quad %zy.04 = \phi \text{ double} \ [ \ %4, \ %\text{loop} \ ], \ [ \ 0.000000e+00, \ %\text{preheader} \ ]
\quad %zx.03 = \phi \text{ double} \ [ \ %6, \ %\text{loop} \ ], \ [ \ 0.000000e+00, \ %\text{preheader} \ ]
\quad %\text{iter}.02 = \phi \text{ i32} \ [ \ %9, \ %\text{loop} \ ], \ [ \ 0, \ %.\text{lr}.\text{ph}.\text{preheader} \ ]
\quad %2 = \text{fmul} \text{ double} \ %zx.03, \ 2.000000e+00
\quad %3 = \text{fmul} \text{ double} \ %2, %zy.04
\quad %4 = \text{fadd} \text{ double} \ %3, %y
\quad %5 = \text{fsub} \text{ double} %zx2.05, %zy2.06
\quad %6 = \text{fadd} \text{ double} %5, %x
\quad %7 = \text{fmul} \text{ double} %6, %6
\quad %8 = \text{fmul} \text{ double} %4, %4
\quad %9 = \text{add} \text{ i32} %\text{iter}.02, 1
\quad %10 = \text{icmp} \text{ ult} \text{ i32} %9, %\text{max}_\text{iter}
\quad %11 = \text{fadd} \text{ double} %7, %8
\quad %12 = \text{fcmp} \text{ olt} \text{ double} %11, 4.000000e+00
\quad %\text{or}.\text{cond} = \text{and}\ i1 %10, %12
\quad \text{br} i1 %\text{or}.\text{cond}, \text{label} %\text{loop}, \text{label} %\text{loopexit} \]
Example

```assembly
loop:
  // zx = zy = zx2 = zy2 = 0;
  %zy2.06 = phi double [ %8, %loop ], [ 0.000000e+00, %preheader ]
  %zx2.05 = phi double [ %7, %loop ], [ 0.000000e+00, %preheader ]
  %zy.04 = phi double [ %4, %loop ], [ 0.000000e+00, %preheader ]
  %zx.03 = phi double [ %6, %loop ], [ 0.000000e+00, %preheader ]
  %iter.02 = phi i32 [ %9, %loop ], [ 0, %preheader ]
  // zy = 2 * zx * zy + y;
  %2 = fmul double %zx.03, 2.000000e+00
  %3 = fmul double %2, %zy.04
  %4 = fadd double %3, %y
  // zx = zx2 - zy2 + x;
  %5 = fsub double %zx2.05, %zy2.06
  %6 = fadd double %5, %x
  // zx2 = zx * zx;
  %7 = fmul double %6, %6
  // zy2 = zy * zy;
  %8 = fmul double %4, %4
  // iter++
  %9 = add i32 %iter.02, 1
  // iter < max_iter
  %10 = icmp ult i32 %9, %max_iter
  // zx2 + zy2 < 4
  %11 = fadd double %7, %8
  %12 = fcmp olt double %11, 4.000000e+00
  // &&
  %or.cond = and i1 %10, %12
  br i1 %or.cond, label %loop, label %loopexit
```

```c
zx = zy = zx2 = zy2 = 0;
for (;
    iter < max_iter
    && zx2 + zy2 < 4;
    iter++)
{
    zy = 2 * zx * zy + y;
    zx = zx2 - zy2 + x;
    zx2 = zx * zx;
    zy2 = zy * zy;
}```
zx = zy = zx2 = zy2 = 0;
for (;
    iter < max_iter
    && zx2 + zy2 < 4;
    iter++) {
    zy = 2 * zx * zy + y;
    zx = zx2 - zy2 + x;
    zx2 = zx * zx;
    zy2 = zy * zy;
}
Clang

- **Goals:**
  - Fast compile time
  - Low memory usage
  - GCC compatibility
  - Expressive diagnostics

- Several tools built on top of Clang:
  - Clang static analyzer
  - clang-format, clang-modernize, clang-tidy
Clang Diagnostics

[t@ws-520 examples]$ cat t1.c
int a[2][2] = { 0, 1, 2, 3 };

[t@ws-520 examples]$ clang-3.5 -c -Wall t1.c
t1.c:1:17: warning: suggest braces around initialization of subobject [-Wmissing-braces]
int a[2][2] = { 0, 1, 2, 3 };
    ^~~~

2 warnings generated.

[t@ws-520 examples]$ gcc-4.9 -c -Wall t1.c
t1.c:1:1: warning: missing braces around initializer [-Wmissing-braces]
  int a[2][2] = { 0, 1, 2, 3 };
  ^
t1.c:1:1: warning: (near initialization for 'a[0]' ) [-Wmissing-braces]
Clang Diagnostics

[t@ws-520 examples]$ cat t2.cpp
class A {
    A(int _a, int _b) : a(_a, b(_b) {}}
        int a, b;
}

[t@ws-520 examples]$ clang++-3.5 -c -Wall t2.cpp

t2.cpp:4:13: error: expected ‘)'
    int a, b;
       ^
t2.cpp:2:26: note: to match this ‘(‘
    A(int _a, int _b) : a(_a, b(_b) {}}
       ^
t2.cpp:5:2: error: expected ‘;’ after class
t2.cpp:5:2: error: expected ‘)’ at end of input
    int a, b;
       ^
2 errors generated.

[t@ws-520 examples]$ g++-4.9 -c -Wall t2.cpp

t2.cpp:5:1: error: expected ‘;’ after class definition
}
^
t2.cpp: In constructor ‘A::A(int, int)’:
    t2.cpp:2:25: error: class ‘A’ does not have any field named ‘a’
        A(int _a, int _b) : a(_a, b(_b) {}}
        ^
t2.cpp:2:35: error: ‘b’ was not declared in this scope
        A(int _a, int _b) : a(_a, b(_b) {}}
        ^
t2.cpp:4:12: error: expected ‘)’ at end of input
        int a, b;
           ^
t2.cpp:4:12: error: expected ‘{’ at end of input

Clang Diagnostics

```c
extern bool f(int n);

void g(int a, int b)
{
    if (f(a) == 2)
        f(b);
}
```

```
[t@ws-520 examples]$ clang++-3.5 -c -Wall t3.cpp
1 warning generated.
```

```
[t@ws-520 examples]$ g++-4.9 -c -Wall t3.cpp
```

```
[t@ws-520 examples]$ cat t3.cpp
```
Clang Diagnostics

[t@ws-520 examples]$ cat t4.c
void foo(char *str) {
  strcpy(str, "foo");
}

[t@ws-520 examples]$ clang-3.5 -c -Wall t4.c
t4.c:2:3: warning: implicitly declaring library function 'strcpy' with type 'char *(char *, const char *)'
  strcpy(str, "foo");
  ^
t4.c:2:3: note: include the header <string.h> or explicitly provide a declaration for 'strcpy'
1 warning generated.

[t@ws-520 examples]$ gcc-4.9 -c -Wall t4.c
t4.c: In function 'foo':
t4.c:2:3: warning: implicit declaration of function 'strcpy' [-Wimplicit-function-declaration]
  strcpy(str, "foo");
  ^
t4.c:2:3: warning: incompatible implicit declaration of built-in function 'strcpy'

Clang Diagnostics

[t@ws-520 examples]$ cat t5.c
#include <stdio.h>
void foo(void) {
    printf("%s %d", "Hello, world");
}

[t@ws-520 examples]$ clang-3.5 -c -Wall t5.c
t5.c:3:15: warning: more '%' conversions than data arguments [-Wformat]
    printf("%s %d", "Hello, world");
    ^
1 warning generated.

[t@ws-520 examples]$ gcc-4.9 -c -Wall t5.c
t5.c:3:15: warning: more '%' conversions than data arguments [-Wformat]
    printf("%s %d", "Hello, world");
    ^
1 warning generated.
Clang Static Analyzer

- Part of Clang
- Tries to find bugs without executing the program
- Slower than compilation
- False positives
- Source annotations
- Works best on C code
- Runs from the commandline (scan-build), web interface for results
Clang Static Analyzer

- Core Checkers
- C++ Checkers
- Dead Code Checkers
- Security Checkers
- Unix Checkers
### Bug Summary

<table>
<thead>
<tr>
<th>Bug Type</th>
<th>Quantity</th>
<th>Display?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All Bugs</strong></td>
<td>228</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>API</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Argument with <code>nonnull</code> attribute passed null</td>
<td>2</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Dead store</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dead assignment</td>
<td>50</td>
<td>✔️</td>
</tr>
<tr>
<td>Dead increment</td>
<td>7</td>
<td>✔️</td>
</tr>
<tr>
<td>Dead initialization</td>
<td>3</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Logic error</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array subscript is undefined</td>
<td>2</td>
<td>✔️</td>
</tr>
<tr>
<td>Assigned value is garbage or undefined</td>
<td>3</td>
<td>✔️</td>
</tr>
<tr>
<td>Branch condition evaluates to a garbage value</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>Called C++ object pointer is null</td>
<td>76</td>
<td>✔️</td>
</tr>
<tr>
<td>Called C++ object pointer is uninitialized</td>
<td>1</td>
<td>✔️</td>
</tr>
<tr>
<td>Called function pointer is null (null dereference)</td>
<td>3</td>
<td>✔️</td>
</tr>
<tr>
<td>Dereference of null pointer</td>
<td>23</td>
<td>✔️</td>
</tr>
<tr>
<td>Division by zero</td>
<td>6</td>
<td>✔️</td>
</tr>
</tbody>
</table>
### Clang Static Analyzer

#### Reports

<table>
<thead>
<tr>
<th>Bug Group</th>
<th>Bug Type</th>
<th>File</th>
<th>Function/Method</th>
<th>Line</th>
<th>Path Length</th>
<th>View Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic error</td>
<td>Assigned value is garbage or undefined</td>
<td>lib/objc/IR/IRObjectFile.cpp</td>
<td>moveSymbolInNext</td>
<td>136</td>
<td>3</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Assigned value is garbage or undefined</td>
<td>lib/support/ScaledNumber.cpp</td>
<td>toStringAPPCheat</td>
<td>172</td>
<td>16</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Assigned value is garbage or undefined</td>
<td>lib/Target/X86/X86SELLowering.cpp</td>
<td>operator()</td>
<td>8532</td>
<td>7</td>
<td>View Report</td>
</tr>
<tr>
<td>Dead store</td>
<td>Dead increment</td>
<td>lib/llvm/obj/ldd-platform/Platform.cpp</td>
<td>main</td>
<td>238</td>
<td>1</td>
<td>View Report</td>
</tr>
<tr>
<td>Dead store</td>
<td>Dead increment</td>
<td>lib/llvm/obj/ldd-platform/Platform.cpp</td>
<td>WriteMemory</td>
<td>2536</td>
<td>1</td>
<td>View Report</td>
</tr>
<tr>
<td>Dead store</td>
<td>Dead increment</td>
<td>lib/llvm/obj/ldd-mtl/Crun/LinuxDebugging.cpp</td>
<td>MIPSResponseMbrkPtInfo</td>
<td>1253</td>
<td>1</td>
<td>View Report</td>
</tr>
<tr>
<td>Dead store</td>
<td>Dead increment</td>
<td>lib/llvm/obj/ldd-mtl/Crun/LinuxDebugging.cpp</td>
<td>MIPSStoppedAtBreakPoint</td>
<td>1069</td>
<td>1</td>
<td>View Report</td>
</tr>
<tr>
<td>Dead store</td>
<td>Dead increment</td>
<td>lib/llvm/obj/ldd-platform/Platform.cpp</td>
<td>main</td>
<td>237</td>
<td>1</td>
<td>View Report</td>
</tr>
<tr>
<td>Dead store</td>
<td>Dead increment</td>
<td>lib/llvm/obj/ldd-platform/Platform.cpp</td>
<td>DecodeLD4Dup instruction</td>
<td>2952</td>
<td>1</td>
<td>View Report</td>
</tr>
<tr>
<td>Dead store</td>
<td>Dead increment</td>
<td>lib/llvm/obj/ldd-platform/Platform.cpp</td>
<td>WriteAllRegisterValues</td>
<td>217</td>
<td>1</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>build-llvm-analysis-object/interpreter/LLDBWrapPython.cpp</td>
<td>_wrap_SBTARGET_LAUNCH_NONE</td>
<td>40975</td>
<td>12</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>build-llvm-analysis-object/interpreter/LLDBWrapPython.cpp</td>
<td>_wrap_SBTARGET_LAUNCH_SIMPLE</td>
<td>41166</td>
<td>9</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>build-llvm-analysis-object/interpreter/LLDBWrapPython.cpp</td>
<td>_wrap_SBPATTERN_CSTRING_FROM_MEMORY</td>
<td>33539</td>
<td>9</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>clang/test/c-index-test/c-index-test.t</td>
<td>perform_test_reparray_source</td>
<td>1598</td>
<td>28</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>clang/test/c-index-test/c-index-test.t</td>
<td>find_file_includes_in</td>
<td>2478</td>
<td>7</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>build-llvm-analysis-object/interpreter/LLDBWrapPython.cpp</td>
<td>_wrap_SBPATTERN_READ_MEMORY</td>
<td>33405</td>
<td>9</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>build-llvm-analysis-object/interpreter/LLDBWrapPython.cpp</td>
<td>TestBody</td>
<td>47</td>
<td>2</td>
<td>View Report</td>
</tr>
<tr>
<td>Memory Error</td>
<td>Memory leak</td>
<td>build-llvm-analysis-object/interpreter/LLDBWrapPython.cpp</td>
<td>_wrap_SBPATTERN_REMOTE_LAUNCH</td>
<td>32437</td>
<td>9</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/CodeGen/ dump/CodeGenPrinter.cpp</td>
<td>EmlAlignment</td>
<td>1534</td>
<td>5</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/CodeGen/InlineSplit.cpp</td>
<td>lssNPInliner</td>
<td>249</td>
<td>7</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>tools/lib-obj-dump/MachOdDump.cpp</td>
<td>SegInfo</td>
<td>2746</td>
<td>8</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/Target/Mips/instAlign/mipsAlignImmediate.cpp</td>
<td>GetInstSeq.s LL</td>
<td>44</td>
<td>25</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/Target/AArch64/AArch64SeLcInstToDAG.cpp</td>
<td>SelectArchInstUnscaled</td>
<td>648</td>
<td>5</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/Target/NEON/NEONExtDAGToDAG.cpp</td>
<td>SelectBIE</td>
<td>4812</td>
<td>10</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/Target/ARM/ARMSeLcLowering.cpp</td>
<td>PerformBFCcombine</td>
<td>8576</td>
<td>4</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/Target/ARM/ARMConstantInstPass.cpp</td>
<td>UnknownPadding</td>
<td>69</td>
<td>8</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/CodeGen/ExecutionDepFix.cpp</td>
<td>operator()</td>
<td>8476</td>
<td>8</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/CodeGen/ExecutionDepFix.cpp</td>
<td>hasDomain</td>
<td>78</td>
<td>12</td>
<td>View Report</td>
</tr>
<tr>
<td>Logic error</td>
<td>Result of operation is garbage or undefined</td>
<td>lib/MC/MC/AsmStreamer.cpp</td>
<td>EmValueImpl</td>
<td>685</td>
<td>14</td>
<td>View Report</td>
</tr>
</tbody>
</table>
... 

```cpp
const SCEV *MaxBECOUNT = getCouldNotCompute();

if (isa<SCEVConstant>(BECOUNT))
    MaxBECOUNT = BECOUNT;
else
    MaxBECOUNT = computeBECOUNT(getConstant(MaxStart - MinEnd),
    getConstant(MinStride), false);

if (isa<SCEVCouldNotCompute>(MaxBECOUNT))
    MaxBECOUNT = BECOUNT;

return ExitLimit(BECOUNT, MaxBECOUNT, /*MustExit=*/true);
```
NestedNameSpecifierLocBuilder &
NestedNameSpecifierLocBuilder::
  operator=(const NestedNameSpecifierLocBuilder &Other) {
    Representation = Other.Representation;

    if (Buffer && Other.Buffer && BufferCapacity >= Other.BufferSize) {
      // Re-use our storage.
      BufferSize = Other.BufferSize;
      memcpy(Buffer, Other.Buffer, BufferSize);
      return *this;
    }

    // Free our storage, if we have any.
    if (BufferCapacity) {
      // Taking true branch
      free(Buffer);

      // Taking false branch
      BufferCapacity = 0;
    }

    if (!Other.Buffer) {
      // Taking false branch
      // Empty.
      Buffer = nullptr;
      BufferSize = 0;
      return *this;
    }

    // Deep copy.
    if (Other.BufferCapacity == 0) {
      // Taking false branch
      // Shallow copy is okay.
      Buffer = Other.Buffer;
      BufferSize = Other.BufferSize;
      return *this;
    }

    // Calling 'Append'
    Append(Other.Buffer, Other.Buffer + Other.BufferSize,
           Buffer, BufferSize,
           BufferCapacity);
    return *this;
  }
namespace {
  void Append(char *Start, char *End, char **Buffer, unsigned &BufferSize, unsigned &BufferCapacity) {
    if (BufferSize + (End - Start) > BufferCapacity) {

      // Reallocate the buffer.
      unsigned NewCapacity
      = std::max((unsigned)(BufferCapacity? BufferCapacity * 2

      : sizeof(void*) * 2),
          (unsigned)(BufferSize + (End - Start)));
      char *NewBuffer = static_cast<char *>(malloc(NewCapacity));
      memcpy(NewBuffer, Buffer, BufferSize);

      if (BufferCapacity)
        free(Buffer);
      Buffer = NewBuffer;
      BufferCapacity = NewCapacity;
    }

    memcpy(Buffer + BufferSize, Start, End - Start);
    BufferSize += End - Start;
  }
}
- Automatic formatting
- Developers waste time on formatting
- Supports different style guides
- Consistent coding style is important
- Detect bug prone coding patterns
- Enforce coding conventions
- Advocate modern and maintainable code
- Checks can be more expensive than compilation
Sanitizers

- LLVM/Clang-based Sanitizer projects:
  - AddressSanitizer – Fast memory error detector
  - ThreadSanitizer – Detects data races
  - LeakSanitizer – Memory leak detector
  - MemorySanitizer – Detects reads of uninitialized variables
  - UBSanitizer – Detects undefined behavior
What's new?

- LLVM 3.5 (released in September)
- Self-hosting on SPARC64 (Linux, FreeBSD)
- Integrated assembler and EHABI enabled by default on ARM
- Merging of the AArch64 backends completed
- Optimization reports
- Experimental support for C++17 features
Current major efforts

- Better Windows support
- Improved debugging
- LTO parallelization
- Vectorization
- Profile-guided optimizations
  - Profiling infrastructure (sampling/instrumentation-based)
  - Analyses and transformations which take advantage of the gathered data
Performance

- SPEC CPU2000 comparison of GCC 4.9 vs LLVM 3.4
- GCC generates 6% faster code at -O3 and 2% faster code when doing LTO
- Compiling at -Ofast -march=core-avx2 -mtune=corei7
- Measured on Haswell 3.4GHz i5-4670
Clang 3.5 and GCC 4.9.1 binaries both compiled with GCC 4.9.1 at -O2

- Clang 3.5 release build (-O3)
  - Clang 3.5: 6m46s
  - GCC 4.9.1: 9m56s

- Clang 3.5 debug build (-O0 -gsplit-dwarf)
  - Clang 3.5: 7m13s
  - GCC 4.9.1: 10m34s

Clang ~46% faster in both builds!

Measured on Fedora 20 (3.5GHz i7-4770K, 16GB RAM, 2TB HDD)
Summary

• Great compiler infrastructure
• Fast C/C++ compiler with expressive diagnostics
• Bug detection at compile time
• Automated formatting of code
• Detect memory bugs early with Sanitizers
Give it a try!

- Visit llvm.org
- Distributions with Clang/LLVM packages:
  - Fedora
  - Debian/Ubuntu
  - openSUSE
  - ...

Thank you.