Boosting Developer Productivity with Clang

Tilmann Scheller
Senior LLVM Compiler Engineer
t.scheller@samsung.com

Samsung Open Source Group
Samsung Research UK

LinuxCon Europe 2015
Dublin, Ireland, October 5 – 7, 2015

Overview

- Introduction
- LLVM Overview
- Clang
- Performance
- Summary

Introduction

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)
- 2.5M C++ LOC (LLVM + Clang combined)

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?

SAMSUNG Open Source Group

Google

SAMSUNG

ARM®

intel®

QUALCOMM®

IBM

SONY COMPUTER ENTERTAINMENT ®

NVIDIA®

Imagination

CRAY

Linaro

AMD

AZUL SYSTEMS®

Who is using LLVM?

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

Clang users

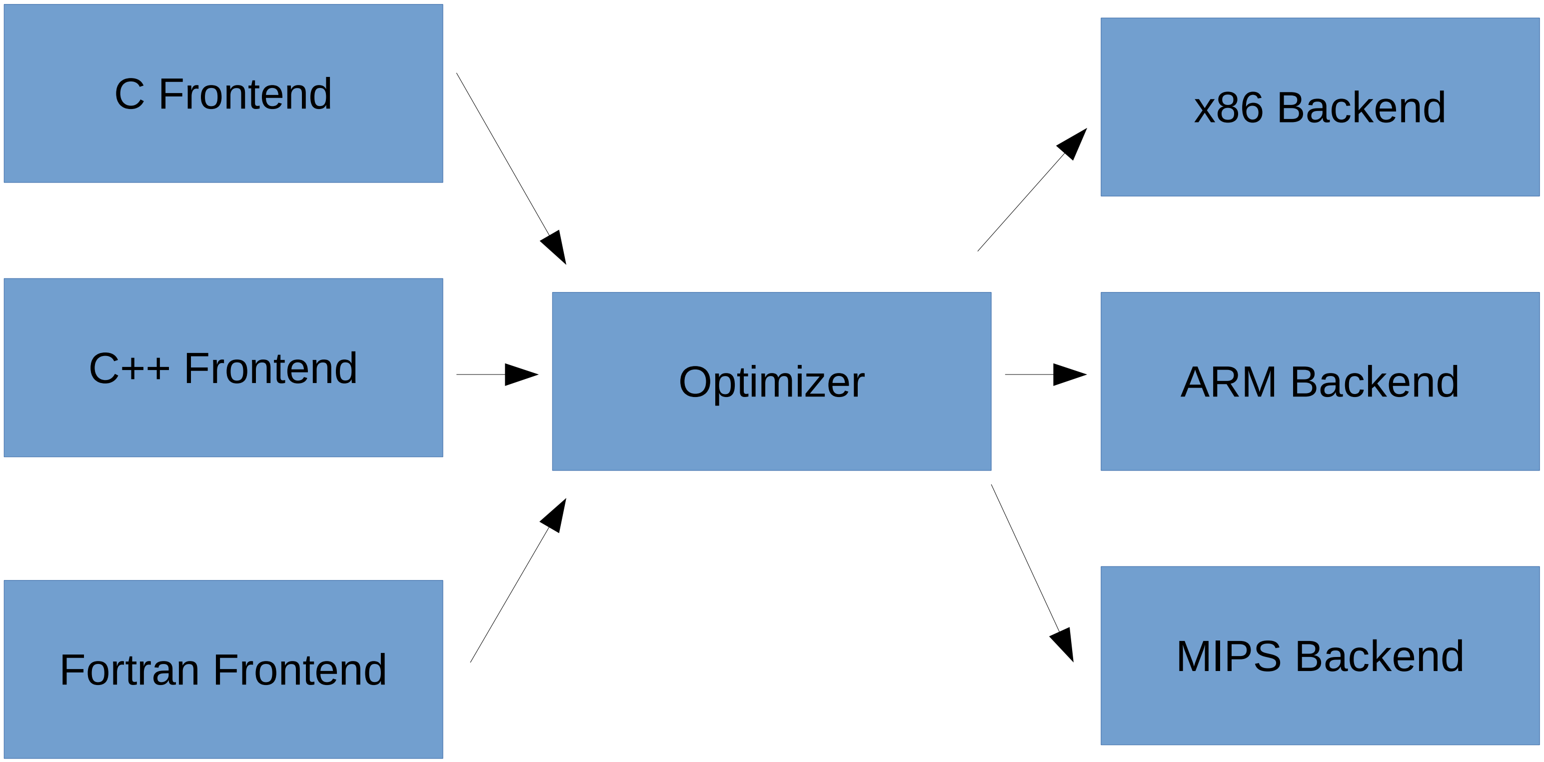
- Default compiler on OS X
- Default compiler on FreeBSD
- Default compiler for native applications on Tizen
- Default compiler on OpenMandriva Lx starting with the next release (2015.0)
- Debian experimenting with Clang as an additional compiler (94.1% of ~22k packages successfully build with Clang 3.6)
- Android NDK ships Clang

LLVM Overview

LLVM

- LLVM IR (Intermediate Representation)
- Scalar optimizations
- Interprocedural optimizations
- Auto-vectorizer (BB, Loop and SLP)
- Profile-guided optimizations

Compiler architecture



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

LLVM Intermediate Representation

- 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

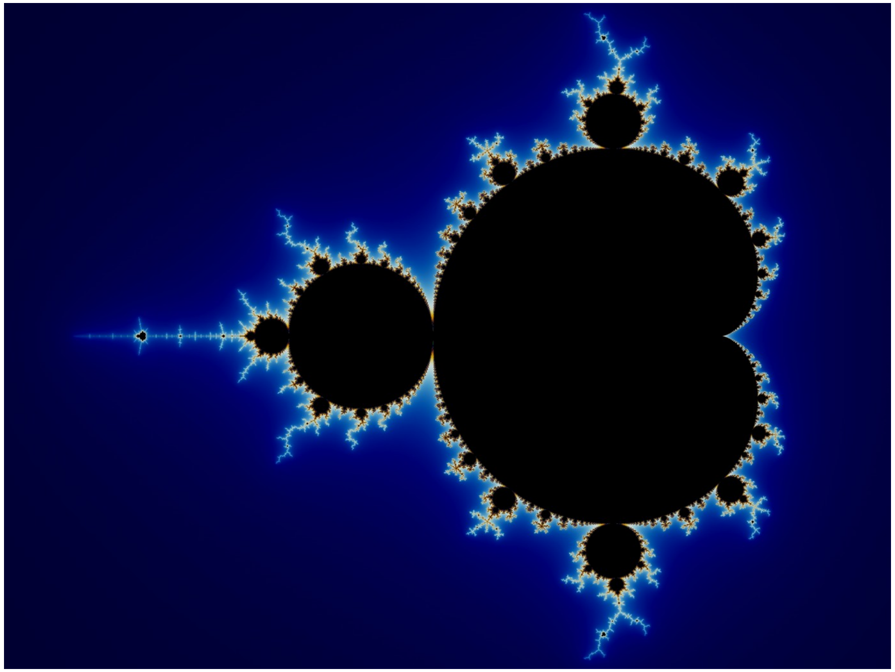
- **Arithmetic: add, sub, mul, udiv, sdiv, ...**
  - %tmp = add i32 %indvar, -512
- **Logical operations: shl, lshr, ashr, and, or, xor**
  - %shr21 = ashr i32 %mul20, 8
- **Memory access: load, store, alloca, getelementptr**
  - %tmp3 = load i64* %tmp2
- **Comparison: icmp, select**
  - %cmp12 = icmp slt i32 %add, 1024
- **Control flow: call, ret, br, switch, ...**
  - call void @foo(i32 %phitmp)
- **Types: integer, floating point, vector, structure, array, ...**
  - i32, i342, double, <4 x float>, {i8, <2 x i16>}, [40 x i32]

Target-independent code generator

- 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

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

```
loop:
  %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, %.lr.ph.preheader ]
  %2 = fmul double %zx.03, 2.000000e+00
  %3 = fmul double %2, %zy.04
  %4 = fadd double %3, %y
  %5 = fsub double %zx2.05, %zy2.06
  %6 = fadd double %5, %x
  %7 = fmul double %6, %6
  %8 = fmul double %4, %4
  %9 = add i32 %iter.02, 1
  %10 = icmp ult i32 %9, %max_iter
  %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
```

Example

```
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
```

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

```
.LBB0_2:
        @ d17 = 2 * zx
        vadd.f64        d17, d12, d12
        @ iter < max_iter
        cmp     r1, r0
        @ d17 = (2 * zx) * zy
        vmul.f64        d17, d17, d11
        @ d18 = zx2 - zy2
        vsub.f64        d18, d10, d8
        @ d12 = (zx2 – zy2) + x
        vadd.f64        d12, d18, d0
        @ d11 = (2 * zx * zy) + y
        vadd.f64        d11, d17, d9
        @ zx2 = zx * zx
        vmul.f64        d10, d12, d12
        @ zy2 = zy * zy
        vmul.f64        d8, d11, d11
        bhs     .LBB0_5
@ BB#3:
        @ zx2 + zy2
        vadd.f64        d17, d10, d8
        @ iter++
        adds    r1, #1
        @ zx2 + zy2 < 4
        vcmpe.f64       d17, d16
        vmrs    APSR_nzcv, fpscr
        bmi     .LBB0_2
        b       .LBB0_5
```

Clang

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 };
                ^~~~
                {   }
t1.c:1:24: 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
}
 ^
 ;
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

```
[t@ws-520 examples]$ cat t3.cpp
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
t3.cpp:5:12: warning: comparison of constant 2 with expression of type 'bool'
             is always false [-Wtautological-constant-out-of-range-compare]
  if (f(a) == 2)
      ~~~~ ^  ~
1 warning generated.
```

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

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: In function ‘foo’:
t5.c:3:3: warning: format ‘%d’ expects a matching ‘int’ argument [-Wformat=]
   printf("%s %d", "Hello, world");
   ^
```

Clang Static Analyzer

- Part of Clang
- Tries to find bugs without executing the program
- Slower than compilation
- False positives
- 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

Clang Static Analyzer

llvm-toolchain-snapshot-3.6~svn219049 - scan-build results - Google Chrome

llvm-toolchain-snapsh ×

llvm.org/reports/scan-build/

llvm-toolchain-snapshot-3.6~svn219049 - scan-build results

User:	pbuilder@irill3
Working Directory:	/tmp/buildd/llvm-toolchain-snapshot-3.6~svn219049
Command Line:	/usr/bin/make -j 7 -C build-llvm VERBOSE=1 CLANG_VENDOR=Debian 'CXXFLAGS= -std=c++0x' 'LDFLAGS= -fuse-ld=gold' REQUIRES_RTTI=1 DEBUGMAKE=1
Clang Version:	Debian clang version 3.6.0-svn219049-1~exp1 (trunk) (based on LLVM 3.6.0)
Date:	Sat Oct 4 12:22:52 2014

Bug Summary

Bug Type	Quantity	Display?
All Bugs	**228**	☑
API		
Argument with 'nonnull' attribute passed null	2	☑
Dead store		
Dead assignment	53	☑
Dead increment	7	☑
Dead initialization	3	☑
Logic error		
Array subscript is undefined	2	☑
Assigned value is garbage or undefined	3	☑
Branch condition evaluates to a garbage value	1	☑
Called C++ object pointer is null	76	☑
Called C++ object pointer is uninitialized	1	☑
Called function pointer is null (null dereference)	3	☑
Dereference of null pointer	23	☑
Division by zero	6	☑

Clang Static Analyzer

Reports

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length	
Logic error	Assigned value is garbage or undefined	lib/Object/IRObjectFile.cpp	moveSymbolNext	186	3	View Report
Logic error	Assigned value is garbage or undefined	lib/Support/ScaledNumber.cpp	toStringAPFloat	172	16	View Report
Logic error	Assigned value is garbage or undefined	lib/Target/X86/X86ISelLowering.cpp	operator()	8532	7	View Report
Dead store	Dead increment	lldb/tools/lldb-platform/lldb-platform.cpp	main	238	1	View Report
Dead store	Dead increment	lldb/source/Target/Process.cpp	WriteMemory	2536	1	View Report
Dead store	Dead increment	lldb/tools/lldb-mi/MICmnLLDBDebugSessionInfo.cpp	MIResponseFormBrkPtInfo	1253	1	View Report
Dead store	Dead increment	lldb/tools/lldb-mi/MICmnLLDBDebuggerHandleEvents.cpp	MiStoppedAtBreakPoint	1069	1	View Report
Dead store	Dead increment	lldb/tools/lldb-platform/lldb-platform.cpp	main	237	1	View Report
Dead store	Dead increment	lib/Target/ARM/Disassembler/ARMDisassembler.cpp	DecodeVLD4DupInstruction	2982	1	View Report
Dead store	Dead increment	lldb/source/Plugins/Process/POSIX/RegisterContextPOSIXProcessMonitor_mips64.cpp	WriteAllRegisterValues	217	1	View Report
Memory Error	Memory leak	build-llvm/tools/lldb/source/Interpreter/LLDBWrapPython.cpp	_wrap_SBTarget_Launch__SWIG_0	40975	12	View Report
Memory Error	Memory leak	build-llvm/tools/lldb/source/Interpreter/LLDBWrapPython.cpp	_wrap_SBTarget_LaunchSimple	41116	9	View Report
Memory Error	Memory leak	build-llvm/tools/lldb/source/Interpreter/LLDBWrapPython.cpp	_wrap_SBProcess_ReadCStringFromMemory	33539	9	View Report
Memory Error	Memory leak	clang/tools/c-index-test/c-index-test.c	perform_test_reparse_source	1598	28	View Report
Memory Error	Memory leak	clang/tools/c-index-test/c-index-test.c	find_file_includes_in	2448	7	View Report
Memory Error	Memory leak	build-llvm/tools/lldb/source/Interpreter/LLDBWrapPython.cpp	_wrap_SBProcess_ReadMemory	33405	9	View Report
Memory Error	Memory leak	unittests/IR/WaymarkTest.cpp	TestBody	47	2	View Report
Memory Error	Memory leak	build-llvm/tools/lldb/source/Interpreter/LLDBWrapPython.cpp	_wrap_SBProcess_RemoteLaunch	32437	9	View Report
Logic error	Result of operation is garbage or undefined	lib/CodeGen/AsmPrinter/AsmPrinter.cpp	EmitAlignment	1534	5	View Report
Logic error	Result of operation is garbage or undefined	lib/CodeGen/InlineSpiller.cpp	isSnippet	249	7	View Report
Logic error	Result of operation is garbage or undefined	tools/llvm-objdump/MachODump.cpp	SegInfo	2746	8	View Report
Logic error	Result of operation is garbage or undefined	lib/Target/Mips/MipsAnalyzeImmediate.cpp	GetInstSeqLsSLL	44	25	View Report
Logic error	Result of operation is garbage or undefined	lib/Target/AArch64/AArch64ISelDAGToDAG.cpp	SelectAddrModeUnscaled	648	5	View Report
Logic error	Result of operation is garbage or undefined	lib/Target/NVPTX/NVPTXISelDAGToDAG.cpp	SelectBFE	4842	10	View Report
Logic error	Result of operation is garbage or undefined	lib/Target/ARM/ARMISelLowering.cpp	PerformBFICombine	8576	4	View Report
Logic error	Result of operation is garbage or undefined	lib/Target/ARM/ARMConstantIslandPass.cpp	UnknownPadding	69	8	View Report
Logic error	Result of operation is garbage or undefined	lib/Target/X86/X86ISelLowering.cpp	operator()	8476	8	View Report
Logic error	Result of operation is garbage or undefined	lib/CodeGen/ExecutionDepsFix.cpp	hasDomain	78	12	View Report
Logic error	Result of operation is garbage or undefined	lib/MC/MCAsmStreamer.cpp	EmitValueImpl	685	14	View Report

Clang Static Analyzer - Example

```
...
  const SCEV *MaxBECount = getCouldNotCompute();
```

Value stored to 'MaxBECount' during its initialization is never read

```
  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);
}
```

Clang Static Analyzer - Example

```
NestedNameSpecifierLocBuilder &
NestedNameSpecifierLocBuilder::
operator=(const NestedNameSpecifierLocBuilder &Other) {
  Representation = Other.Representation;

  if (Buffer && Other.Buffer && BufferCapacity >= Other.BufferSize) {

  1 Taking false branch →

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

  // Free our storage, if we have any.
  if (BufferCapacity) {

  2 ← Taking true branch →

    free(Buffer);

    3 ← Memory is released →

    BufferCapacity = 0;
  }

  if (!Other.Buffer) {

  4 ← Taking false branch →

    // Empty.
    Buffer = nullptr;
    BufferSize = 0;
    return *this;
  }

  if (Other.BufferCapacity == 0) {

  5 ← Taking false branch →

    // Shallow copy is okay.
    Buffer = Other.Buffer;
    BufferSize = Other.BufferSize;
    return *this;
  }

  // Deep copy.
  Append(Other.Buffer, Other.Buffer + Other.BufferSize, Buffer, BufferSize,

  6 ← Calling 'Append' →

         BufferCapacity);
  return *this;
}
```

Clang Static Analyzer - Example

```
namespace {
  void Append(char *Start, char *End, char *&Buffer, unsigned &BufferSize,
              unsigned &BufferCapacity) {
    if (BufferSize + (End - Start) > BufferCapacity) {
```

7 ← Taking true branch →

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

8 ← '?' condition is false →

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

9 ← Use of memory after it is freed

```

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

    memcpy(Buffer + BufferSize, Start, End - Start);
    BufferSize += End-Start;
  }
```

clang-format

- Automatic formatting
- Developers waste time on formatting
- Supports different style guides
- Consistent coding style is important

clang-tidy

- Detect bug prone coding patterns
- Enforce coding conventions
- Advocate modern and maintainable code
- Checks can be more expensive than compilation

clang-modernize

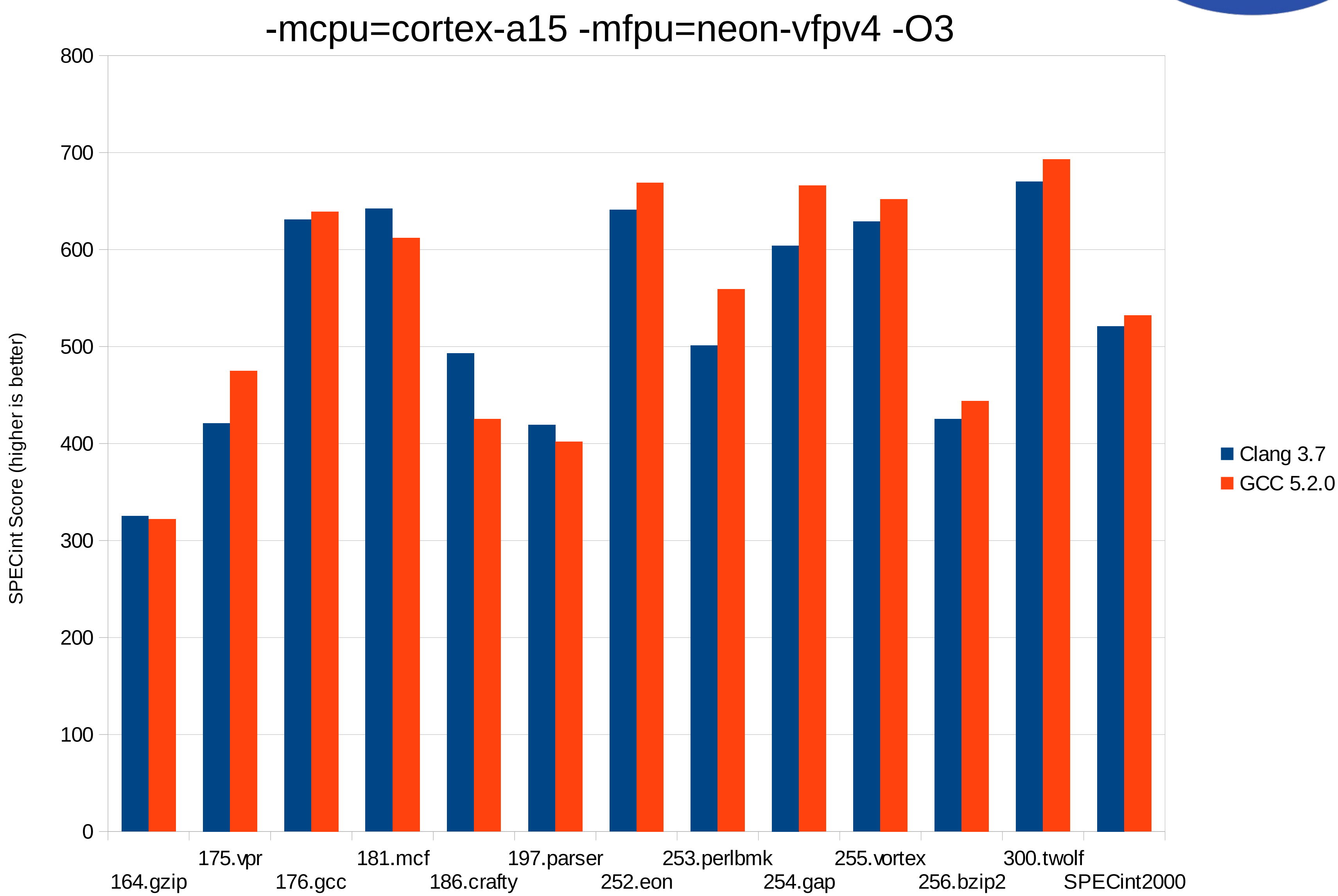
- Move source code to newer C++ standards
- Source-to-source translation
- Makes use of C++11 range-based for loops where possible
- Makes use of the new C++11 keyword nullptr where possible
- ...and many more transformations

Sanitizers

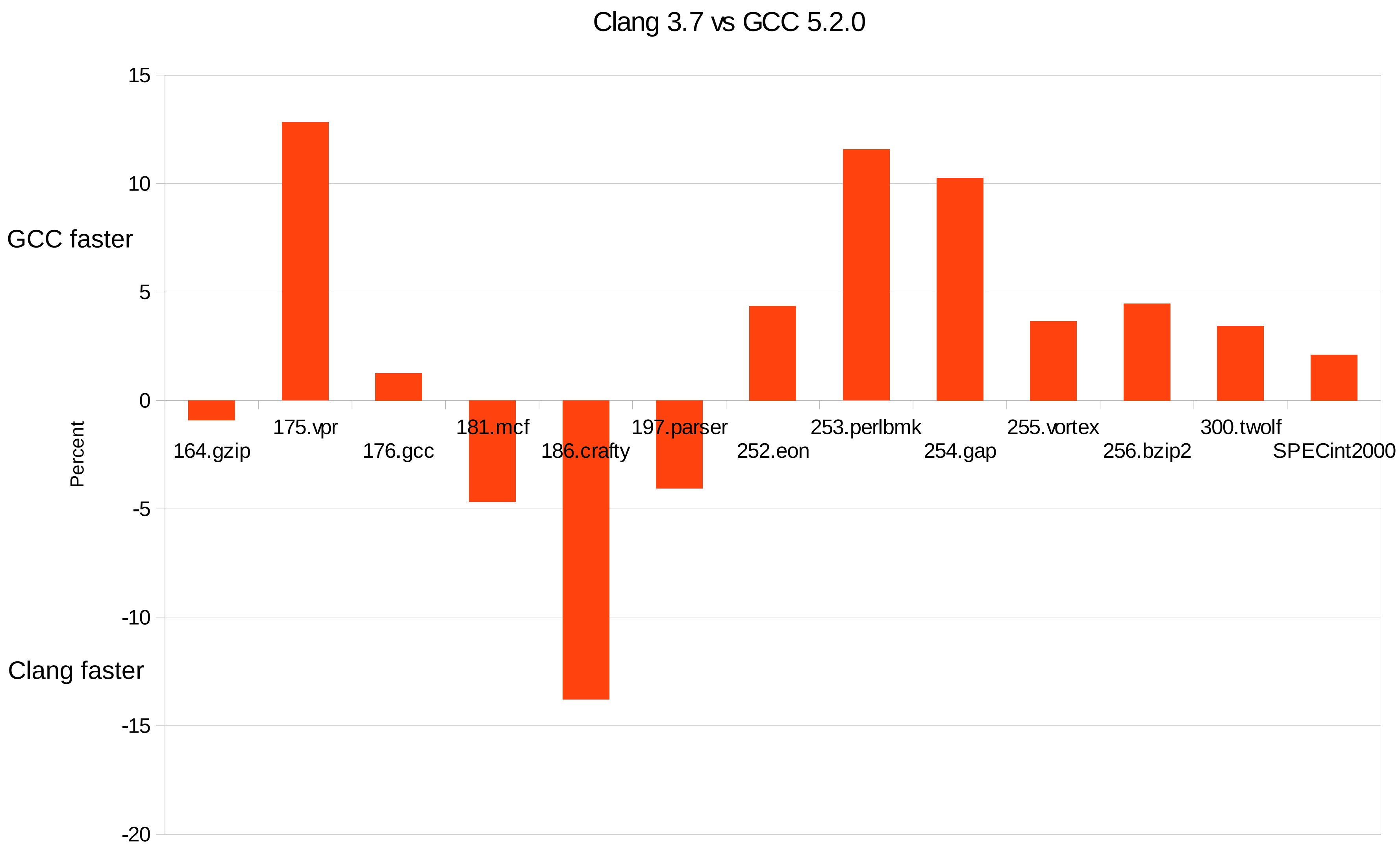
- LLVM/Clang-based Sanitizer projects:
  - AddressSanitizer – Fast memory error detector
  - ThreadSanitizer – Detects data races
  - LeakSanitizer – Memory leak detector
  - MemorySanitizer – Detects reads of uninitialized variables
  - UBSanitizer – Detects undefined behavior

Performance - SPEC CPU2000

SPEC CPU2000

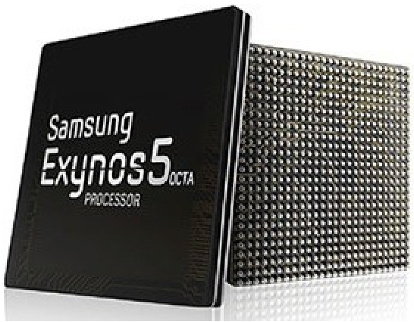


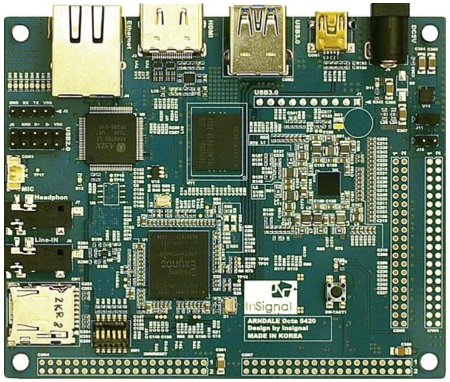
SPEC CPU2000 - Relative performance



SPEC CPU2000

- On average GCC is just ~2% faster
- Three benchmarks where GCC is doing significantly better: 175.vpr, 253.perlbmk, 254.gap
- 254.gap relies on signed overflow, needs to be compiled with -fwrapv
- Measured on an Arndale Octa board running Ubuntu 14.04 (quad-core Cortex-A15 @ 1.8GHz, 2GB of RAM)





Performance - Compile time

Compile time experiment

- Squeeze maximum performance out of the toolchain
- Speed up the build without buying new hardware
- Building Clang (2.5M C++ LOC), results should be applicable to other large C++ applications
- Focus on x86-64 Linux
- Test machine: Fedora 21 on i7-4770K CPU @ 3.50GHz, 16GB RAM, 2TB 7200RPM HDD

Ideas

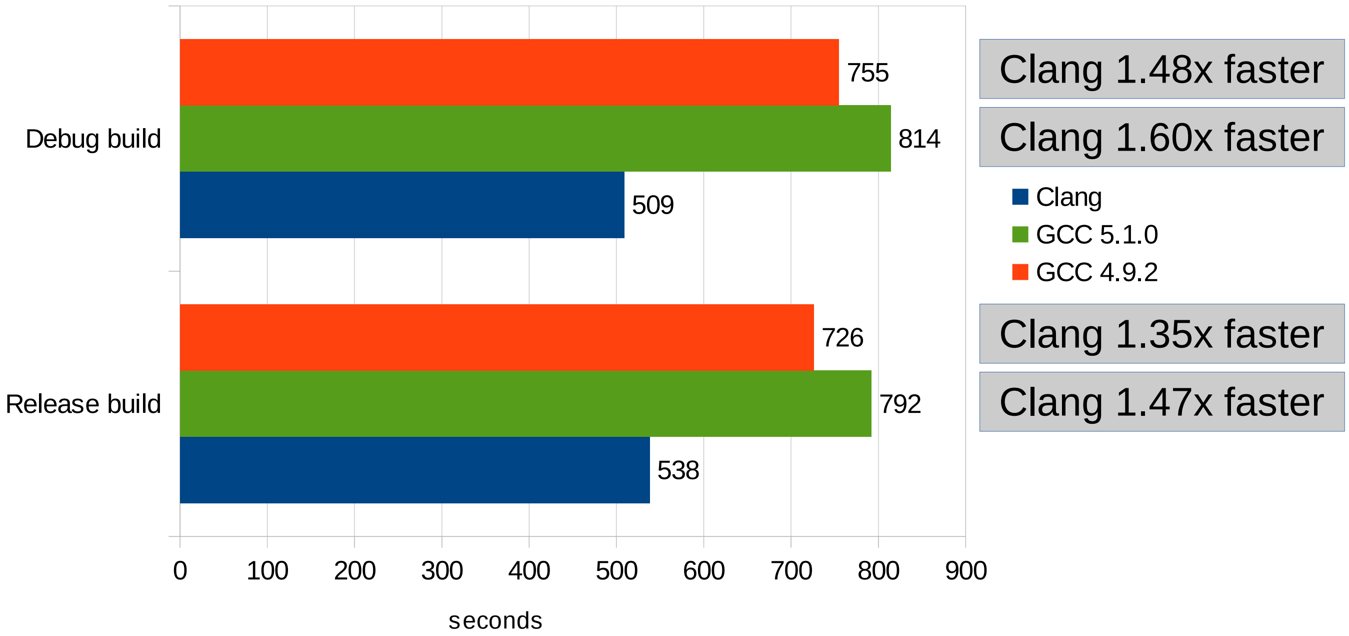
- Build with Clang instead of GCC
- Use a faster linker: GNU gold instead of GNU ld
- Use a heavily optimized compiler binary for the build (LTO, PGO, LTO+PGO)
- Use the CMake Ninja generator rather than the default Makefile generator

Measurement

- Using the GCC 4.9.2 binary shipping with Fedora 21
- Clang trunk snapshot (r234392 from April 8, 2015)
- Standard debug/release builds of Clang (CMake build with Ninja generator) unless otherwise noted
- Using GNU gold 2.24 for linking
- Make invoked with -j8
- Best of five runs

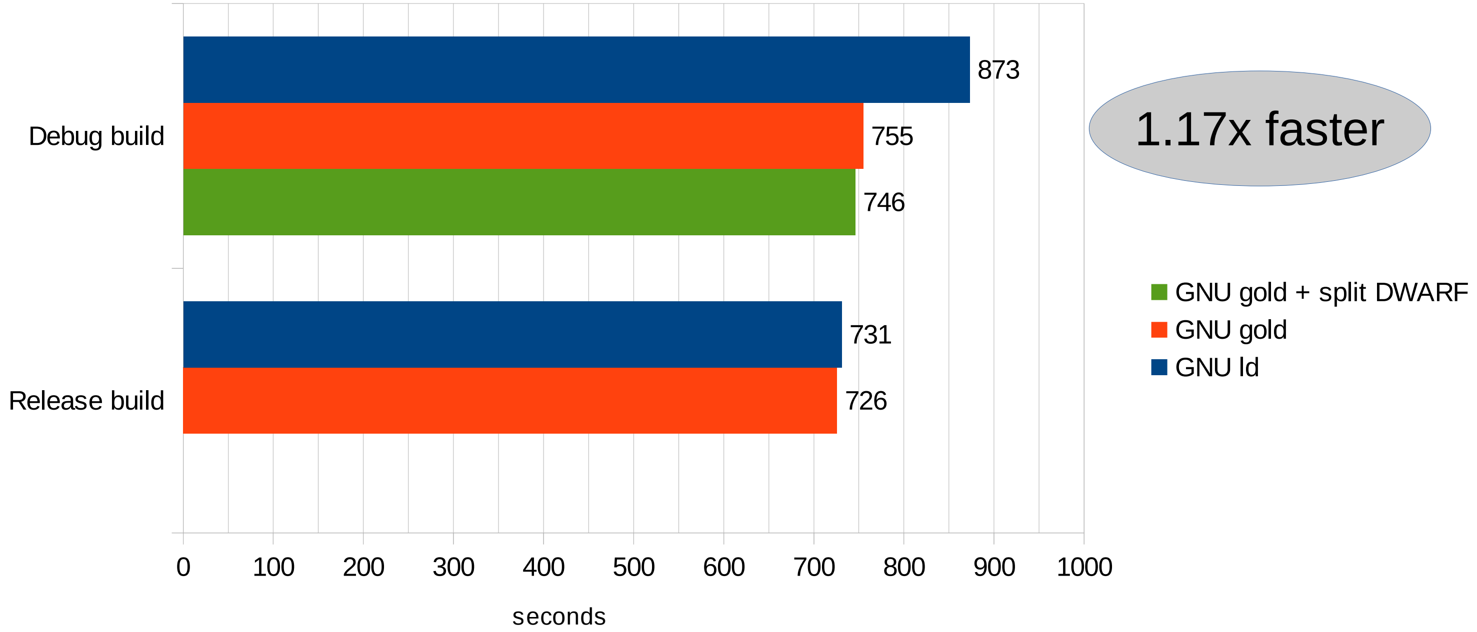
Clang vs. GCC compile time

- Both Clang and GCC are compiled with GCC 4.9.2



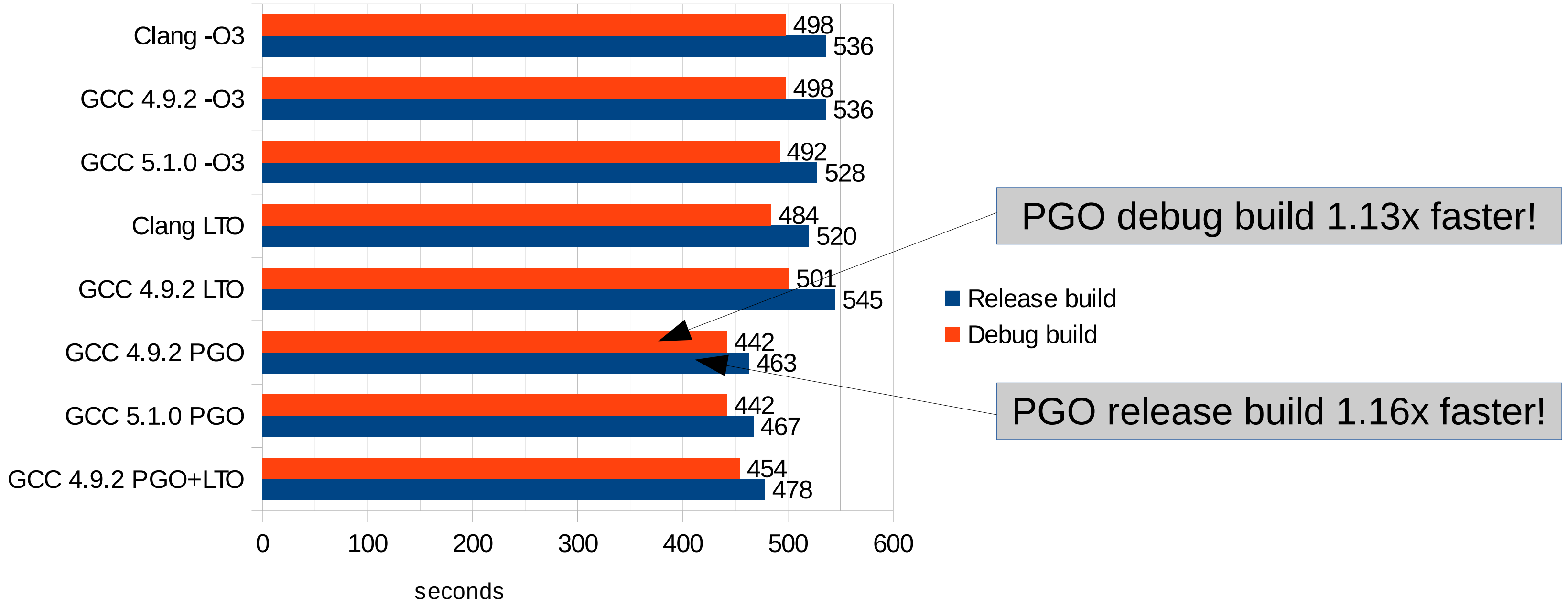
Use a faster linker

- Using GNU gold instead of GNU ld
- Building Clang with GCC 4.9.2 as the host compiler
- Default CMake generator (Makefiles)

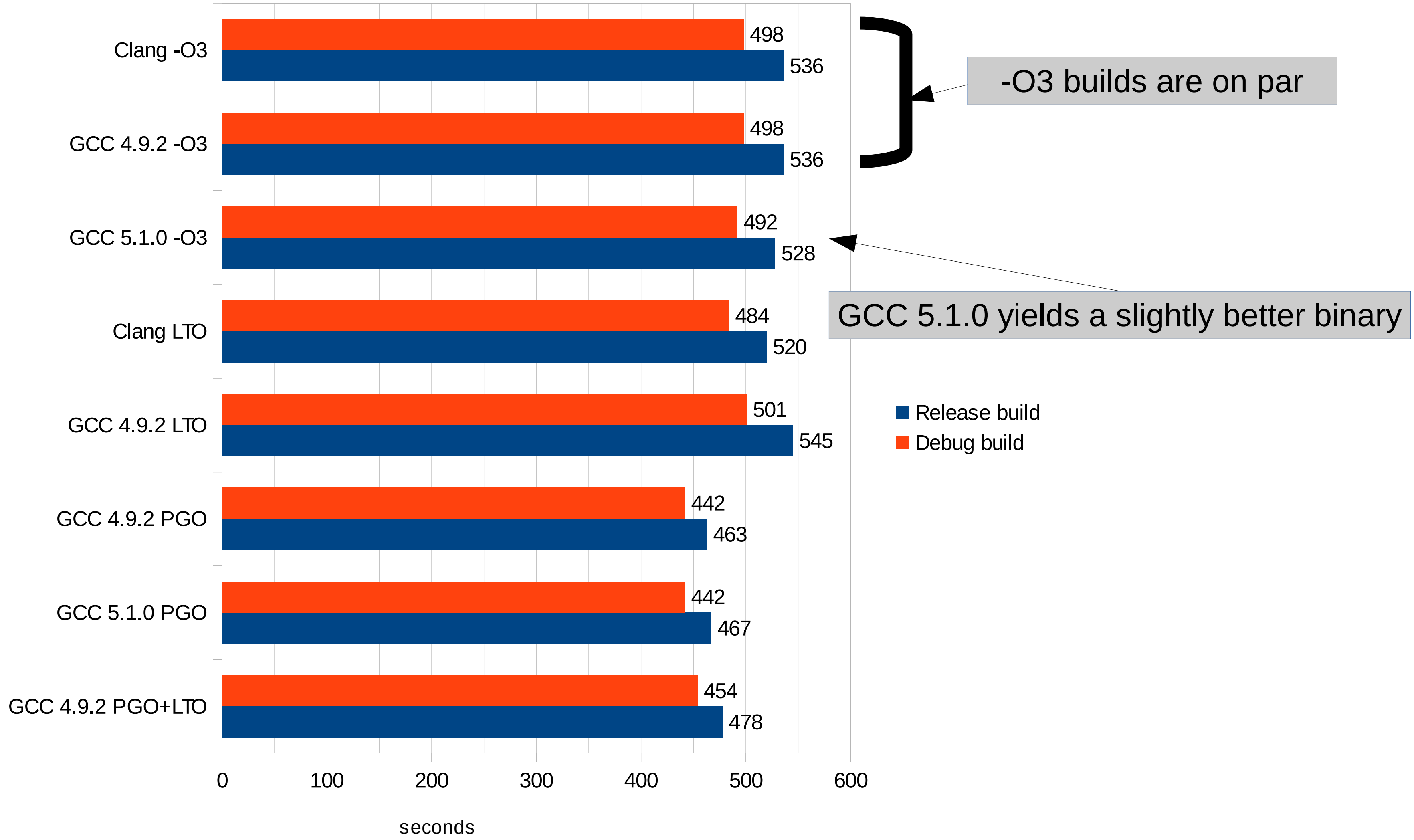


Optimize host compiler binary aggressively

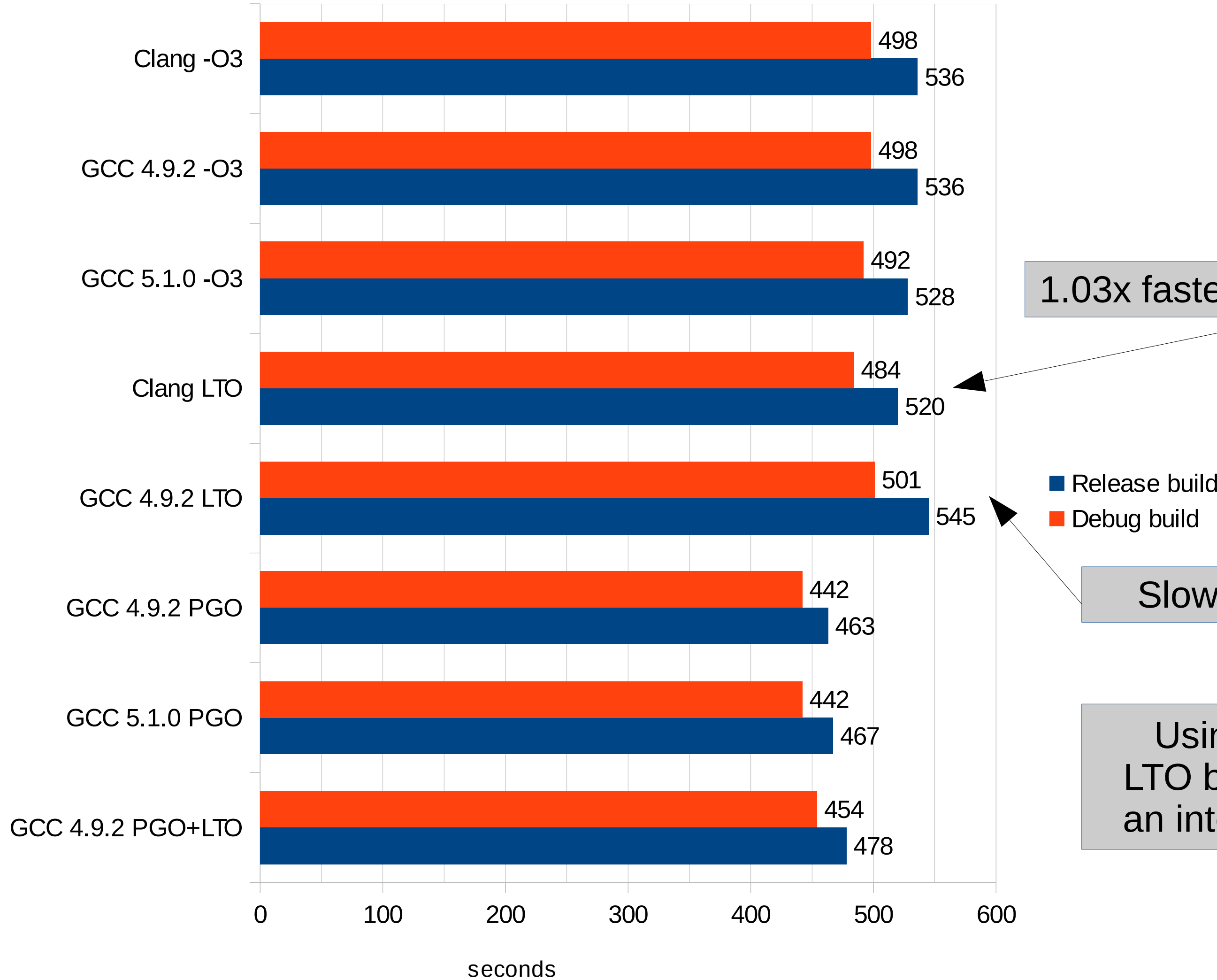
- Building Clang with a heavily optimized host Clang build
- Host Clang was compiled with Clang or GCC at the various different optimization levels



Optimize host compiler binary aggressively



Optimize host compiler binary aggressively



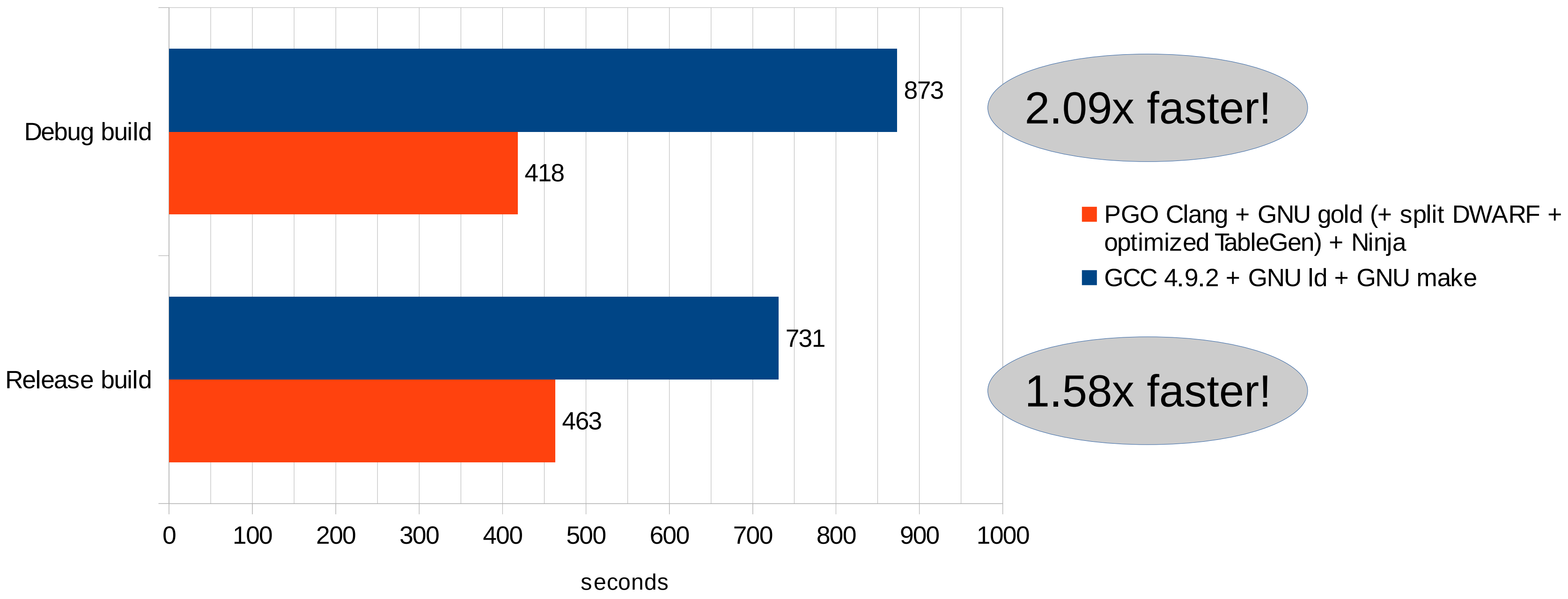
1.03x faster than the -O3 build

Slower than the -O3 build

Using GCC 5.1.0 for an LTO build of Clang leads to an internal compiler error :(

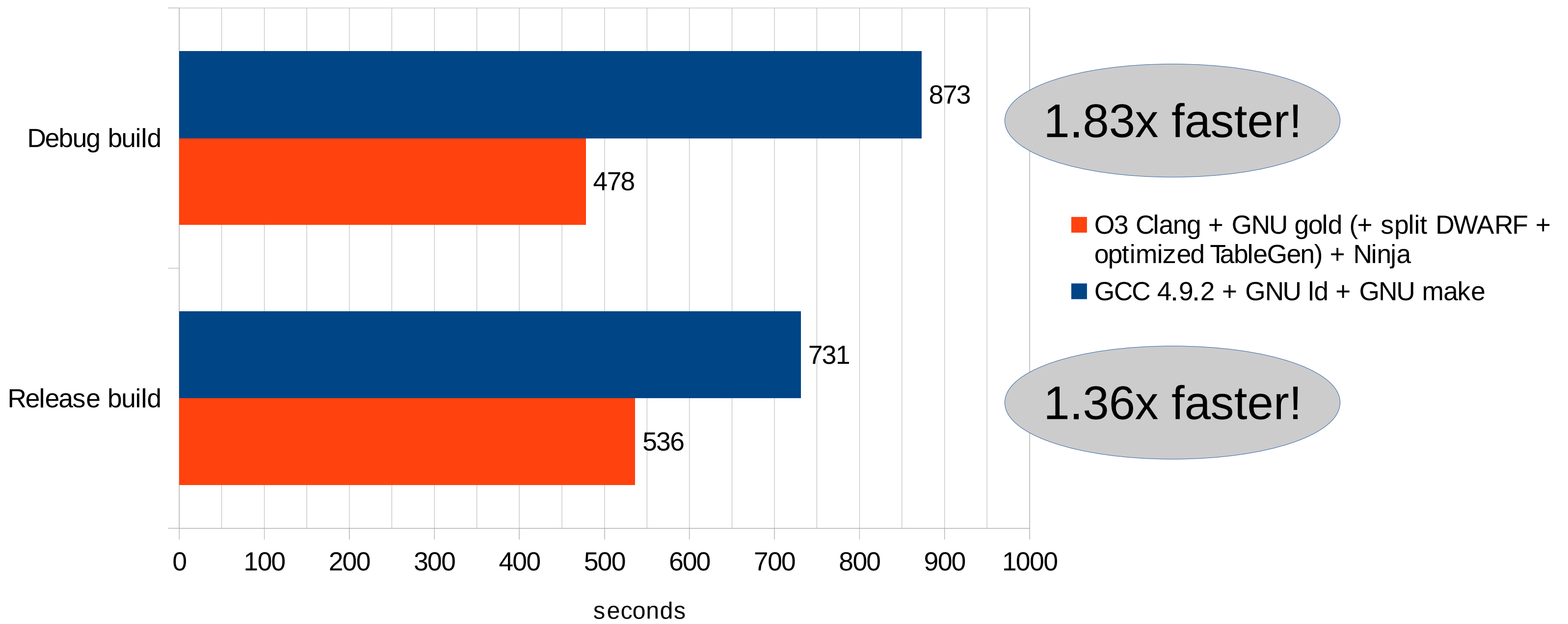
Overall speedup

- Using a PGO build of Clang (built with GCC 4.9.2)



Overall speedup

- Standard -O3 build of Clang



Conclusion

- Always build with Clang if you care about compile time
- Use GNU gold rather than GNU ld
- Building Clang with GCC in PGO mode produces fastest Clang host compiler binary

Summary

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
  - Arch Linux
  - ...and many more

Thank you.

Contact Information:

Tilmann Scheller
t.scheller@samsung.com

Samsung Open Source Group
Samsung Research UK