Linaro connect
San Francisco 2017

Getting Started with LLVM
The TL;DR Version

Diana Picus

Please read the docs!

http://llvm.org/docs/

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

What This Talk Is About

- ~~The LLVM IR(s)~~
- ~~The LLVM codebase~~

- The LLVM community
- Committing patches
- LLVM developer tools
- Writing tests for LLVM

- More discussions: Grand Peninsula C, Wednesday 4PM

Interacting with the Community

- Mailing lists
  - $PROJECT-dev
    - Try to keep an eye on the discussions here
  - $PROJECT-commits
    - Use mail filters
    - Start reviewing patches

$PROJECT ∈ { llvm, cfe, … }

Interacting with the Community

- Mailing lists
- IRC
  - #llvm on oftc
  - #llvm-build

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Interacting with the Community

- Mailing lists
- IRC
- Dev meetings
  - US LLVM
  - EuroLLVM
  - Devrooms and workshops at other conferences
  - YouTube channel
    https://www.youtube.com/channel/UCv2_41bSAa5Y_8BacJUZfjQ

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Interacting with the Community

- Mailing lists
- IRC
- Dev meetings
- Socials
  - https://www.meetup.com/pro/llvm/

Interacting with the Community

- Mailing lists
- IRC
- Dev meetings
- Socials
- Bugzilla
  - https://bugs.llvm.org/

Interacting with the Community

- Mailing lists
- IRC
- Dev meetings
- Socials
- Bugzilla
- Code of conduct
  - https://llvm.org/docs/CodeOfConduct.html

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Submitting a Patch

- https://llvm.org/docs/DeveloperPolicy.html

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Submitting a Patch

- https://llvm.org/docs/DeveloperPolicy.html
- Make sure your patch is based on trunk
- Keep it small
  - Submit unrelated changes as separate patches
  - Try to break your patch into logical chunks

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Submitting a Patch

- https://llvm.org/docs/DeveloperPolicy.html
- Make sure your patch is based on trunk
- Keep it small
- Please have tests!
  - Or mark your patch as NFC

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Submitting a Patch

- https://llvm.org/docs/DeveloperPolicy.html
- Make sure your patch is based on trunk
- Keep it small
- Please have tests!
- Have a descriptive summary
  - Short and catchy title
  - Why is your patch needed?
  - How does your patch work?

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Submitting a Patch

- https://llvm.org/docs/DeveloperPolicy.html
- Make sure your patch is based on trunk
- Keep it small
- Please have tests!
- Have a descriptive summary
- Please follow the coding style
  - clang/tools/clang-format/git-clang-format or clang-format-diff.py
  - Only catches formatting issues, try to internalize the rest
  - https://llvm.org/docs/CodingStandards.html
  - https://llvm.org/docs/ProgrammersManual.html

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Phabricator

- https://llvm.org/docs/Phabricator.html
- Web interface or Arcanist

Phabricator

- https://llvm.org/docs/Phabricator.html
- Web interface or Arcanist
- Full context
  - git diff -U99999 or equivalent

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Phabricator

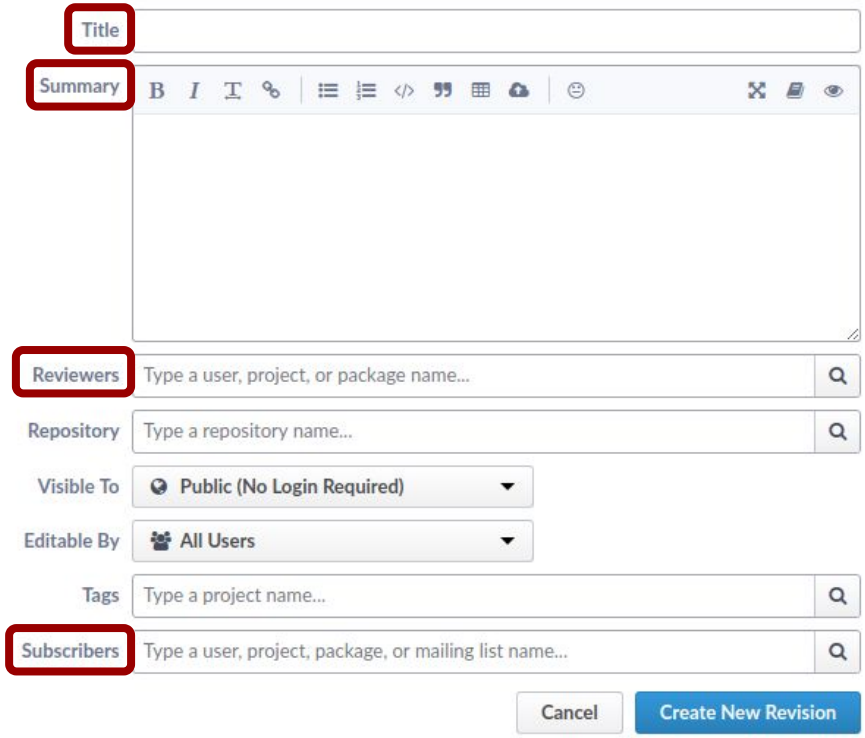
- https://llvm.org/docs/Phabricator.html
- Web interface or Arcanist
- Full context
- Subscribers
  - $PROJECT-commits
  - Please refrain from adding $PROJECT-dev

Phabricator

- https://llvm.org/docs/Phabricator.html
- Web interface or Arcanist
- Full context
- Subscribers
- Reviewers
  - git blame
  - CODE_OWNERS.txt

Phabricator

- https://llvm.org/docs/Phabricator.html
- Web interface or Arcanist
- Full context
- Subscribers
- Reviewers



Start with this!

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Phabricator

- https://llvm.org/docs/Phabricator.html
- Web interface or Arcanist
- Full context
- Subscribers
- Reviewers
- Ping / rebase ~ once/week

Committing a Patch

- Patches need to be accepted
  - When at least one person has accepted a patch, it can be committed
  - Patches can still be reviewed after commit
  - Please address any follow-up comments

Committing a Patch

- Patches need to be accepted
- Committing a patch
  - If you don’t have commit access, ask the person that approved the patch to commit for you
  - After 2-3 patches you can ask for commit access

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Committing a Patch

- Patches need to be accepted
- Committing a patch
- Last line of summary - link to Phabricator review
  - Differential Revision: https://reviews.llvm.org/D01234
  - If you forget to add that, close the Phabricator review manually

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Committing a Patch

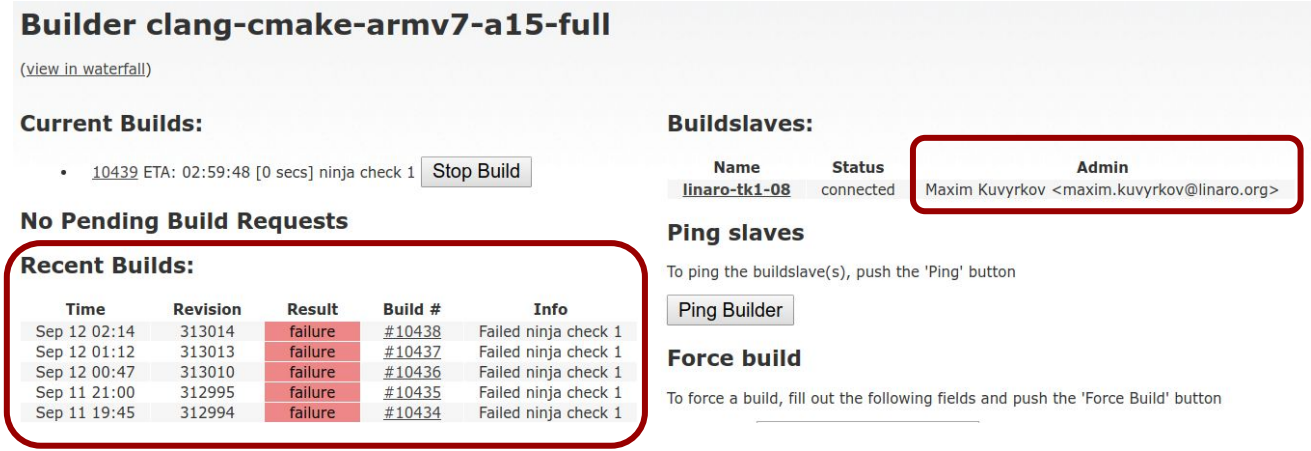
- Patches need to be accepted
- Committing a patch
- Last line of summary - link to Phabricator review
- Please keep an eye on the buildbots

Buildbots

- Different platforms and configurations
  - Hardware, OS, compiler
  - Projects added (compiler-rt, lld, test-suite)
  - Bootstrap / selfhost
- 2-3h for 70-80% of the buildbots
- Blame emails
- IRC chat bots
- Console
  - http://lab.llvm.org:8011/console

Dealing with failures

- Revision ranges
- Ask if unsure!
- Top priority: appeasing the bots
  - Fix or **revert**
  - **Don't be afraid to revert :)**
  - Always mention which revision you're fixing/reverting/reapplying



Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Releases

- 2 major releases / year
- 1-2 minor releases in between
- Avoid disruptive changes around the release branch point


UPlanFL N
4x/0.13
∞/– /FN26.5

Developer Tools

- Not for production use!
- There are several of them
- We'll only cover opt, llc, FileCheck
- https://llvm.org/docs/CommandGuide/
  - -help or -help-hidden is often more useful

opt

- Interfaces with the middle-end

- Can use the old (default) pass manager or the new one
- This talk will focus on the old pass manager interface

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Running passes

```
opt -mem2reg -simplify-cfg in.bc -o out.bc
                                 -S -o out.ll
                                 -S -o -

opt -O0 in.ll -S -o out.ll
    -O1
    -O2
    -O3
    -Os
    -Oz

opt -inline -O1 -instcombine in.ll -o out.bc
```

ENGINEERS
AND DEVICES
WORKING
TOGETHER

What Is Run

```
opt -sroa -debug-pass=Structure in.ll [...]

  Pass Arguments: -targetlibinfo -tti -targetpassconfig
                  -assumption-cache-tracker -domtree -sroa
                  -verify -write-bitcode (or -print-module)
  Target Library Information
  Target Transform Information
  Target Pass Configuration
  Assumption Cache Tracker
    ModulePass Manager
      FunctionPass Manager
        Dominator Tree Construction
        SROA
        Module Verifier
      Bitcode Writer (or Print Module IR)
```

What Is Run

```
opt -O3 -debug-pass=Arguments in.ll [...]
   vs
clang -O3 -mllvm -debug-pass=Arguments [...]
```


Debug dumps

```
opt -O3 -debug-only=licm [...]

  LICM hoisting to for.body.lr.ph:
   %1 = load i32, i32* %x, align 4
  LICM hoisting to for.body.lr.ph:
   %wide.trip.count = zext i32 %n to i64
```

http://llvm.org/docs/ProgrammersManual.html#the-debug-macro-and-debug-option

ENGINEERS
AND DEVICES
WORKING
TOGETHER

ENGINEERS
AND DEVICES
WORKING
TOGETHER

IR dumps

```
opt -O3 -print-before=loop-vectorize [...]
opt -O3 -print-after=loop-vectorize [...]
```

```
*** IR Dump After Loop Vectorization ***
; Function Attrs: norecurse nounwind readonly
define i32 @f(i32 %x) local_unnamed_addr #0 {
entry:
  %x.addr = alloca i32, align 4
  [...]
}
```

ENGINEERS
AND DEVICES
WORKING
TOGETHER

IR dumps

```
opt -O3 -print-before-all
        -filter-print-funcs=f [...]
opt -O3 -print-after-all [...]
```

```
*** IR Dump After Loop Vectorization ***
; Function Attrs: norecurse nounwind readonly
define i32 @f(i32 %x) local_unnamed_addr #0 {
entry:
  [...]
}
*** IR Dump After Canonicalize natural loops ***
; Function Attrs: norecurse nounwind readonly
define i32 @f(i32 %x) local_unnamed_addr #0 {
entry:
  [...]
}
```

Other Features

- Statistics

  http://llvm.org/docs/ProgrammersManual.html#the-statistic-class-stats-option
  - opt -stats [...]
- Debug counters

  http://llvm.org/docs/ProgrammersManual.html#adding-debug-counters-to-aid-in-debugging-your-code
  - opt --debug-counter=my-counter-skip=1,my-counter-count=3 [...]
- Dot graphs

  http://llvm.org/docs/ProgrammersManual.html#viewing-graphs-while-debugging-code
  - opt -dot-cfg [...] & related flags
  - Function::viewGraph() from a debugger
- Timers http://llvm.org/doxygen/classllvm_1_1Timer.html
  - opt -time-passes [...]

llc

- Interfaces with the backend

- Some of the flags from opt still work
  - E.g. -debug-pass, -debug-only, -print-after-all etc
- Running passes is slightly different
  - `llc -run-pass=machine-scheduler [...]`
- `Partial pipelines`
  - `llc -O3 -stop-before=machine-scheduler [...]`
  - `llc -O3 -start-after=machine-scheduler [...]`
- Caveat: not all passes are registered

Case Study: Instruction selection

- Global ISel - new, still in development
  - Organized as passes, very llc friendly
- DAG ISel - old, default framework
  - Uses its own representation, different from IR and MIR (SelectionDAG)
  - Has some internal steps, but they are not proper passes
    - => special debug flags
  - All the info: llc -debug-only=isel
  - SelectionDAGs after various steps:
    - -view-dag-combine1-dags
    - -view-legalize-types-dags
    - -view-legalize-dags
    - -view-dag-combine2-dags
    - -view-dag-combine-lt-dags
    - -view-isel-dags
    - -view-sched-dags
    - -view-sunit-dags
    - Can be filtered by basic block with -filter-view-dags
  - See SelectionDAGISel::CodeGenAndEmitDAG

Beware of the Flags!

- https://llvm.org/docs/CommandLine.html
- Decentralized system for adding flags
  - Each implementation file can add its own custom flags
- Flags may be accepted but (silently) ignored
  - clang -mllvm -stop-before=machine-scheduler 👍 Actually stops
  - clang -mllvm -stop-before=loop-vectorize 👎 Doesn't stop


Testing LLVM

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Testing LLVM

- https://llvm.org/docs/TestingGuide.html
- Regression tests
  - check-all
  - Should be run before every commit (on a Release build)
  - New functionality => new regression test

Testing LLVM

- https://llvm.org/docs/TestingGuide.html
- Regression tests
- Test-suite
  - Separate repo
  - Lots of applications for testing correctness
  - Some benchmarks
  - Can skip for simple commits

Regression tests

- $PROJECT/unittests
  - Use these for testing APIs
  - Google Test and Google Mock
  - Target for all tests: check-$PROJECT-unit
  - Targets for specific tests: grep for add_$PROJECT_unittest

$PROJECT ∈ { llvm, clang, … }

Regression tests

- $PROJECT/unittests
- $PROJECT/test
  - Use these for everything else
  - opt, llc, llvm-mc etc
  - FileCheck
  - Target for all tests: check-$PROJECT
  - Target for specific tests: check-path-to-test-in-lowercase
    - llvm/test/Transforms/EarlyCSE/AArch64/*.ll
    - check-llvm-transforms-earlycse-aarch64
  - Can also run one test or all tests in one directory with LIT
    - llvm-lit llvm/test/Transforms/EarlyCSE/AArch64/intrinsics.ll
    - llvm-lit llvm/test/Transforms/EarlyCSE

$PROJECT ∈ { llvm, clang, … }

Adding an IR Test

- The easy way: Copy and modify an existing test
  - ...but not a very old one
- The long way:
  - Step 1: Get a clean IR module
  - Step 2: Add RUN lines
  - Step 3: Add CHECK lines

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an IR Test

- The long way: Get a starting point in plain C

```
void f(int *x, int *y,
       int * restrict z, int n) {
  for (int i = 0; i < n; i++)
    z[i] = x[i] + y[i];
}
```

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an IR Test

`clang -O0 -S -emit-llvm vect.c -o vect.ll`

```
; ModuleID = 'vect.c'
source_filename = "vect.c"
target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32
target triple = "aarch64--linux"

; Function Attrs: noinline nounwind optnone
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) #0 {
entry:
  %x.addr = alloca i32*, align 8
  %y.addr = alloca i32*, align 8
  ret void
}

attributes #0 = { noinline nounwind optnone "correctly-rounded-div

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 6.0.0 "}
```

Adding an IR Test

```
clang -O0 -S -emit-llvm vect.c -o vect.ll
```

```
; ModuleID = 'vect.c'
source_filename = "vect.c"
target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32
target triple = "aarch64--linux"

; Function Attrs: noinline nounwind optnone
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) #0 {
entry:
  %x.addr = alloca i32*, align 8
  %y.addr = alloca i32*, align 8
  ret void
}

attributes #0 = { noinline nounwind optnone "correctly-rounded-div

!llvm.module.flags = !{!0}
!llvm.ident = !{!1}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{!"clang version 6.0.0 "}
```

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an IR Test

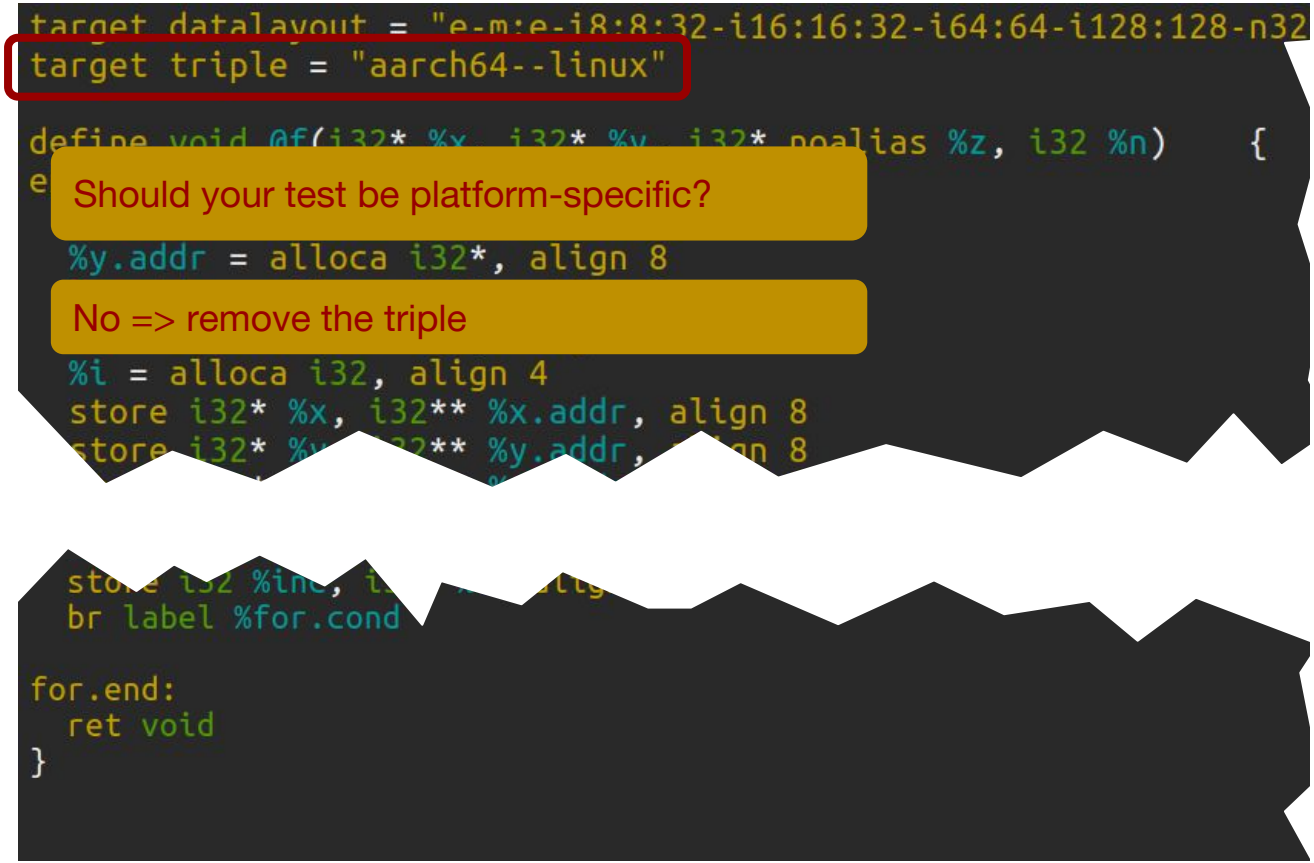
```
target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32
target triple = "aarch64--linux"

define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) #0 {
entry:
  %x.addr = alloca i32*, align 8
  %y.addr = alloca i32*, align 8
  %z.addr = alloca i32*, align 8
  %n.addr = alloca i32, align 4
  %i = alloca i32, align 4
  store i32* %x, i32** %x.addr, align 8
  store i32* %y, i32** %y.addr, align 8
```

```
  br label %for.cond

for.end:
  ret void
}

attributes #0 = { noinline nounwind optnone "correctly-rounded-divid
```

!!!

Adding an IR Test

Adding an IR Test



Adding an IR Test

Should your test be platform-specific?

Yes => put the test in a target dir
E.g. test/Transforms/LoopVectorize/**AArch64**
If you have to create the dir, make sure you copy and update a **lit.local.cfg** file

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an IR Test

```
target datalayout = "e-m:e-i8:8:32-i16:16:32-i64:64-i128:128-n32
target triple = "aarch64--linux"

d                                                    ias %z, i32 %n)    {
e
  %y.addr = alloca i32*, align 8
  %z.addr = alloca i32*, align 8
  %n.addr = alloca i32, align 4
  %i = alloca i32, align 4
  store i32* %x, i32** %x.addr, align 8
  store i32* %                 32** %y.addr,          n 8
```

Should your test care about type alignments, native type sizes, endianness?

```
  br label %for.cond

for.end:
  ret void
}
```

Adding an IR Test

```
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n)    {
entry:
  %x.addr = alloca i32*, align 8
  %y.addr = alloca i32*, align 8
  %z.addr = alloca i32*, align 8
  %n.addr = alloca i32, align 4
  %i = alloca i32, align 4
  store i32* %x, i32** %x.addr, align 8
  store i32* %y, i32** %y.addr, align 8
  store i32* %z, i32** %z.addr, align 8
```

opt can clean these up for you

```
for.inc:
  %10 = load i32, i32* %i, align 4
  %inc = add nsw i32 %10, 1
  store i32 %inc, i32* %i, align 4
  br label %for.cond

for.end:
  ret void
}
```

Adding an IR Test

`opt -mem2reg vect.ll -S -o vect.ll`

```
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
entry:
  br label %for.cond

for.cond:
  %i.0 = phi i32 [ 0, %e
  %cmp = icmp slt i32 %i.
  br i1 %cmp, label %for

for.body:
  %idxprom = sext i32 %i
  %arrayidx = getelement
  %0 = load i32, i32* %arrayidx, align 4
  %idxprom1 = sext i32 %i.0 to i64
  %arrayidx2 = getelementptr inbounds i32, i32* %y, i64 %idxprom1
  %1 = load i32, i32* %arrayidx2, align 4
  %add = add nsw i32 %0, %1
  %idxprom3 = sext i32 %i.0 to i64
  %arrayidx4 =
```

Nice and clean.
You may still want to run other passes to further canonicalize / clean up the code.
E.g. -loop-rotate, -simplifycfg, -instcombine

Alternatively, you can use -print-before on a -O3 run, or edit the file manually.
Run -verify on it before proceeding to make sure the IR is well formed.

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
entr
  %c
  br i1 %cmp1, label %for.body.lr.ph, label %for.end

for.body.lr.ph:
  br label %for.body

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
  %idxprom1 = sext i32 %i.02 to i64
  %arrayidx2 = getelementptr inbounds i32, i32* %y, i64 %idxprom1
  %1 = load i32, i32* %arrayidx2, align 4
  %add = add nsw i32 %0, %1
  %idxprom3 = sext i32 %i.02 to i64
  %arrayidx4 = getelementptr inbounds i32, i32* %z, i64 %idxprom3
  store i32 %add, i32* %arrayidx4, align 4
```

Add one or more RUN lines

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
ent
                                                  label %for.end
fo

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
  %idxprom1 = sext i32 %i.02 to i64
  %arrayidx2 = getelementptr inbounds i32, i32* %y, i64 %idxprom1
  %1 = load i32, i32* %arrayidx2, align 4
  %add = add nsw i32 %0, %1
  %idxprom3 = sext i32 %i.02 to i64
  %arrayidx4 = getelementptr inbounds i32, i32* %z, i64 %idxprom3
  store i32 %add, i32* %arrayidx4, align 4
```

%s = the current file
%t = a temporary file (won't conflict with temporaries for other tests)

See docs for other substitutions

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
entry
  %ci
  br                                    r.ph, label %for.end

for.body.lr.ph:
  br label %for.body

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
  %idxprom1 = sext i32 %i.02 to i64
  %arrayidx2 = getelementptr inbounds i32, i32* %y, i64 %idxprom1
  %1 = load i32, i32* %arrayidx2, align 4
  %add = add nsw i32 %0, %1
  %idxprom3 = sext i32 %i.02 to i64
  %arrayidx4 = getelementptr inbounds i32, i32* %z, i64 %idxprom3
  store i32 %add, i32* %arrayidx4, align 4
```

Feeds the input through stdin
=> no path in the ModuleID
=> fewer spurious matches

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;      | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
;      | FileCheck %s --check-prefixes=CHECK,WIDTH8
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
; CHECK-LABEL: @f
entry:
  %cmp1 = icmp slt i32 0, %
  br i1 %cmp1, label %for.body.lr.ph, label %for.end

for.body.lr.ph:
  br label %for.body

  ; WIDTH4: [[NMODVF:%.*]] = urem i32 %n, 4
  ; WIDTH8: [[NMODVF:%.*]] = urem i32 %n, 8
  ; CHECK: {{%.*}} = sub i32 %n, [[NMODVF]]

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
```

This is what actually checks the results.

Linaro connect
San Francisco 2017

ENGINEERS AND DEVICES WORKING TOGETHER

Linaro connect
San Francisco 2017

ENGINEERS AND DEVICES WORKING TOGETHER

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH8
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
; CHECK-LABEL:
entry:
  %cmp1 = ic
  br i1 %cmp1, label %for.body.lr.ph, label %for.end

for.body.lr.ph:
  br label %for.body

  ; WIDTH4: [[NMODVF:%.*]] = urem i32 %n, 4
  ; WIDTH8: [[NMODVF:%.*]] = urem i32 %n, 8
  ; CHECK: {{%.*}} = sub i32 %n, [[NMODVF]]

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
```

grep-like tool adapted to LLVM's testing needs

Linaro connect
San Francisco 2017

ENGINEERS AND DEVICES WORKING TOGETHER

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH8
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
  br i1 %cmp1, label %for.body.lr.ph, label %for.end

for.body.lr.ph:
  br label %for.body

  ; WIDTH4: [[NMODVF:%.*]] = urem i32 %n, 4
  ; WIDTH8: [[NMODVF:%.*]] = urem i32 %n, 8
  ; CHECK: {{%.*}} = sub i32 %n, [[NMODVF]]

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
```

Read the test output from stdin

Read the test reference output from the current file

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
```

```
vector.ph:
  %n.mod.vf = urem i32 %n, 4
  %n.vec = sub i32 %n, %n.mod.vf
  br label %vector.body
```

Expected output for a vector width of 4

```
  br

; WIDTH4: [[NMODVF:%.*]] = urem i32 %n, 4
; WIDTH8: [[NMODVF:%.*]] = urem i32 %n, 8
; CHECK: {{%.*}} = sub i32 %n, [[NMODVF]]

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
```

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
```

```
vector.ph:
  %n.mod.vf = urem i32 %n, 4
  %n.vec = sub i32 %n, %n.mod.vf
  br label %vector.body
```

Expected output for a vector width of 4

Actual name relevant for the test
=> Should be used in the checks

```
  br

  ; WIDTH4: [[NMODVF:%.*]] = urem i32 %n, 4
  ; WIDTH8: [[NMODVF:%.*]] = urem i32 %n, 8
  ; CHECK: {{%.*}} = sub i32 %n, [[NMODVF]]

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
```

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
```

```
vector.ph:
  %n.mod.vf = urem i32 %n, 4
  %n.vec = sub i32 %n, %n.mod.vf
```

Expected output for a vector width of 4

Actual name **not** relevant for the test
=> Better to use a regex

```
; WIDTH4: [[NMODVF:%.*]] = urem i32 %n, 4
; WIDTH8: [[NMODVF:%.*]] = urem i32 %n, 8
; CHECK: {{%.*}} = sub i32 %n, [[NMODVF]]

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
```

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;       | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
```

```
vector.ph:
  %n.mod.vf = urem i32 %n, 4
  %n.vec = sub i32 %n, %n.mod.vf
  br label %vector.body
```

Expected output for a vector width of 4

```
  br

  ; WIDTH4: [[NM                    i32 %n, 4
  ; WIDTH8: [[NM                    i32 %n, 8
  ; CHECK: {{%.*}} = sub i32 %n, [[NMODVF]]

for.body:
  %i.02 = ph                                  %inc, %for.inc ]
  %idxprom =
  %arrayidx                                   * %x, i64 %idxprom
  %0 = load i32                 * %arrayidx, align 4
```

Unnamed regex

Alternative check if we don't care about the assignment:
; CHECK: sub i32 %n, [[NMODVF]]

ENGINEERS AND DEVICES WORKING TOGETHER

Adding an IR Test

```
; RUN: opt < %s -S -loop-vectorize -force-vector-width=4 \
;      | FileCheck %s --check-prefixes=CHECK,WIDTH4
; RUN: opt < %s -S -loop-vectorize -force-vector-width=8 \
;      | FileCheck %s --check-prefixes=CHECK,WIDTH8
define void @f(i32* %x, i32* %y, i32* noalias %z, i32 %n) {
; CHECK-LABEL: @f
entry:
  %cm
  br                                          %for.end

for.b
  br label %for.body

  ; W
  ; W
  ; C

for.body:
  %i.02 = phi i32 [ 0, %for.body.lr.ph ], [ %inc, %for.inc ]
  %idxprom = sext i32 %i.02 to i64
  %arrayidx = getelementptr inbounds i32, i32* %x, i64 %idxprom
  %0 = load i32, i32* %arrayidx, align 4
```

Helps avoid spurious matches by dividing the file into blocks. Check lines can only match output in the current block.

Protip: Don't name your functions @f :) Label names should only match in one place in the test output. Pick something suggestive (test-vect-add, test-noalias).

ENGINEERS AND DEVICES WORKING TOGETHER

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Other FileCheck Features

- https://llvm.org/docs/CommandGuide/FileCheck.html
- CHECK-NEXT
- CHECK-SAME
- CHECK-DAG

Other LIT Features

- https://llvm.org/docs/CommandGuide/lit.html
- LIT_USE_INTERNAL_SHELL
  - Makes it easier to know which RUN line in a test failed
- REQUIRES, UNSUPPORTED, XFAIL

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an MIR Test

- Same principles apply
- Fortunately, you can use -stop-before/after from clang
  - clang -mllvm -stop-before=expand-isel-pseudos -o test.mir [...]
- Unfortunately, that will produce A LOT of output

Adding an MIR Test

- Same principles apply
- Fortunately, you can use -stop-before/after from clang
  - clang -mllvm -stop-before=expand-isel-pseudos -o test.mir [...]
- Unfortunately, that will produce A LOT of output
- IR part
  - In some cases you may be able to remove the whole IR section
  - Otherwise, clean up as you would for opt
  - Replace function bodies with { ret void }

Linaro
connect
San Francisco 2017

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an MIR Test

- Same principles apply
- Fortunately, you can use -stop-before/after from clang
  - clang -mllvm -stop-before=expand-isel-pseudos -o test.mir [...]
- Unfortunately, that will produce A LOT of output
- IR part
- MIR part
  - Remove any irrelevant function properties
    - alignment? exposesReturnsTwice? frameInfo? fixedStack? etc etc
  - Remove basic block successors (if irrelevant)
    - successors: %bb.6.middle(0x04000000), %bb.5.vector.body(0x7c000000)

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Adding an MIR Test

- Same principles apply
- Fortunately, you can use -stop-before/after from clang
  - clang -mllvm -stop-before=expand-isel-pseudos -o test.mir [...]
- Unfortunately, that will produce A LOT of output
- IR part
- MIR part
- llc caveat
  - Unlike opt, llc produces an output file by default
  - That file will remain in the test directory
  - Next test run => spurious test without RUN lines => buildbot failure
  - Avoid by redirecting the output to stdout, /dev/null or %t

One Last Thing about Debugging...

- Many objects in LLVM have a dump() method
- You can use this in a debugger or as part of your debug dumps

Linaro connect
San Francisco 2017

Thank You

#SFO17

SFO17 keynotes and videos on: connect.linaro.org

For further information: www.linaro.org

Images from www.pexels.com