THE ATTRIBUTOR: A VERSATILE INTER-PROCEDURAL
FIXPOINT ITERATION FRAMEWORK

LLVM-Dev’19 — October 22, 2019 — San Jose, CA, USA

Johannes Doerfert*, Hideto Ueno, Stefan Stipanovic

*Leadership Computing Facility
Argonne National Laboratory
https://www.alcf.anl.gov/
Two of the authors were supported by Google Summer of Code (GSoC)!

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation’s exascale computing imperative.
I. BACKGROUND
int * checkAndAdvance(int * __attribute__((aligned(16))) p) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}

What is the alignment of:

(1) the return type?
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

(1) the return type? (2) the returned value?
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

1. the `return type`?
2. the `returned value`?
3. the `argument`?
What is the alignment of:

(1) the **return type**? (2) the **returned value**? (3) the **argument**?
(4) the **returned value**?
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

1. the return type?
2. the returned value?
3. the argument?
4. the returned value?
5. the return type?
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4) ;
    return p ;
}
```

What is the alignment of:

(1, ∞)

(1) the `return type`? (2) the `returned value`? (3) the `argument`?
(4) the `returned value`? (5) the `return type`?
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
  if (*p == 0)
    return checkAndAdvance(p + 4);
  return p;
}
```

What is the alignment of:

1. the return type?
2. the returned value?
3. the argument?
4. the returned value?
5. the return type?

(1, \infty) (1, \infty) (16, 16) (1, \infty) (1, 16)
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

1. the return type?  
   - (1, ∞)
2. the returned value?  
   - (1, ∞)
3. the argument?  
   - (16, 16)
4. the returned value?  
   - (16, 16)
5. the return type?  
   - (1, 16)
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}

What is the alignment of:

(1, ∞)    (16, 16) ←-------------- (16, 16)
(1) the return type? (2) the returned value? (3) the argument?
(4) the returned value? (5) the return type?
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

(1) the return type?  (2) the returned value?  (3) the argument?
(4) the returned value?  (5) the return type?
(1, ∞)  (16, 16) ←------------ (16, 16)
(1, ∞)  (16, 16)
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
  if (*p == 0)
    return checkAndAdvance(p + 4);
  return p;
}
```

What is the alignment of:

1. the `return` type?
2. the returned value?
3. the argument?
4. the returned value?
5. the `return` type?

(1, ∞)  (16, 16) ← -------------- (16, 16)  
(1, ∞)  (16, 16)  
(1, ∞)  (1, ∞)
```
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

1. the return type? (1, ∞)
2. the returned value? (16, 16) ← ------------ (16, 16)
3. the argument? (16, 16)
4. the returned value? (1, ∞) ← ------------ (1, ∞)
5. the return type? (1, ∞) ← ------------ (1, ∞)
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

1. The return type?
2. The returned value?
3. The argument?
4. The returned value?
5. The return type?

```
(1, 16) ← ⊙ ---------- (16, 16) ← ---------- (16, 16)
(1, ∞) ← ----------------- (1, ∞)
```
int * checkAndAdvance(int * __attribute__((aligned(16))) p) {
    if (*p == 0) {
        return checkAndAdvance(p + 4);
    }
    return p;
}

What is the alignment of:

(1, 16)          (16, 16)
16, 16          16, 16

(1) the return type? (2) the returned value? (3) the argument?
(4) the returned value? (5) the return type?

(1, ∞)          (1, 16)
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

1. the **return type**?
   - (1, 16)
2. the **returned value**?
   - (16, 16) ← (16, 16)
3. the **argument**?
   - (1, 16) ← (1, 16)
4. the **returned value**?
   - (1, 16) ← (1, 16)
5. the **return type**?
   - (16, 16) ← (16, 16)
```c
int * checkAndAdvance(int * __attribute__((aligned(16))) p) {
    if (*p == 0)
        return checkAndAdvance(p + 4);  // (16, 16) ← ⋄ 16, 16 ← 16, 16 ← 16, 16
    return p;  // (1, 16) ← ⋄ 16, 16 ← 16, 16 ← 16, 16
}
```

What is the alignment of:

1. The return type?
2. The returned value?
3. The argument?
4. The returned value?
5. The return type?

```
(1, 16) ← ⋄ 16, 16 ← 16, 16 ← 16, 16
```

⊙ 2/16
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```

What is the alignment of:

1. The return type?
2. The returned value?
3. The argument?
4. The returned value?
5. The return type?

ABSTRACT STATES

Assumed

Known
Abstract States

Assumed

Known

Fixpoint states

Assumed == Known
Abstract States

Assumed

indicate **pessimistic** fixpoint

Known

indicate **optimistic** fixpoint

fixpoint states

Know == Assumed

Assumed == Known

Assumed == Known
```c
int * checkAndAdvance( int * __attribute__((aligned(16))) p ) {
    if (*p == 0)
        return checkAndAdvance(p + 4);
    return p;
}
```
Attributor A;

// Select what information is to be deduced.
IRPosition IRPRRet = IRPosition::returned(Fn);
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRRet);

// Deduce information and manifest it in the IR.
auto Changed = A.run(*Fn->getParent());
Attributor A;

// Select what information is to be deduced.
IRPosition IRPRet = IRPosition::returned(Fn);
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);

// Deduce information and manifest it in the IR.
auto Changed = A.run(*Fn->getParent());
Attributor A;

// Select what information is to be deduced.
IRPosition IRPRet = IRPosition::returned(Fn);
const auto &AA = A.getOrCreateAAFor<AAAlign>(IRPRet);

// Deduce information and manifest it in the IR.
auto Changed = A.run(*Fn->getParent());
// Restrict deduction to specific abstract attributes.
auto Whitelist = {&AAAlign::ID};

Attributor A(Whitelist);

// Select what information is to be deduced.
IRPosition IRPRet = IRPosition::returned(Fn);
const auto &AA = A.getOrCreateAAFor<AAAlign>(IRPRet);

// Deduce information and manifest it in the IR.
auto Changed = A.run(*Fn->getParent());
// Restrict deduction to specific abstract attributes.
auto Whitelist = {{&AAAlign::ID,
   /* Think IP-SCCP */ &AAIsDead::ID, &AAValueSimplify::ID}};

Attributor A(Whitelist);

// Select what information is to be deduced.
IRPosition IRPRet = IRPosition::returned(Fn);
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);

// Deduce information and manifest it in the IR.
auto Changed = A.run(*Fn->getParent());
// Restrict deduction to specific abstract attributes.
auto Whitelist = {
    &AAAlign::ID,
    /* Think IP-SCCP */ &AAIsDead::ID, &AAValueSimplify::ID
};

Attributor A(Whitelist);

// Select what information is to be deduced.
IRPosition IRPRet = IRPosition::returned(Fn);
const auto &AA = A.getOrCreateAAFor< AAAlign >(IRPRet);

// Deduce information and manifest it in the IR.
auto Changed = A.run(*Fn->getParent());
THE ATTRIBUTOR — WHAT IT IS

- easy way to perform fixpoint analyses
- dependence tracking, work list algorithm, timeouts, …
- powerful way to perform fixpoint analyses
- utilize concurrently deduced information, e.g., liveness
- alternative to inlining
- IPO + internalization + function rewriting, e.g., argument promotion
- easy way to perform fixpoint analyses
  dependence tracking, work list algorithm, timeouts, ...
- easy way to perform fixpoint analyses
  dependence tracking, work list algorithm, timeouts, ...

- powerful way to perform fixpoint analyses
  utilize concurrently deduced information, e.g., liveness
- easy way to perform fixpoint analyses
  dependence tracking, work list algorithm, timeouts, ...

- powerful way to perform fixpoint analyses
  utilize concurrently deduced information, e.g., liveness

- alternative to inlining
  IPO + internalization + function rewriting, e.g., argument promotion
- easy way to perform fixpoint analyses
  dependence tracking, work list algorithm, timeouts, ...

- powerful way to perform fixpoint analyses
  utilize concurrently deduced information, e.g., liveness

- alternative to inlining
  IPO + internalization + function rewriting, e.g., argument promotion

All good, but why?
II. Motivation
THE ATTRIBUTOR — THE WHY IPO?

Inlining has limits:

- recursion ≡ loops
- code size
- parallelism (think pthread_create)

(declarations) ⇒
inlining has limits:
inlining has limits:

- recursion
inlining has limits:

- recursion ≡ loops
Inlining has limits:

- recursion \equiv loops
- code size
inlining has limits:

• recursion \equiv loops

• code size

• parallelism (think `pthread_create`)
inlining has limits:

- recursion ≡ loops
- code size
- parallelism (think `pthread_create`) ↑
- (declarations) →
THE ATTRIBUTOR — WHY A FRAMEWORK?
THE ATTRIBUTOR — WHY A FRAMEWORK?
THE ATTRIBUTOR — WHY A FRAMEWORK?
III. Design
define i32* @f(i32* %argument) #0 {
    %callsite-returned = tail call i32* @g(i32* %argument) #1
    %flt = getelementptr inbounds i32, i32* %callsite-returned, i64 1
    ret i32* %flt
}

return value
argument
function
call site argument
call site

floating

call site return value
ChangeStatus updateImpl(Attribute &A) override {

}
updateImpl(Attributor &A) override {
    Optional<Value *> Before = getAssumedSimplifiedValue();
    Optional<Value *> After = getAssumedSimplifiedValue();
    if (Before == After)
        return ChangeStatus::UNCHANGED;
    return ChangeStatus::CHANGED;
}
ChangeStatus updateImpl(Attributor &A) override {
  Optional<Value *> Before = getAssumedSimplifiedValue();

  auto Pred = [&](Instruction &I) {
  
  };  

  if (!A.checkForAllInstructions(Pred, this, {Instruction::Ret}))
    return indicatePessimisticFixpoint();

  Optional<Value *> After = getAssumedSimplifiedValue();
  if (Before == After)
    return ChangeStatus::UNCHANGED;
  return ChangeStatus::CHANGED;
}
ChangeStatus updateImpl(Attributor &A) override {
    Optional<Value *> Before = getAssumedSimplifiedValue();

    auto Pred = [&] (Instruction &I) {
        A.getAAFor<AAValueSimplify>(this, I.getOperand(0));
    };
    if (!A.checkForAllInstructions(Pred, this, {Instruction::Ret}))
        return indicatePessimisticFixpoint();

    Optional<Value *> After = getAssumedSimplifiedValue();
    if (Before == After)
        return ChangeStatus::UNCHANGED;
    return ChangeStatus::CHANGED;
}
ChangeStatus updateImpl(Attributor &A) override {
    Optional<Value *> Before = getAssumedSimplifiedValue();

    auto Pred = [&](Instruction &I) {
        return combine(A.getAAFor<AAValueSimplify>(this, I.getOperand(0)));}
    if (!A.checkForAllInstructions(Pred, this, {Instruction::Ret}))
        return indicatePessimisticFixpoint();

    Optional<Value *> After = getAssumedSimplifiedValue();
    if (Before == After)
        return ChangeStatus::UNCHANGED;
    return ChangeStatus::CHANGED;
}
nofree
NEW ATTRIBUTES

nonsync
will return
derefereceable_globally
liveness
returned values
value simplify
heap-2-stack
pointer privatization
The Attributor — Challenges

when to specialize for call sites
≡ "inlining + outlining"
when to specialize for call sites
(≡ “inlining + outlining”)
how to seed abstract attributes (heuristics, pgo-based, ...)

THE ATTRIBUTOR — CHALLENGES
reduce overheads
combine deduction schemes, e.g., context-based & def-use-based
### Evaluation — FunctionAttrs (Late) vs. Attributor (Early)

<table>
<thead>
<tr>
<th>loc.</th>
<th>attribute</th>
<th># w/o A.</th>
<th># w/ A.</th>
<th>A. $\Delta$</th>
<th>tot. w/o A.</th>
<th>tot. w/ A.</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn.</td>
<td>nosync</td>
<td>0</td>
<td>7612</td>
<td>0.0%</td>
<td>4.36%</td>
<td></td>
</tr>
</tbody>
</table>
## Evaluation — FunctionAttrs (Late) vs. Attributor (Early)

<table>
<thead>
<tr>
<th>loc.</th>
<th>attribute</th>
<th># w/o A.</th>
<th># w/ A.</th>
<th>A. Δ</th>
<th>tot. w/o A.</th>
<th>tot. w/ A.</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn.</td>
<td>nosync</td>
<td>0</td>
<td>7612</td>
<td></td>
<td>0.0%</td>
<td>4.36%</td>
</tr>
<tr>
<td>arg. dereferenceable</td>
<td>61825</td>
<td>66317</td>
<td></td>
<td>+7.27%</td>
<td>35.4%</td>
<td>38.0%</td>
</tr>
<tr>
<td>loc.</td>
<td>attribute</td>
<td># w/o A.</td>
<td># w/ A.</td>
<td>A. Δ</td>
<td>tot. w/o A.</td>
<td>tot. w/ A.</td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>----------</td>
<td>---------</td>
<td>---------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>fn.</td>
<td>nosync</td>
<td>0</td>
<td>7612</td>
<td></td>
<td>0.0%</td>
<td>4.36%</td>
</tr>
<tr>
<td>arg.</td>
<td>dereferenceable</td>
<td>61825</td>
<td>66317</td>
<td>+7.27%</td>
<td>35.4%</td>
<td>38.0%</td>
</tr>
<tr>
<td>fn.</td>
<td>nofree</td>
<td>5762</td>
<td>10188</td>
<td>+76.81%</td>
<td>3.3%</td>
<td>5.83%</td>
</tr>
<tr>
<td>fn.</td>
<td>willreturn</td>
<td>0</td>
<td>4146</td>
<td></td>
<td>0.0%</td>
<td>2.37%</td>
</tr>
<tr>
<td>arg.</td>
<td>writeonly</td>
<td>0</td>
<td>3562</td>
<td></td>
<td>0.0%</td>
<td>2.04%</td>
</tr>
<tr>
<td>arg.</td>
<td>readnone</td>
<td>5377</td>
<td>6040</td>
<td>+12.33%</td>
<td>3.08%</td>
<td>3.46%</td>
</tr>
<tr>
<td>fn.</td>
<td>noreturn</td>
<td>965</td>
<td>1611</td>
<td>+66.94%</td>
<td>0.553%</td>
<td>0.923%</td>
</tr>
<tr>
<td>arg.</td>
<td>align</td>
<td>419</td>
<td>900</td>
<td>+114.80%</td>
<td>0.24%</td>
<td>0.515%</td>
</tr>
<tr>
<td>ret.</td>
<td>dereferenceable</td>
<td>19041</td>
<td>19479</td>
<td>+2.30%</td>
<td>11.2%</td>
<td>11.4%</td>
</tr>
<tr>
<td>arg.</td>
<td>nocapture</td>
<td>28991</td>
<td>29413</td>
<td>+1.46%</td>
<td>16.6%</td>
<td>16.8%</td>
</tr>
<tr>
<td>arg.</td>
<td>readonly</td>
<td>14946</td>
<td>15281</td>
<td>+2.24%</td>
<td>8.56%</td>
<td>8.75%</td>
</tr>
<tr>
<td>arg.</td>
<td>returned</td>
<td>512</td>
<td>599</td>
<td>+16.99%</td>
<td>0.293%</td>
<td>0.343%</td>
</tr>
<tr>
<td>arg.</td>
<td>noalias</td>
<td>4098</td>
<td>4158</td>
<td>+1.46%</td>
<td>2.35%</td>
<td>2.38%</td>
</tr>
<tr>
<td>ret.</td>
<td>noalias</td>
<td>1150</td>
<td>1194</td>
<td>+3.83%</td>
<td>0.676%</td>
<td>0.701%</td>
</tr>
<tr>
<td>loc.</td>
<td>attribute</td>
<td># w/o A.</td>
<td># w/ A.</td>
<td>A. Δ</td>
<td>tot. w/o A.</td>
<td>tot. w/ A.</td>
</tr>
<tr>
<td>------</td>
<td>--------------------</td>
<td>----------</td>
<td>---------</td>
<td>-------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>fn.</td>
<td>nosync</td>
<td>0</td>
<td>7612</td>
<td>0.0%</td>
<td>4.36%</td>
<td></td>
</tr>
<tr>
<td>arg.</td>
<td>dereferenceable</td>
<td>61825</td>
<td>66317</td>
<td>+7.27%</td>
<td>35.4%</td>
<td>38.0%</td>
</tr>
<tr>
<td>fn.</td>
<td>nofree</td>
<td>5762</td>
<td>10188</td>
<td>+76.81%</td>
<td>3.3%</td>
<td>5.83%</td>
</tr>
<tr>
<td>fn.</td>
<td>willreturn</td>
<td>0</td>
<td>2164</td>
<td>0.0%</td>
<td>2.37%</td>
<td></td>
</tr>
<tr>
<td>arg.</td>
<td>write</td>
<td>0</td>
<td>66317</td>
<td>0.0%</td>
<td>2.04%</td>
<td></td>
</tr>
<tr>
<td>arg.</td>
<td>read</td>
<td>0</td>
<td>10188</td>
<td>0.0%</td>
<td>3.08%</td>
<td>3.46%</td>
</tr>
<tr>
<td>fn.</td>
<td>noreturn</td>
<td>965</td>
<td>1611</td>
<td>+66.94%</td>
<td>0.553%</td>
<td>0.923%</td>
</tr>
<tr>
<td>arg.</td>
<td>align</td>
<td>419</td>
<td>900</td>
<td>+114.80%</td>
<td>0.24%</td>
<td>0.515%</td>
</tr>
<tr>
<td>ret.</td>
<td>dereferenceable</td>
<td>19041</td>
<td>19479</td>
<td>+2.30%</td>
<td>11.2%</td>
<td>11.4%</td>
</tr>
<tr>
<td>arg.</td>
<td>nocapture</td>
<td>28991</td>
<td>29413</td>
<td>+1.46%</td>
<td>16.6%</td>
<td>16.8%</td>
</tr>
<tr>
<td>arg.</td>
<td>readonly</td>
<td>14946</td>
<td>15281</td>
<td>+2.24%</td>
<td>8.56%</td>
<td>8.75%</td>
</tr>
<tr>
<td>arg.</td>
<td>returned</td>
<td>512</td>
<td>599</td>
<td>+16.99%</td>
<td>0.293%</td>
<td>0.343%</td>
</tr>
<tr>
<td>arg.</td>
<td>noalias</td>
<td>4098</td>
<td>4158</td>
<td>+1.46%</td>
<td>2.35%</td>
<td>2.38%</td>
</tr>
<tr>
<td>ret.</td>
<td>noalias</td>
<td>1150</td>
<td>1194</td>
<td>+3.83%</td>
<td>0.676%</td>
<td>0.701%</td>
</tr>
</tbody>
</table>

Details on our poster!
EVALUATION — (ATTRIBUTOR AIDED) “HEADER TIME OPTIMIZATION” (HTO)

HTO matches LTO

LTO better than HTO
EVALUATION — (ATTRIBUTOR AIDED) “HEADER TIME OPTIMIZATION” (HTO)

1. LTO is better than HTO.
2. HTO matches LTO.

Details on our poster!
Tutorial: tomorrow 1:45pm - 2:55pm
Tutorial: tomorrow 1:45pm - 2:55pm

1) introduce a new llvm::Attribute
1) introduce a new llvm::Attribute
2) derive the new llvm::Attribute with the Attributor
Tutorial: tomorrow 1:45pm - 2:55pm

1) introduce a `new llvm::Attribute`
2) derive the `new llvm::Attribute` with the Attributor
3) use the `new llvm::Attribute` to improve alias analysis
Tutorial: tomorrow 1:45pm - 2:55pm
Posters: tomorrow 4:00pm - 5:00pm
Acknowledgements

This is a problem in real programs!

What we really want are two LLVM attributes:

- `attr:readonly`
- `attr:readonly`& `attr:noalias`

LLVM automatically derives these attributes as part of the compilation process, then throws it away when it's done.

Vector

- `struct` (not deduce information for dead code, it will simplify values (think IPSCCP), perform heap-to-stack conversion, and more.

As part of this work we also added and infer new attributes (add `attr:readonly` & `attr:noalias` for functions, then throw it away when it's done):

- `attr:readonly`
- `attr:noalias`

Figure 1.

When we allow users to output a diff (easier for integration) rather than pipeline (easier for deployment), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `mystruct[3]` is incomparable, have a script to automatically generate random names), C++ member functions (since they can't forward declared), array type of `struct/classes` (type `m...
The Attributor Framework @ LLVM-Dev’19

Tutorial: tomorrow 1:45pm - 2:55pm
Posters: tomorrow 4:00pm - 5:00pm

Visit our posters and tutorial!
<table>
<thead>
<tr>
<th>loc.</th>
<th>attribute</th>
<th># w/o A.</th>
<th># w/ A.</th>
<th>A. Δ</th>
<th>tot. w/o A.</th>
<th>tot. w/ A.</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn.</td>
<td>nosync</td>
<td>0</td>
<td>78491</td>
<td></td>
<td>0.0%</td>
<td>45.90%</td>
</tr>
<tr>
<td>arg.</td>
<td>dereferenceable</td>
<td>59578</td>
<td>64214</td>
<td>+7.78%</td>
<td>34.8%</td>
<td>37.50%</td>
</tr>
<tr>
<td>fn.</td>
<td>nofree</td>
<td>25649</td>
<td>76719</td>
<td>+199.11%</td>
<td>15.0%</td>
<td>44.90%</td>
</tr>
<tr>
<td>fn.</td>
<td>willreturn</td>
<td>0</td>
<td>64748</td>
<td></td>
<td>0.0%</td>
<td>37.90%</td>
</tr>
<tr>
<td>arg.</td>
<td>writeonly</td>
<td>0</td>
<td>4229</td>
<td></td>
<td>0.0%</td>
<td>2.47%</td>
</tr>
<tr>
<td>arg.</td>
<td>readnone</td>
<td>40505</td>
<td>38414</td>
<td>-5.16%</td>
<td>23.7%</td>
<td>22.50%</td>
</tr>
<tr>
<td>fn.</td>
<td>noreturn</td>
<td>879</td>
<td>2394</td>
<td>+172.36%</td>
<td>0.514%</td>
<td>1.40%</td>
</tr>
<tr>
<td>arg.</td>
<td>align</td>
<td>449</td>
<td>1028</td>
<td>+128.95%</td>
<td>0.263%</td>
<td>0.60%</td>
</tr>
<tr>
<td>ret.</td>
<td>dereferenceable</td>
<td>18064</td>
<td>19419</td>
<td>+7.50%</td>
<td>10.8%</td>
<td>11.60%</td>
</tr>
<tr>
<td>arg.</td>
<td>nocapture</td>
<td>153523</td>
<td>155294</td>
<td>+1.15%</td>
<td>89.8%</td>
<td>90.80%</td>
</tr>
<tr>
<td>arg.</td>
<td>returned</td>
<td>9418</td>
<td>13937</td>
<td>+47.98%</td>
<td>5.51%</td>
<td>8.15%</td>
</tr>
<tr>
<td>arg.</td>
<td>noalias</td>
<td>4113</td>
<td>4189</td>
<td>+1.85%</td>
<td>2.41%</td>
<td>2.45%</td>
</tr>
<tr>
<td>ret.</td>
<td>noalias</td>
<td>3015</td>
<td>3310</td>
<td>+9.78%</td>
<td>1.81%</td>
<td>1.98%</td>
</tr>
<tr>
<td>fn.</td>
<td>writeonly</td>
<td>8089</td>
<td>9877</td>
<td>+22.10%</td>
<td>4.73%</td>
<td>5.78%</td>
</tr>
<tr>
<td>fn.</td>
<td>nounwind</td>
<td>123516</td>
<td>125480</td>
<td>+1.59%</td>
<td>72.2%</td>
<td>73.40%</td>
</tr>
</tbody>
</table>
MUST-BE-EXECUTED-CONTEXT
```c
#define internal void @f(i32* %P, il %cmp) {
    store i32 1, i32* %P;
    br il %cmp, label %then, label %else
}

then:
call void @g(i32* %P)
call void @g(i32* dereferenceable(8) %P)
br label %else

else:
call void @g(i32* %P)
ret void
}

declare void @g(i32*) { willreturn nounwind; }
```
INLINING VS. IPO

The "inline-first" approach:
I: aggressive inlining, e.g., all $N$ call sites
II: perform intra-procedural analyses + transformations ($N$ times)
III: derive information + transformation opportunities inter-procedurally

The "IPO-first" approach:
I: derive information + transformation opportunities inter-procedurally
II: internalize & specialize functions if necessary & beneficial
III: inline where benefit can be expected
The “inline-first” approach:

I: aggressive inlining, e.g., all $N$ call sites
II: perform intra-procedural analyses + transformations ($N$ times)
III: derive information + transformation opportunities inter-procedurally
INLINING VS. IPO

The “inline-first” approach:

I: aggressive inlining, e.g., all $N$ call sites
II: perform intra-procedural analyses + transformations ($N$ times)
III: derive information + transformation opportunities inter-procedurally

The “IPO-first” approach:

I: derive information + transformation opportunities inter-procedurally
II: internalize & specialize functions if necessary & beneficial
III: inline where benefit can be expected