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/

Acknowledgment

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?

```
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?

```
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?

```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?

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

What is the alignment of:

$$(1, \infty)$$

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

(4) the returned value? (5) the return type?

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

What is the alignment of:

$(1, \infty)$ $(1, \infty)$

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

(4) the returned value? (5) the return type?

```
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? $(1, \infty)$
(2) the returned value? $(1, \infty)$
(3) the argument? $(16, 16)$
(4) the returned value?
(5) the return type?

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

What is the alignment of:

$(1, \infty)$ $(16, 16) \leftarrow (16, 16)$

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

(4) the returned value? (5) the return type?

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

What is the alignment of:

$(1, \infty)$ $(16, 16) \leftarrow (16, 16)$

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

(4) the returned value? (5) the return type?

$(1, \infty)$

Fixpoint Data Flow Analysis — Alignment Example

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

What is the alignment of:

$(1, \infty)$ (1) the return type?

$(16, 16)$ (2) the returned value?

$(16, 16)$ (3) the argument?

$(1, \infty)$ (4) the returned value?

$(1, \infty)$ (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:

$(1, \infty)$ $(16, 16) \leftarrow (16, 16)$

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

(4) the returned value? (5) the return type?

$(1, \infty) \leftarrow (1, \infty)$

Fixpoint Data Flow Analysis — Alignment Example

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

What is the alignment of:

$(1, 16) \leftarrow \odot$ $(16, 16) \leftarrow (16, 16)$

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

(4) the returned value? (5) the return type?

$(1, \infty) \leftarrow (1, \infty)$

Fixpoint Data Flow Analysis — Alignment Example

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

What is the alignment of:

$(1, 16)$ $(16, 16) \leftarrow$ - - - - - - - - - - $(16, 16)$

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

(4) the returned value? (5) the return type?

$(1, \infty)$ $(1, 16)$

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

What is the alignment of:

$(1, 16)$ $(16, 16)$ ← $(16, 16)$

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

(4) the returned value? (5) the return type?

$(1, 16)$ ← $(1, 16)$

Fixpoint Data Flow Analysis — Alignment Example

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

What is the alignment of:

$(1,16)$ ⊙ $(16,16)$ $(16,16)$

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

(4) the returned value? (5) the return type?

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

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

What is the alignment of:

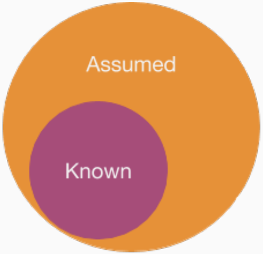
$(16,16) \leftarrow \odot$ ------- $(16,16) \leftarrow$ ------------- $(16,16)$

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

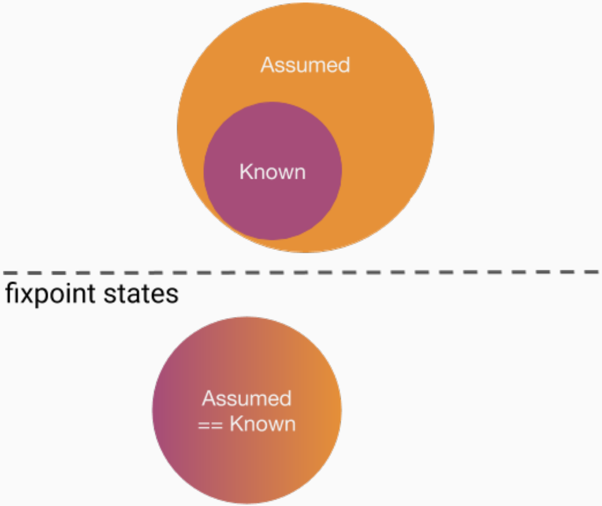
(4) the returned value? (5) the return type?

------- $(16,16) \leftarrow$ ------------- $(16,16)$

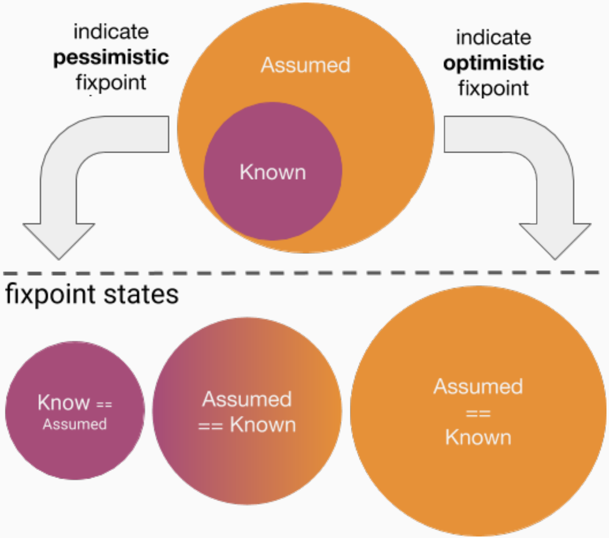
Abstract States


Assumed
Known

Abstract States



Abstract States



```
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 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());
```

```
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());
```

The Attributor — Usage

```
// Restrict deduction to specific abstract attributes.
auto Whitelist = {&AAAlign::ID,
    /* Think IP-SCCP */ &AAIsDead::ID, &AAValueSimplify::ID };

Att
```

`AAAlign` is *unaware* of `AAIsDead` and `AAValueSimplify`!

```
// 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, ...

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

- 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

The Attributor — What it is

- easy way to perform fixpoint analyses

  dependence tracking, work list algorithm, timeouts, ...

- powerful wa

  utilize concu

All good, but *why*?

- alternative to inlining

  IPO + internalization + function rewriting, e.g., argument promotion

II. Motivation

inlining has limits:

inlining has limits:

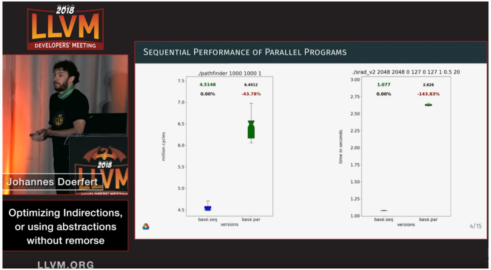
- recursion

inlining has limits:

- recursion $\equiv$ *loops*

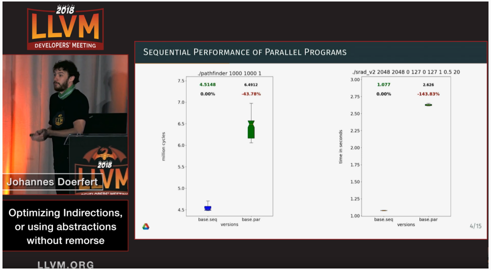
inlining has limits:

- recursion $\equiv$ *loops*
- code size



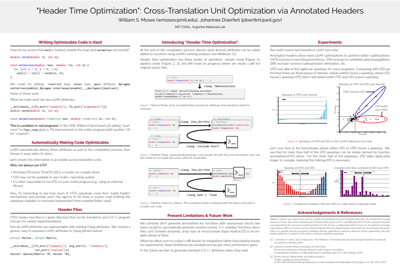
inlining has limits:

- recursion $\equiv$ *loops*
- code size
- *parallelism* (think `pthread_create`) $\Uparrow$

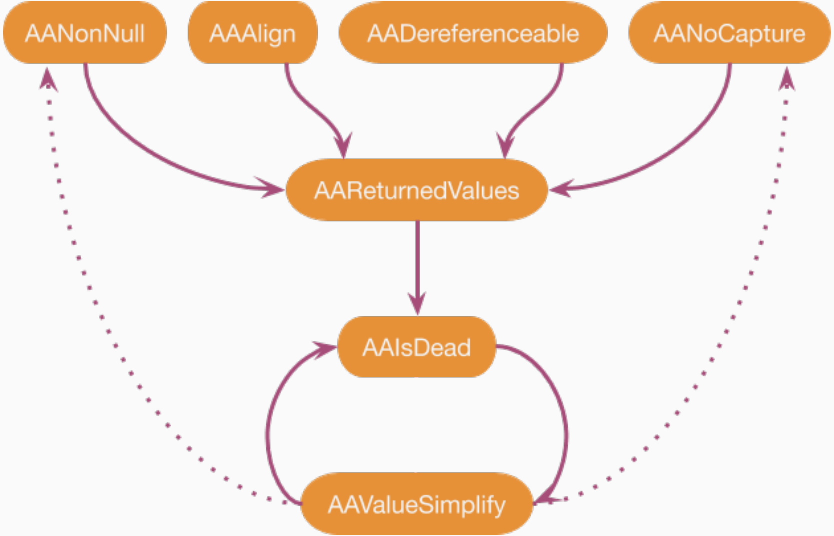


inlining has limits:

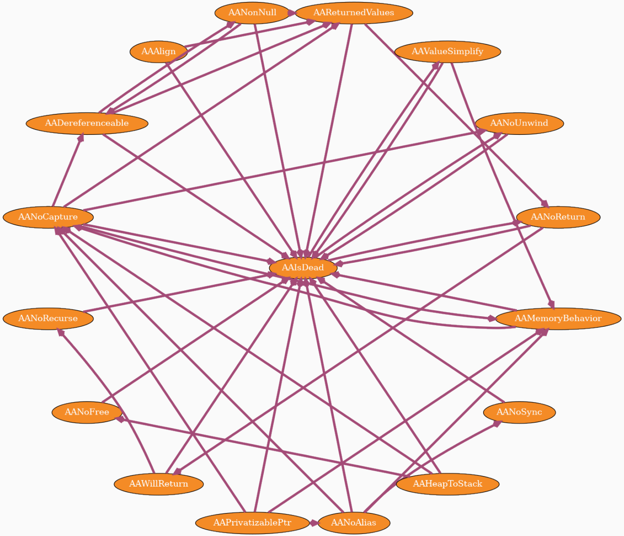
- recursion $\equiv$ *loops*
- code size
- *parallelism* (think **`pthread_create`**) $\Uparrow$
- *(declarations)* $\Rightarrow$



The Attributor — Why a Framework?

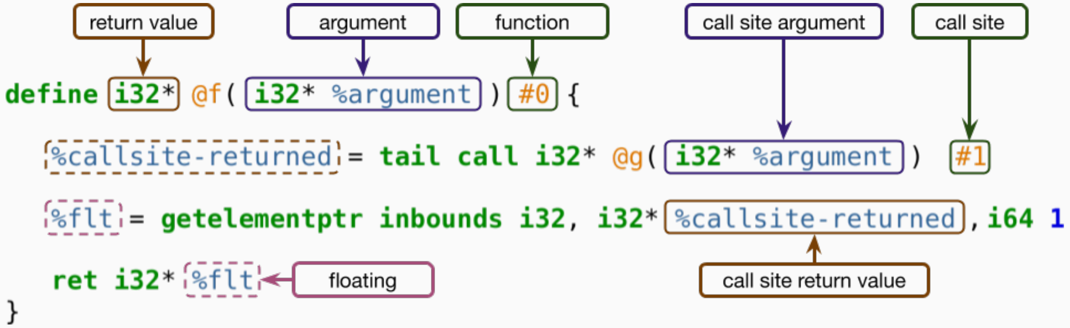


The Attributor — Why a Framework?



III. Design

LLVM-IR Positions




```
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
}
```

AAValueSimplifyReturned::updateImpl(Attributor &A)

```
ChangeStatus updateImpl(Attributor &A) override {










}
```

```
ChangeStatus updateImpl(Attributor &A) override {
  Optional<Value *> Before = getAssumedSimplifiedValue();







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

AAValueSimplifyReturned::updateImpl(Attributor &A)

```
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;
}
```

AAValueSimplifyReturned::updateImpl(Attributor &A)

```
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;
}
```

New Attributes

`nofree`

nosync

`willreturn`

`dereferenceable_globally`

Non-Attribute Deductions

liveness

returned values

value simplify

heap-2-stack

pointer privatization

The Attributor — Challenges

when to specialize for call sites
($\equiv$ "inlining + outlining")

how to seed abstract attributes
(heuristics, pgo-based, ...)

reduce overheads

combine deduction schemes, e.g.,
context-based & def-use-based

The Attributor — Challenges

...

Evaluation — FunctionAttrs (late) vs. Attributor (early)

loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	7612		0.0%	4.36%

Evaluation — FunctionAttrs (late) vs. Attributor (early)

loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	`nosync`	0	7612		0.0%	4.36%
arg.	`dereferenceable`	61825	66317	+7.27%	35.4%	38.0%

Evaluation — FunctionAttrs (late) vs. Attributor (early)

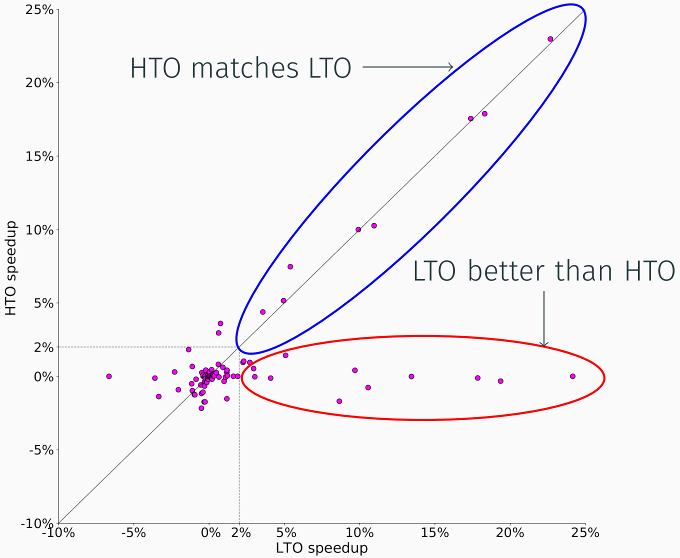
loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	7612		0.0%	4.36%
arg.	dereferenceable	61825	66317	+7.27%	35.4%	38.0%
fn.	nofree	5762	10188	+76.81%	3.3%	5.83%
fn.	willreturn	0	4146		0.0%	2.37%
arg.	writeonly	0	3562		0.0%	2.04%
arg.	readnone	5377	6040	+12.33%	3.08%	3.46%
fn.	noreturn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

Evaluation — FunctionAttrs (late) vs. Attributor (early)

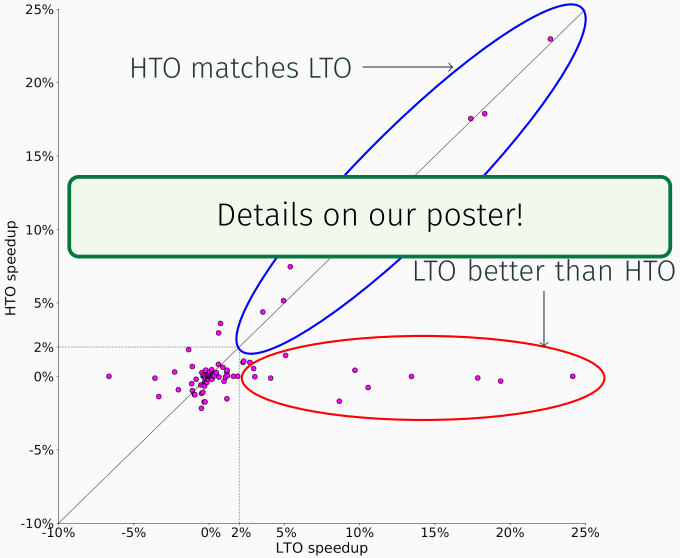
loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	7612		0.0%	4.36%
arg.	dereferenceable	61825	66317	+7.27%	35.4%	38.0%
fn.	nofree	5762	10188	+76.81%	3.3%	5.83%
fn.	willreturn	0	4146		0.0%	2.37%
arg.	write				0.0%	2.04%
arg.	read				3.08%	3.46%
fn.	noreturn	965	1611	+66.94%	0.553%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

Details on our poster!

Evaluation — (Attributor aided) "Header Time Optimiztion" (HTO)



Evaluation — (Attributor aided) "Header Time Optimiztion" (HTO)



Tutorial: tomorrow 1:45pm - 2:55pm

Tutorial: tomorrow 1:45pm - 2:55pm

1) introduce a **new** `llvm::Attribute`

Tutorial: tomorrow 1:45pm - 2:55pm

1) introduce a **new** `llvm::Attribute`
2) derive the **new** `llvm::Attribute` with the *Attributor*

Tutorial: tomorrow 1:45pm - 2:55pm

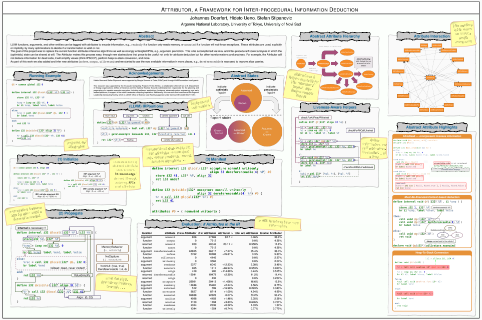
1) introduce a **new** `llvm::Attribute`
2) derive the **new** `llvm::Attribute` with the *Attributor*
3) use the **new** `llvm::Attribute` to improve *alias analysis*

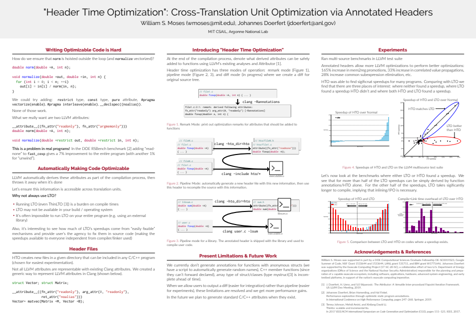
Tutorial: tomorrow 1:45pm - 2:55pm

Posters: tomorrow 4:00pm - 5:00pm

Tutorial: tomorrow 1:45pm - 2:55pm

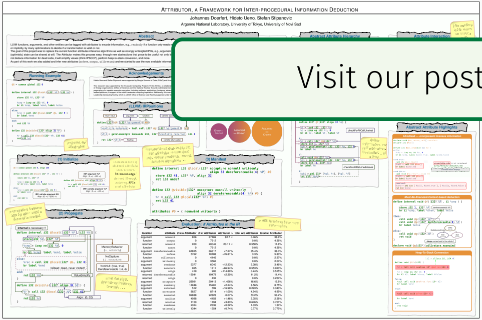
Posters: tomorrow 4:00pm - 5:00pm



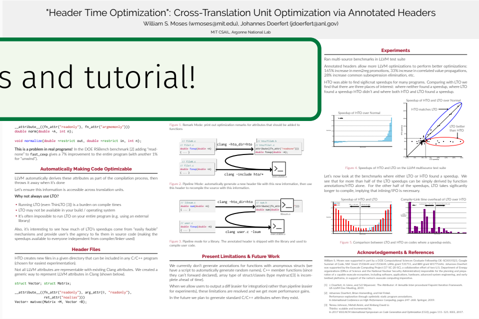


Tutorial: tomorrow 1:45pm - 2:55pm

Posters: tomorrow 4:00pm - 5:00pm



Visit our posters and tutorial!



The Attributor — Evaluation — Assuming Exact Definitions

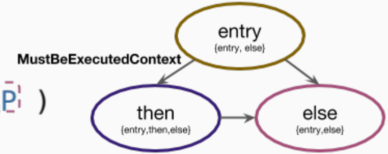
loc.	attribute	# w/o A.	# w/ A.	A. Δ	tot. w/o A.	tot. w/ A.
fn.	nosync	0	78491		0.0%	45.90%
arg.	dereferenceable	59578	64214	+7.78%	34.8%	37.50%
fn.	nofree	25649	76719	+199.11%	15.0%	44.90%
fn.	willreturn	0	64748		0.0%	37.90%
arg.	writeonly	0	4229		0.0%	2.47%
arg.	readnone	40505	38414	-5.16%	23.7%	22.50%
fn.	noreturn	879	2394	+172.36%	0.514%	1.40%
arg.	align	449	1028	+128.95%	0.263%	0.60%
ret.	dereferenceable	18064	19419	+7.50%	10.8%	11.60%
arg.	nocapture	153523	155294	+1.15%	89.8%	90.80%
arg.	returned	9418	13937	+47.98%	5.51%	8.15%
arg.	noalias	4113	4189	+1.85%	2.41%	2.45%
ret.	noalias	3015	3310	+9.78%	1.81%	1.98%
fn.	writeonly	8089	9877	+22.10%	4.73%	5.78%
fn.	nounwind	123516	125480	+1.59%	72.2%	73.40%

Must-Be-Executed-Context

Must-Be-Executed-Context

```
define internal void @f( i32* %P , i1 %cmp ) {

    store i32 1, i32* %P
    br i1 %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
```

dereferenceable(4) %P



Always comes back to the callsite !!

Inlining vs. IPO

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

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