Representation-Independent Alias Analysis and Data-Flow Analysis

Michelle Mills Strout

FRACTAL

February 10, 2007

Colorado State University
Knowledge to Go Places

OpenAnalysis Collaborators

- Paul Hovland, Priya Malusare, and Luis Ramos (Argonne)
- Shweta Behere and Andy Stone (Colorado State University)
- Brian White and Sally McKee (Cornell University)
- John Mellor-Crummey and Nathan Tallent (Rice University)
- Barbara Kreaseck (La Sierra University)
- Jean Utke (University of Chicago)

OpenAnalysis

- Problem: Insufficient analysis support in existing compiler infrastructures due to non-transferability of analysis implementations
- Decouples analysis algorithms from intermediate representations (IRs) by developing analysis-specific interfaces
- Analysis reuse across compiler infrastructures
  - Enable researchers to leverage prior work
  - Enable direct comparisons amongst analyses
  - Increase the impact of compiler analysis research

Importance of Program Analysis

- High Performance and Scientific Computing
  - performance-improving transformations (i.e. data locality and parallelism)
  - automatic differentiation (i.e. given a program F, generates program F' that calculates the derivative of outputs with respect to inputs)
- Software Engineering
  - program understanding and refactoring
  - debugging and testing
- Embedded processors
  - transformations that reduce energy consumption

Example Projects that Found Analysis Support Wanting

- Caching policies in parallel file systems [Vilayannur]
  - estimates memory references in perfect loops with constant loop bounds
  - symbolic analysis would provide a better estimate, but was unavailable in the infrastructure being used
- Hancock at AT&T [Fisher and Rogers]
  - domain-specific language for statisticians to manipulate transactions using a familiar notation
  - two researchers could not do the enormous amount of work to support domain-specific analysis of Hancock
- ROSE at LLNL [Quinlan et al.]
  - needed C++ compiler infrastructure that could perform source-to-source, high-level, and domain-specific transformations
  - ROSE has been in development for over a decade and we are just starting to add alias analysis using OpenAnalysis

Multitude of Compiler Infrastructures

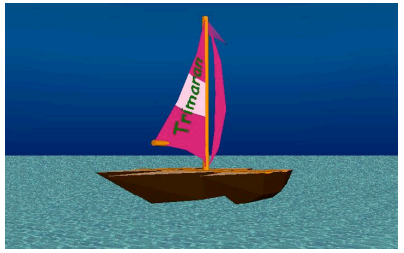
Open64/SL
software tools for source-to-source transformation of programs

PROMIS

LLVM

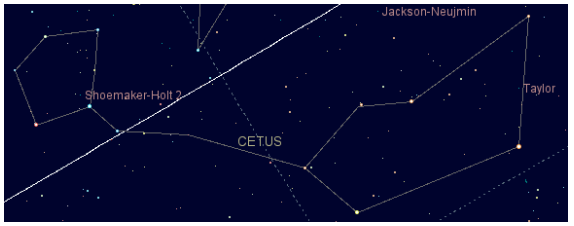
ORC

SUIF
Compiler System



ROSE

C-Breeze
C Compiler Infrastructure



Scale

A Scalable Compiler for Analytical Experiments

Colorado State University
Knowledge to Go Places

Root Causes of Analysis Support Problem

- No single intermediate representation (IR) works for all projects, therefore cannot settle on one IR
- Analysis implementation takes a significant amount of time and effort
- Compiler infrastructures are difficult to support long-term and branching can occur with no system for central updates
- We hypothesize that **most fundamental problem** is that all compiler infrastructures integrate program analysis with the program representation or IR

Outline

- Motivation and Problem
  - Analysis support problem
- OpenAnalysis: Representation-Independent Analysis
- Evaluating with respect to Alias Analysis
- Evaluating with respect to Data-Flow Analysis
- Conclusions

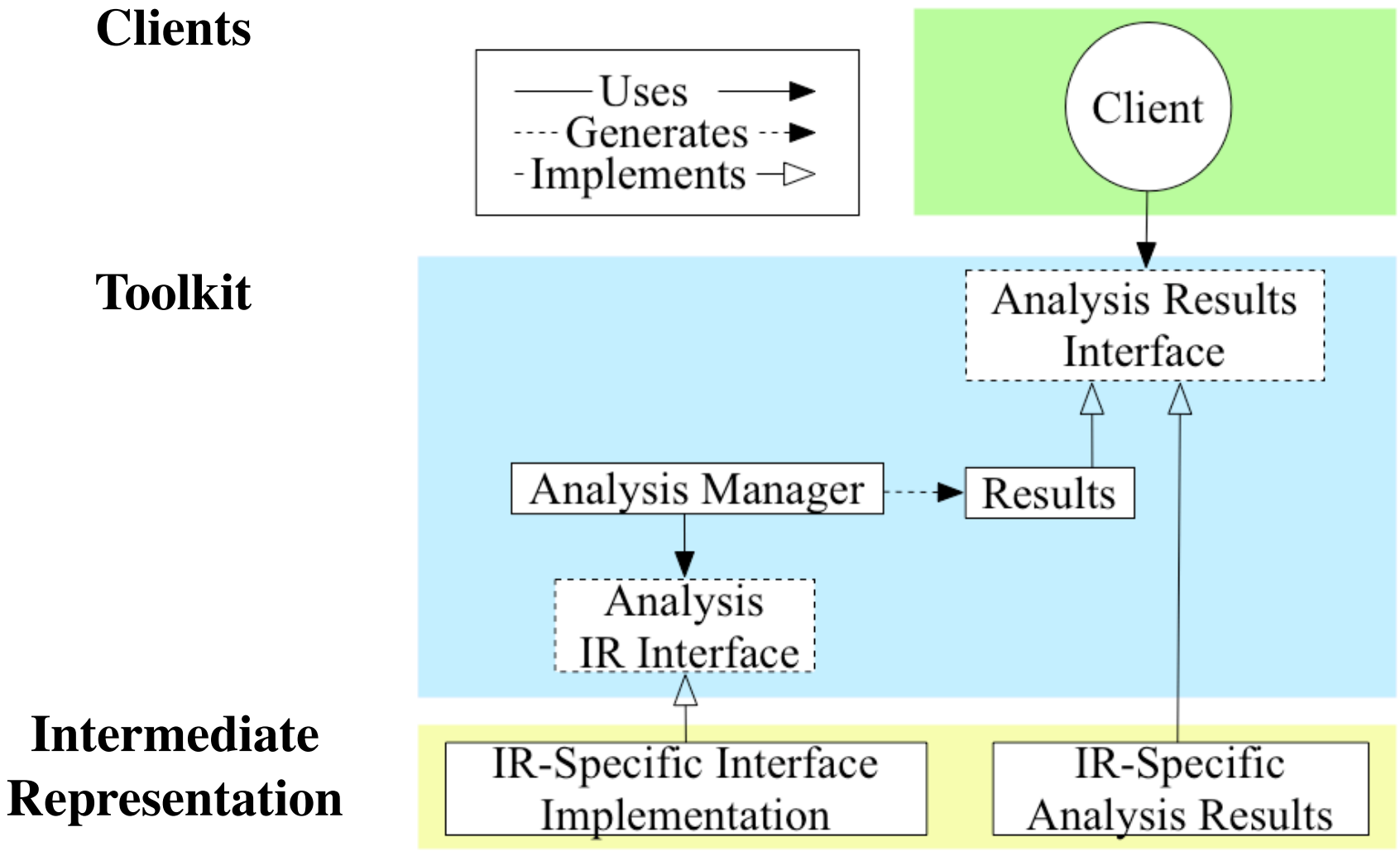
Analysis-Specific IR Interfaces

- Represent imperative programming constructs with opaque handles: procedures, statements, memory references, expressions, operations, constants, etc.
- Make queries on handles
- Example: Control-flow graph analysis

```
IRStmtType getStmtType(StmtHandle)


SIMPLE, LOOP,
STRUCT_TWOWAYCONDITIONAL, ...
```

Software Architecture for OpenAnalysis



Evaluating OpenAnalysis

- Ease of Use: how easy is it to ...
  - implement an analysis-specific IR interface?
  - contribute an analysis implementation to the toolkit?
  - use analysis results?
- Coverage
  - how many important analysis algorithms are expressible?
  - how many imperative language features can be modeled?
- Accuracy
  - how much is lost due to IR independence?
- Efficiency
  - what is the performance cost of the extra layer of abstraction?

Outline

- Motivation and Problem
  - Analysis support problem
- OpenAnalysis: Representation-Independent Analysis
- Evaluating with respect to Alias Analysis
- Evaluating with respect to Data-Flow Analysis
- Conclusions

Alias Analysis / Pointer Analysis

Determines the program state (ie. memory locations) that each memory reference may or must access

```
void foo(int &x,int &y);
int g, A[N];

void bar() {
  int a;
  foo(a,a);

  A[g] = ...;
  ... = A[5];

  int *p = &a;
  *p = ...;
  a = ...
```

- **x** and **y** are aliased in call to **foo** that happens in **bar**
- **A[g]** may alias **A[5]**
- ***p** must alias **a** until **p** is reassigned
- Other: unions, common blocks, and equivalence

Multitude of Alias/Pointer Analysis Algorithms

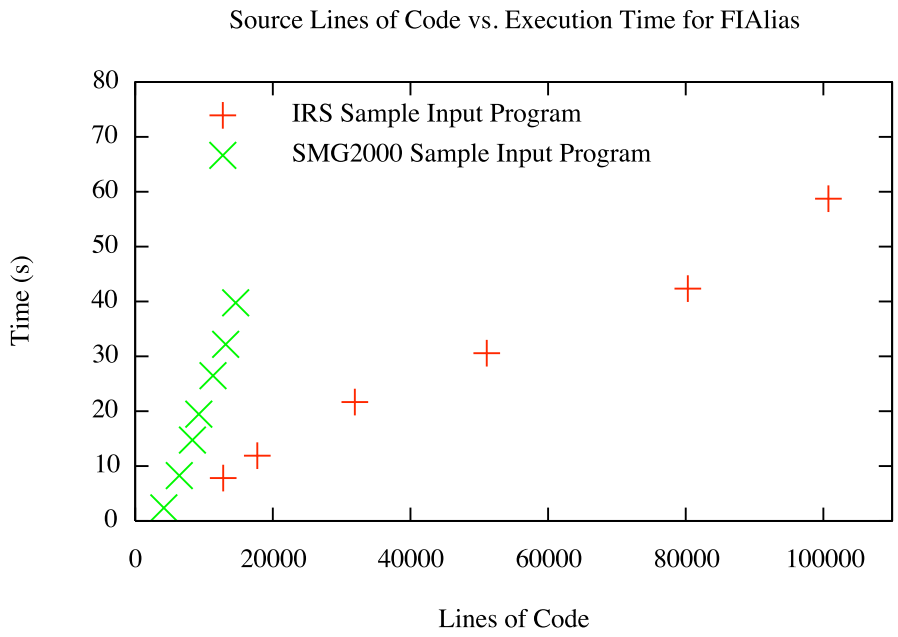
- Heuristics with varying levels of accuracy, cost, and applicability
  - [Landi 90], [Choi 93], [Emami 93], [Anderson 94], [Deutch 94], [Wilson 95], [Burke et al 95], [Steensgaard 96], [Ryder et al 00], [Cheng et al 00], [Nystrom 04], ...
- Requirements for their implementation
  - memory reference and location abstractions
  - iterator over pointer assignments in statement
  - a memory reference for each call
  - iterator over parameter bindings

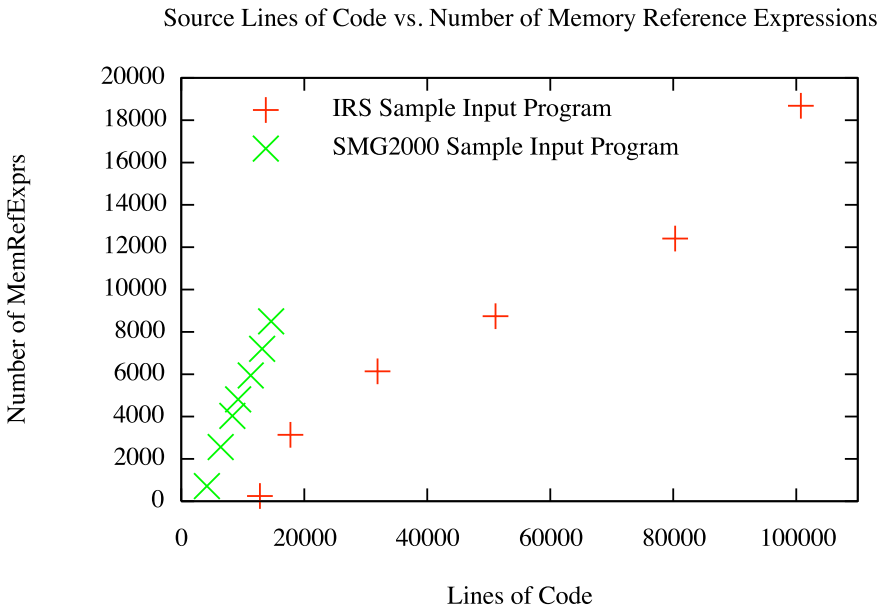
Evaluating OA wrt Alias Analysis

- Ease of Use: how easy is it to ...
  - − implement an analysis-specific IR interface? much trial and error for 1st
  - + contribute an analysis implementation? flow-insens not bad
  - + use analysis results? quite easy
- Coverage
  - − how many important analysis algorithms are expressible?
    - + FIAlias, Steensgaard, Anderson, flow and/or context sensitivity
    - − no type-based analyses
  - + how many imperative language features can be modeled? C/C++
- Accuracy
  - − how much is lost due to IR independence? still working on this
- Efficiency (see next slide for an example)

FIAlias (equivalent to Steensgaard)

Scaling Linearly wrt Memory References





Colorado State University
Knowledge to Go Places

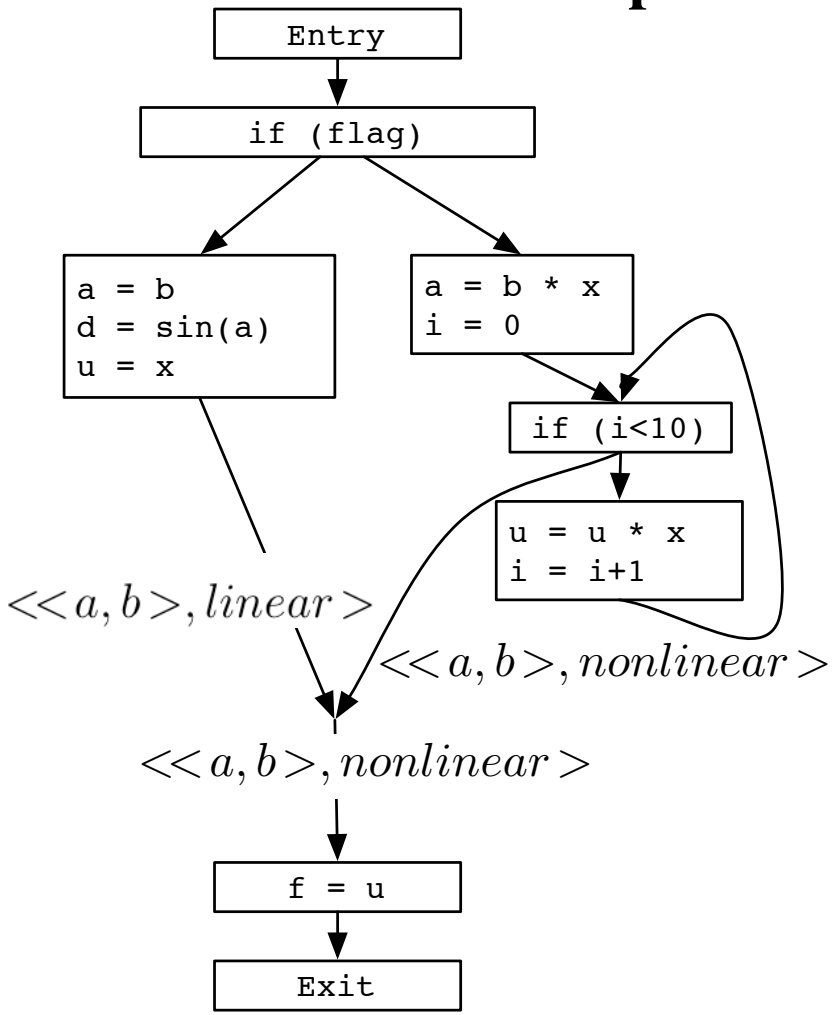
Outline

- Motivation and Problem
  - Analysis support problem
- OpenAnalysis: Representation-Independent Analysis
- Evaluating with respect to Alias Analysis
- Evaluating with respect to Data-Flow Analysis
- Conclusions

Colorado State University
Knowledge to Go Places

Data-flow Analysis Framework

Control Flow Graph



Lattice

T ≡ *no dependence*

linear

⊥ ≡ *non-linear dependence*

Meet Operation

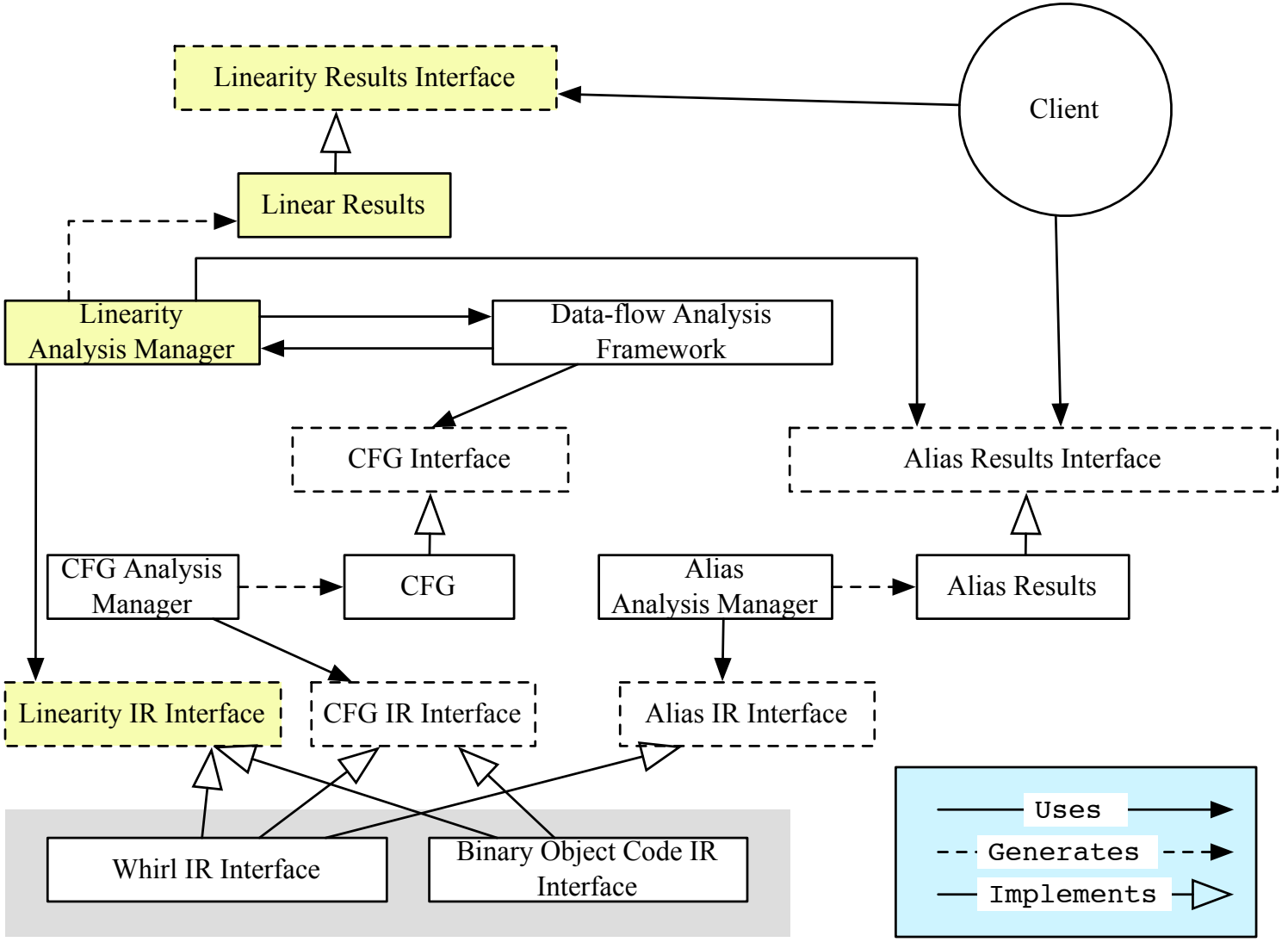
$$\top \sqcap X \rightarrow X$$

$$\bot \sqcap X \rightarrow \bot$$

Issues for Data-Flow Analysis in OA

- More than one assignment per statement
- Some languages, like C, do not have a complete ordering within the statement
- Approach
  - enable partial ordering specifications within a statement
  - operator-level transfer statements versus the current statement-level approach
  - working on a data-flow analysis framework that handles this complexity for user

Implementing Linearity Analysis within OpenAnalysis



Evaluating OA wrt Data-Flow Analysis

- Ease of Use: how easy is it to ...
  - + implement an analysis-specific IR interface? quite easy, CFG and Alias are the difficult ones
  - - contribute an analysis implementation? to much iteration in transfer func
  - + use analysis results? by statement results are easy to access
- Coverage
  - + how many important analysis algorithms are expressible? have not found any limitations here
  - + how many imperative language features can be modeled? C/C++
- Accuracy
  - - how much is lost due to IR independence? dependent on alias analysis
- Efficiency, slow, no performance optimization done yet

Outline

- Motivation and Problem
  - Analysis support problem
- OpenAnalysis: Representation-Independent Analysis
- Evaluating with respect to Alias Analysis
- Evaluating with respect to Data-Flow Analysis
- Conclusions

OpenAnalysis Status

- Number of analyses implemented (eg. FIAlias, side-effect analysis, reach defs, activity analysis, etc.)
- IR interfaces for ROSE and Open64 compiler infrastructures
- Large regression test suite
- Developing data-flow analysis implementation generation from a set-based language
- Developing array data dependence IR interface

Groups Using OpenAnalysis

- ACTS, Adjoint Compiler Technology & Standards (AD) project at Argonne, Rice, and MIT
  - WHIRL: Open64 compiler infrastructure
  - Sage3: ROSE at LLNL
- Mesh optimization tool being built by Brian White at Cornell
- Bloop tool in HPCToolkit at Rice
  - Binary code: MIPS, Alpha, Pentium4, IA64, and Sparc
- Telescoping languages at Rice
  - R: scripting language for statistical computing

Conclusions

- Language-independent program analysis enables sharing between and within compiler infrastructures
- Analysis-specific, IR-independent interfaces are the key
  - represent complex language constructs with abstractions that are basic to all imperative programming languages
  - design the interface to satisfy a broad range of implementations
- Ease of implementing the Alias IR interface needs improvement
- The OpenAnalysis toolkit is being actively used and further developed within the context of multiple projects (clients) for multiple IRs