Link-Time Analysis to Optimize Library Usage

Mark Chapman and David He

CS 706 Final Project

December 8, 2010

THE UNIVERSITY of WISCONSIN MADISON

Outline

Problem

Inefficient Library Usage

- Authors: experts in field supply a module
- Users: application developers use modules
- *Problem:* optimal usage requires expertise by both parties

Problem

Inefficient Library Usage

- Authors: experts in field supply a module
- Users: application developers use modules
- *Problem:* optimal usage requires expertise by both parties

Separate the Concerns

- Authors: supply insight to accomplish task **efficiently**
- Users: supply logic to use library **correctly**
- *Solution:* enable libraries to optimize themselves **across method calls**, not just in separate calls

Examples

Rectangle

- Need 4 values
  - upper left x, upper left y, width, height
- Remainder can be computed
  - other corners x and y, center, height/width ratio

Examples

Rectangle

- Need 4 values
  - upper left x, upper left y, width, height
- Remainder can be computed
  - other corners x and y, center, height/width ratio
- Look ahead and store **only** values that are requested later

Examples

Rectangle

- Need 4 values
  - upper left x, upper left y, width, height
- Remainder can be computed
  - other corners x and y, center, height/width ratio
- Look ahead and store **only** values that are requested later

Data Structures

- Map $\longrightarrow$ HashMap or TreeMap
- List $\longrightarrow$ LinkedList or ArrayList
- Only once use is known can the correct subtype be chosen

Matrix Problem

```
public static void main(String[] args) {
  Matrix mA = MatrixFactory.create(
      new double[] {{1.0, 2.0}, {3.0, 4.0}});
  System.out.println(proc01(mA, 4));
}
public static double[] proc01(Matrix mA, int k) {

  mA = MatrixOperations.power(mA, k);
  double[] vL = new double[mA.getNumRows()];
  MatrixOperations.eigenvalues(mA, vL);
  return vL;
}
```

Matrix Problem

```
public static void main(String[] args) {
  Matrix mA = MatrixFactory.create(
      new double[] {{1.0, 2.0}, {3.0, 4.0}});
  System.out.println(proc01(mA, 4));
}
public static double[] proc01(Matrix mA, int k) {
  mA = new EigenDecompMatrix(mA);
  mA = MatrixOperations.power(mA, k);
  double[] vL = new double[mA.getNumRows()];
  MatrixOperations.eigenvalues(mA, vL);
  return vL;
}
```

Overview

Enforce Separation of Concerns

- Library addresses performance internally
- Only abstract base types are visible to user

Overview

Enforce Separation of Concerns

- Library addresses performance internally
- Only abstract base types are visible to user

Optimize at Link-Time

- Static analysis: flow sensitive, context insensitive
- Type flow: finds best types for list of method calls
- Insert transforms: convert between concrete types

Implementation

Link-Time Optimization (LTO)

- Uses *java.lang.instrument* and *BCEL*
- Reads in annotations from self-optimizing library
- Transforms application bytecode when loaded

Implementation

Link-Time Optimization (LTO)

- Uses *java.lang.instrument* and *BCEL*
- Reads in annotations from self-optimizing library
- Transforms application bytecode when loaded

Library Information

- *Manifest*: lists base types, methods, and transforms
- *@Equivalents*: annotations point base types to concrete subtypes and base methods (acting on base types) to called methods (acting on concrete types)
- *@Cost*: annotations give relative times for methods and transforms

Problem formulation

Variables	$X = x_1, x_2, \ldots, x_{\|X\|}$
Types	$T = t_1, t_2, \ldots, t_{\|T\|}$
Method calls	$M = m_1, m_2, \ldots, m_{\|M\|}$

A *configuration* is a type assignment of variables

$$a : X \rightarrow T$$

The set of all configurations is

$$A = T^X$$

Problem formulation

Cost of method calls

$$CM : M \times A \rightarrow \mathbb{Z}$$

Cost of type transforms

$$CT : A \times A \rightarrow \mathbb{Z}$$

Goal: Select $\mathcal{A} = a_1, a_2, \ldots, a_{|M|}$ to minimize

$$C(\mathcal{A}) = \sum_{i=1}^{|M|} CM(m_i, a_i) + \sum_{i=1}^{|M|} CT(a_{i-1}, a_i).$$

LP formulation

directed graph $G = (V, E)$ cost $w : E \to \mathbb{Z}$

source s sink t capacity $c : E \to \mathbb{Z}$

flow f

$$d(v) = \begin{cases} f & \text{if } v = s \\ -f & \text{if } v = t \\ 0 & \text{otherwise} \end{cases}$$

Minimize $\sum_{e \in E} w_e x_e$ subject to

$$\sum_{v:(v,u)\in E} x_{vu} - \sum_{v:(u,v)\in E} x_{uv} = d(u) \qquad \forall u \in V$$

$$x_e \le c_e \qquad \forall e \in E$$

$$x_e \ge 0 \qquad \forall e \in E$$

LP formulation

directed graph $G = (V, E)$ — cost $w : E \to \mathbb{Z}$

source s sink t — capacity $c : E \to \mathbb{Z}$

flow f

$$d(v) = \begin{cases} f & \text{if } v = s \\ -f & \text{if } v = t \\ 0 & \text{otherwise} \end{cases}$$

$$S_m = \{(v_{0ma}, v_{1ma}) \forall a \in A_m\}$$

Minimize $\sum_{e \in E} w_e x_e$ subject to

$$\sum_{v:(v,u)\in E} x_{vu} - \sum_{v:(u,v)\in E} x_{uv} = d(u) \qquad \forall u \in V$$

$$x_e \le c_e \qquad \forall e \in E$$

$$x_e \ge 0 \qquad \forall e \in E$$

$$x_e \in \{0, f\} \qquad \forall m, \forall e \in S_m$$

Current Status

Link-Time Analysis

- Working pieces:
  - reads manifest and annotations (from library)
  - finds method calls (from application)
  - optimizes over types
- Remaining step: connect to optimizer to instrument application methods

Current Status

Link-Time Analysis

- Working pieces:
  - reads manifest and annotations (from library)
  - finds method calls (from application)
  - optimizes over types
- Remaining step: connect to optimizer to instrument application methods

Matrix Library

- Limiting visibility to abstract base types makes some decisions difficult, e.g. user might want a *sparse* matrix
- Allow user to choose some type seeds or leave all decision to optimizer alone

Possible Future Work

Meaning of Cost

- Annotate memory usage instead of execution time
- How could we annotate both and choose a balance?

Possible Future Work

Meaning of Cost

- Annotate memory usage instead of execution time
- How could we annotate both and choose a balance?

Analysis

- Static: add checkpoints where types may change
- Dynamic: change types depending on actual state
- Gains sensitivity to path and context

Possible Future Work

Meaning of Cost

- Annotate memory usage instead of execution time
- How could we annotate both and choose a balance?

Analysis

- Static: add checkpoints where types may change
- Dynamic: change types depending on actual state
- Gains sensitivity to path and context

Type System

- Qualifiers: finer control with lattice rather than hierarchy
- Multiple interfaces: alternative to positive qualifiers