Sharing Analysis:
Arrays, Collections, and Recursive Structures

Mark Marron, Mario Mendez-Lojo
Manuel Hermenegildo, Darko Stefanovic, Deepak Kapur
Motivation

- Want to optimize object-oriented programs which make use of pointer rich structures
  - In an Array or Collection (e.g. java.util.List) are there any elements that appear multiple times?
  - Differentiate structures like compiler AST with/without interned symbols --- backbone is tree with shared symbol objects or a pure tree
Motivation Cont.

- Ability to answer these sharing questions enables application of many classic optimizations
  - Thread Level Parallelization
  - Redundancy Elimination
  - Object co-location
  - Vectorization, Loop Unroll Schedule
Solution

- Start with classic *Abstract Heap Graph Model* and add additional instrumentation relations
  - Nodes represent sets of objects (or recursive data structures), edges represent sets of pointers
  - Has natural representation for data structures and connectivity properties
  - Naturally groups related sets of pointers
  - Efficient to work with
- Augment edges, which represent sets of pointers with additional information on the sharing relations between the pointers
Example: Abstract Heap Graph

A -> Data[] -> Data
[1, A]

B -> Data[]
[3, B]

[2, ?]

[4, ?]
Concrete Sharing

- Region of the heap \((O, P, P_c)\)
  - \(O\) is a set of objects
  - \(P\) is the set of the pointers between them
  - \(P_c\) the references that enter/exit the region
- Given references \(r_1, r_2\) in \(P_c\) pointing to objects \(o_1, o_2\) respectively we say:
  - alias: \(o_1 \equiv o_2\)
  - related: \(o_1 \neq o_2\) but in same weakly-connected component
  - unrelated: \(o_1\) and \(o_2\) in different weakly-connected components
Sharing: Alias and Unrelated
Sharing: Related
Abstract Representation

- Edges abstract sets of references (variable references or pointers)
- Introduce 2 related abstract properties to model sharing
  - *Interference*: Does a single edge (which abstracts possible many references) abstract only references with disjoint targets or do some of these references alias/related?
  - *Connectivity*: Do two edges abstract sets of references with disjoint targets or do some of these references alias/related?
Interference

- For a single edge how are the targets of the references it abstracts related?

- Edge e is:
  - non-interfering: all pairs of references $r_1, r_2$ in $\gamma(e)$ must be unrelated (there are none that alias or are related).
  - interfering: all pairs of references $r_1, r_2$ in $\gamma(e)$, may either be unrelated or related (there are none that alias).
  - share: all pairs of references $r_1, r_2$ in $\gamma(e)$, may be aliasing, unrelated or related.
Interference Example

[Diagram showing data flow and interference]

11
Connectivity

- For two different edges how are the targets of the references they abstract related?

- Edges $e_1, e_2$ are:
  - *disjoint*: all pairs of references $r_1$ in $\gamma(e_1)$, $r_2$ in $\gamma(e_2)$ are *unrelated* (there are none that *alias* or are *related*).
  - *connected*: all pairs of references $r_1$ in $\gamma(e_1)$, $r_2$ in $\gamma(e_2)$ may either be *unrelated* or *related* (there are none that *alias*).
  - *share*: all pairs of references $r_1$ in $\gamma(e_1)$, $r_2$ in $\gamma(e_2)$ may be *aliasing, unrelated* or *related*. 


Connectivity Example

[Diagram showing connectivity between nodes A and B with data structures and annotations.]
Case Study BH (Barnes–Hut)

- N-Body Simulation in 3-dimensions
- Uses Fast Multi-Pole method with space decomposition tree
  - For nearby bodies use naive $n^2$ algorithm
  - For distant bodies compute center of mass of many bodies and treat as single point mass
- Dynamically Updates Space Decomposition Tree to Account for Body Motion
- Has not been successfully analyzed with other existing shape analysis methods
BH Optimizations Memory

- Inline Double[] into MathVector objects, 23% serial speedup 37% memory use reduction
BH Optimizations TLP

- TLP update loop over `bodyTabRev`, factor 3.09 speedup on quad-core machine
General TLP Results

Speedup, 4 core Processor

- tsp
- em3d
- health
- voronoi
- power
- bh
- raytrace
## Benchmark Analysis Statistics

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LOC</th>
<th>Analysis Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bisort</td>
<td>560</td>
<td>0.26s</td>
</tr>
<tr>
<td>mst</td>
<td>668</td>
<td>0.12s</td>
</tr>
<tr>
<td>tsp</td>
<td>910</td>
<td>0.15s</td>
</tr>
<tr>
<td>em3d</td>
<td>1103</td>
<td>0.31s</td>
</tr>
<tr>
<td>health</td>
<td>1269</td>
<td>1.25s</td>
</tr>
<tr>
<td>voronoi</td>
<td>1324</td>
<td>1.80s</td>
</tr>
<tr>
<td>power</td>
<td>1752</td>
<td>0.36s</td>
</tr>
<tr>
<td>bh</td>
<td>2304</td>
<td>1.84s</td>
</tr>
<tr>
<td>db</td>
<td>1985</td>
<td>1.42s</td>
</tr>
<tr>
<td>raytrace</td>
<td>5809</td>
<td>37.09s</td>
</tr>
</tbody>
</table>
Conclusions

- Presented a practical abstraction for modeling sharing in programs
- Allows us to accurately model how objects are stored arrays (or Collections from java.util)
- This information can be usefully applied to compiler optimizations
  - Thread-Level Parallelization
  - Vectorization or Loop Unrolling
  - Various memory locality optimizations
Demo of the (shape) analysis available at:

www.cs.unm.edu/~marron/software.html