Cache-Conscious Structure Definition

T. Chilimbi, University of Wisconsin-Madison
B. Davidson, J. Larus, Microsoft

PLDI ‘99

Outline

- Motivation
- Contributions
- Class Splitting
  - Algorithm Description
  - Empirical Results
- Field Reordering
  - Algorithm Description
  - Empirical Results
- Conclusions

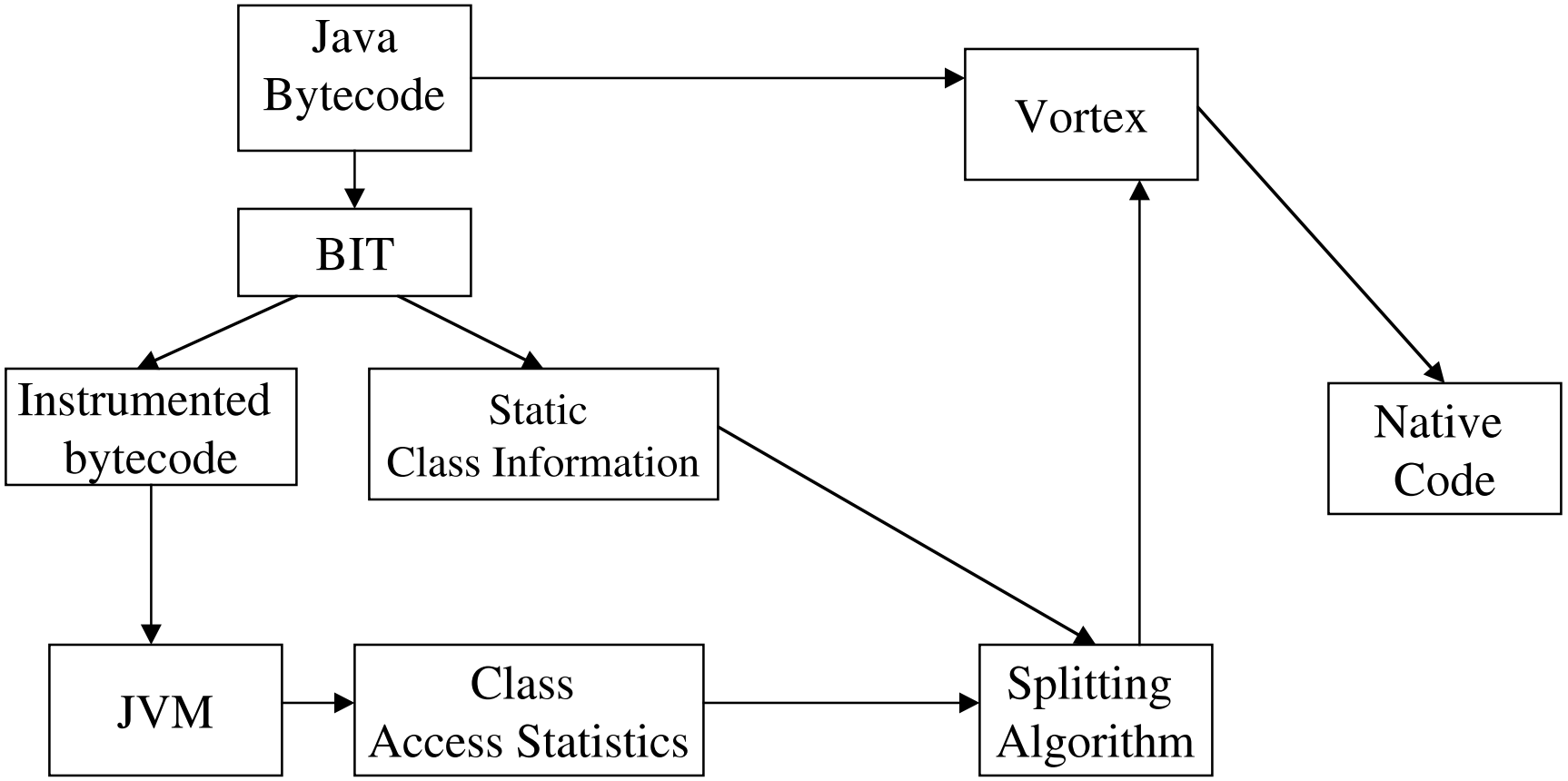
Motivation

- Processor - memory performance gap
- Data reference locality
- Improve performance of layout tools
- Small number of fields frequently accessed

Contribution

- Automatic class splitting
  - Two small objects fit into one cache block
  - Implicit pre-fetch
  - Dynamic co-location is improved
  - Faster execution time
- Field reordering recommendations
  - Time-related fields put in one cache block
  - Better cache utilization, less cache pressure

Class Splitting



Algorithm

- Step 1 - filter classes

  Frequently accessed, Size > 8 bytes, at least 2 fields

- Step 2 - mark cold fields aggressively

  If field access count < $A_t/(2*F_i)$, mark cold

- Step 3 - do split

  If cold portion > 8 bytes

  If *Temperature Differential,* then split

  Otherwise, remark cold fields

  If cold portion > 8 bytes, split.

Temperature Differential

$max(hot(class_i)) - 2 \sum cold(class_i)) >> 0$ Why?

- Assume:

$max(a_1, a_2 \ldots a_n) < cost(o_1) < \sum (a_1, a_2 \ldots a_n)$

- Benefit from locality:

$cost(o_1) + cost(o_2) <$

$(max(max(hot(class_1), max(hot(class_2))) + e) +$
$2(\sum cold(class_1) + \sum cold(class_2))$

Temperature Differential

- To benefit from splitting:

$$max(a_1, a_2 \ldots a_n) + max(b_1, b_2 \ldots b_m) > (max(max(hot(class_1), max(hot(class_2))+e) + 2(\sum cold(class_1) + \sum cold(class_2))$$

- The best they can do?

For every i $max(hot(class_i)) - 2 \sum cold(class_i)) >> 0$

Program Transformation

- Add a cold class
  - Contains public cold fields
  - Only has constructor
- Add reference in the hot class
- Transform program
  - Include reference to new class for every cold field access
  - Create cold class instance

Program Transformation

Empirical Results

- UltraSPARC, 167MHz, 2Gb, 1Mb L2
- 5 Java programs - 3K - 28K LOC
- Optimizations:
  - Vortex with aggressive optimizations (base)
  - CL object co-location (only)
  - Class Splitting + CL
- Metrics
  - L2 miss rate reduction
  - Execution time

Field Reordering

- bbcache recommends better structure field orders in C programs.
- Structures bigger than cache block
- Might be unsafe:

```
struct bar {int x; float *f; int
  y;} *p;
int *a;
a = (int *)p;
```

Bbcache

- Build structure access database (static)
  - Hash table of structures
  - For each structure, list of all instances
  - For each instance, list all accessed fields
  - For each field, list all access sites.
- Process trace (dynamic)
  - Count field accesses
  - Count contemporaneous use < 100 ms

Bbcache (cont.)

- Structure field orders
  - Build per instance affinity graphs
  - Combine into per class affinity graphs
  - Greedy field layout algorithm:
    - Step 1 - Select highest weight edge first
    - Step 2 - Append the field with highest configuration-locality increase
    - Repeat Step 2

Bbcache (cont.)

- Evaluation - qualitative metrics
  - Cache block pressure:

$$\Sigma(b_1, b_{2,} \ldots, b_n)/n$$

  - Cache block utilization:

$$\Sigma(f_{11}, f_{12,} \ldots, f_{nbn})/\Sigma(b_1, b_{2,} \ldots, b_n)$$

Empirical Results

- Pentium II Xeon, 4Gb, 1Mb L2
- MS SQL server running TPC-C
- 5 active SQL server structures reordered
- Performance improved by 2-3%

Conclusion

- Class Splitting works because
  - Field access profiles have bimodal division
  - Splitting insensitive to input data
- Benefits on hot objects co-location
- Disadvantages
  - Another level of indirection
  - Increase objects in memory
  - Code bloat - opposite of Jax transformations