Interface Dispatch in Java (and efficient dispatch in OO languages)

- Based on OOPSLA 2001 paper on the Jalapeno JVM
  - contains a thorough (but messy) discussion of related work
  - a recent point of reference and an introduction to the intricacies of Java
    - with measurements of actual programs

Interface Dispatch in Java

- We saw Myers's scheme for merging type headers in a Java-like language (no multiple inheritance but interface inheritance)
- All such optimizations could apply to Java, but decisions are made at interface load time
- Java supports separate loading:
  - when a class is loaded, the interfaces it references should not be loaded until they are used!
    - loading them may cause exceptions (e.g., not found)
  - objects should be checked dynamically to ensure that they support a certain interface when an interface method is called
    - run-time, not load-time (verifier) check
  - the results of the check could be cached for later use by the VM

On-the-fly Compilation

- High performance Java VMs are essentially compilers from bytecode to native code
- With compilation on-the-fly, the opportunities for better layout are more
  - although the entire inheritance hierarchy of a program is not known, it is slowly discovered while classes/interfaces are loaded
  - previous code can be re-compiled when method offset conflicts are detected

Common Techniques in JVMs

- Every object points to a type structure (instead of having separate pointers to all interface dispatch tables and the class dispatch table)
  - less space per object, but typically slower interface dispatch than standard C++ layout
  - *we lose the type information (interface or class) of the current object pointer*
    - *what happens when we cast the object to an interface? Need to have a lookup mechanism for methods that is not sensitive to the actual type/layout of the object*
  - the pointer to the type structure would be there anyway for run-time type info (`instanceof`), reflection, etc.
  - the type structure can contain the class dispatch table inline, but the interface dispatch tables may have conflicting method index assignments

Interface Method lookup

- Recall, we need techniques that are insensitive to the actual type of the object
  - i.e., they will work the same for all objects that implement the interface, regardless of their actual class
- Naive implementation ("searched ITables"): a linked list of interface dispatch tables is searched on every dispatch
  - some locality can be exploited by self-reorganizing lists (move-to-front)
- Better implementation: the type structure contains a hash table from interface ids to "interface tables"
- Even better implementation (CACAO JVM "Directly Indexed ITables"): make it a perfect hash (unique id per interface when it is loaded, large array—as many entries as interfaces)

  - can even keep the table small with on-the-fly compilation!!! (finite hash table, reorganize and recompile in case of hash conflicts)
- Even then, one more load is needed for interface dispatch than for virtual method dispatch (or compared to the C++ conventional layout)
  - because the object points to type structure that points to interface dispatch table that points to the code

Selector Indexed Tables

- Here is an extreme idea that provides the link to the next few ideas that are more realistic
  - all subsequent techniques are approximations of this ideal
- If we can have a perfect hash for interfaces, why not have a perfect hash for methods (method id to method code) ?
  - thus, we can avoid all method offset conflicts. All class and interface dispatch tables can be merged and can be included inline in the type structure
  - this is possible because of all code being compiled in order: we assign a unique id to every method of every interface as it gets loaded (no recompilation needed)
  - all dispatch tables in the system have slots for all these methods (perfect hashing)
  - problem? (how much space?)

An Approximation

- CACAO (proposed) "selector coloring" for perfect hash:
  - assign method offsets while classes and interfaces are loaded
  - when a conflict is detected, change the offset assignment for a conflicting interface, go back and recompile past code to be consistent with new offset assignments
  - this only works because of on-the-fly compilation

IMT-Based Dispatch

- Another approximation proposed in the paper
- Have a fixed size hash table for methods
- Instead of perfect hashing, make the common case fast (common case: no conflict)
- Jump directly to code pointed to by hash table entry, after storing a method id in a register
- In case of conflict, this code is "conflict resolution code"
  - reads the previously stored id for the method
  - performs tests (comparisons) to determine which code it should really branch to

IMT-based Performance Comparison

- In case of no conflict, the code in the hash table entry is the method code (as fast as virtual dispatch)
- In case of conflict, we have as many indirections as a "directly indexed ITable" (i.e., one more than traditional C++ dispatch-table-based dispatch) + a small computation overhead for the conflict disambiguation code
  - the latter is minimal
- Compared to a regular hash table, no hashing is needed (although still need a check and a branch) in the no-conflict case
- In case of conflicts, pointer chasing in a hash table would be very slow compared to the conflict stubs

Efficient Dynamic Dispatch: Inline Caches

- There are several general techniques to make dynamic dispatch more efficient
  - not only for interface dispatch but also for virtual method invokations
- *Inline caches*:
  - overwrite the call site with a direct call to the latest method invoked from there
  - change the prologue of that method to check that the caller is of the right type
  - if the call is not to the correct method, initiate a general lookup routine
- Inline caches are good when the call site is really monomorphic or has good temporal locality

- *Polymorphic inline caches*:
  - same thing, but for multiple previously seen cases
  - saves one indirection relative to traditional dispatch table layout