TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science · Institute for Computer Engineering

Bump-Pointer Method Caching for Embedded Java Processors

Thomas B. Preußer **Martin Zabel** **Rainer G. Spallek**

Vienna, JTRES'07

Itinerary

- Introduction
- Method Cache Design
- Evaluation
- Discussion
- (Multithreading)

Introduction

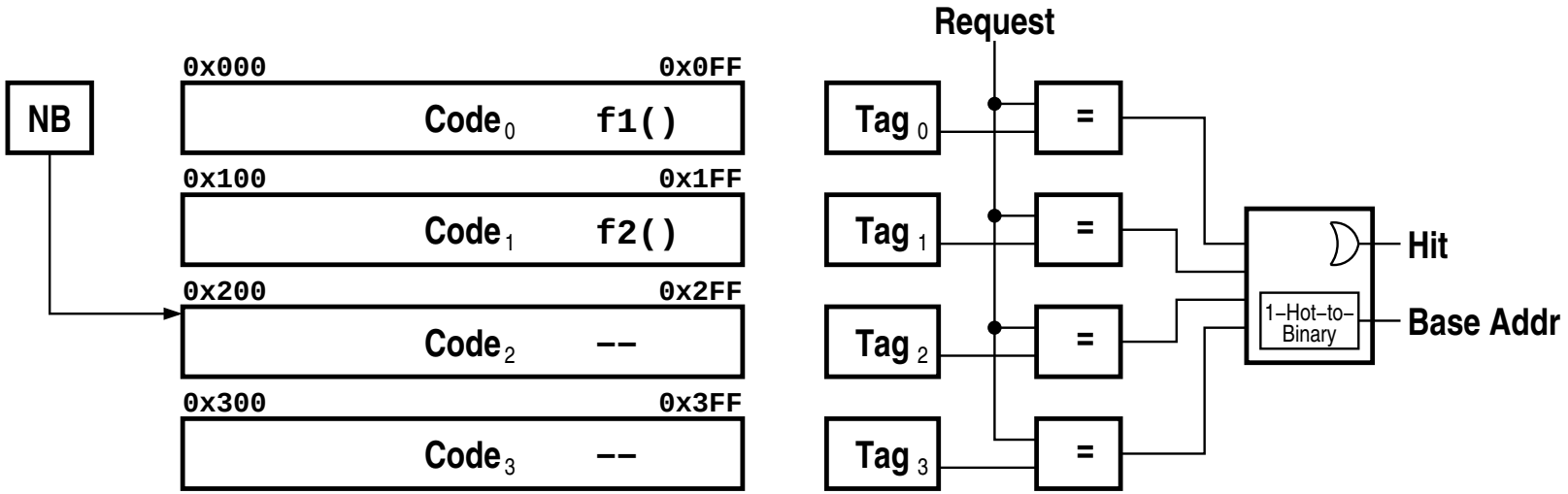
Cache hierarchy

- mandatory to span the performance gap between core and memory.
- accelerates the *common* case.
- complicates the *WCET* analysis.

Java method caches

- feasible as Java methods tend to be small.
- tie predictable costs to method invocations and returns.
- avoid intra-procedural jitter caused by misses.

Blocked Method Cache



Traditional vs. Stack-Based

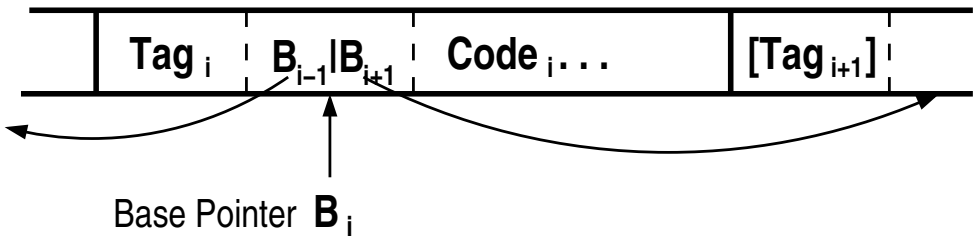
- Traditional cache *often* performs better as thrashing of multiple methods in a loop is avoided:

```
void a() {
  for(;;) {
    b();
    c();
  }
}
```

- Stack-based cache is more predictable as hits only depend on *local* call hierarchy.
- Stack-based cache can be implemented to populate available memory densely through bump-pointer allocation.

Method Data Layout

Ring Buffer with Method Entries:



Additional State:

- `PB, CB, NB` – registered copies of previous, current and next method base pointer
- `PT, CT, NT` – corresponding registered tags
- `VP` – valid pointer: start of memory area not overridden by farthest extend of call hierarchy

Hits can be distinguished into:

returning	one stack frame is unwound
recursive	re-invocation of calling method
invoking	re-invocation of previously invoked method

Valid Pointer

- is increased on method load beyond it encountering order CB, VP, NB rather than CB, NB, VP:

$$VP - CB \leq NB - CB$$

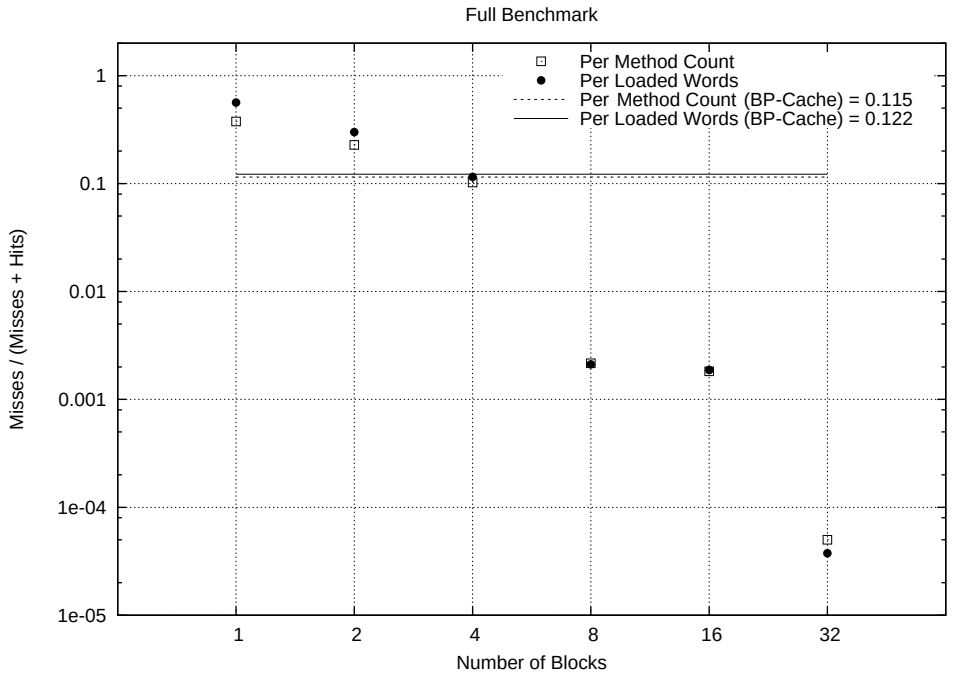
- used to validate returning hits by checking the order VP, PB, CB $\equiv$ CB, VP, PB:

$$VP - CB \leq PB - CB$$

Distance calculation ignores overflows in subtractions.
Re-use of same circuit with single input multiplexor.

Empirical Evaluation

Overall Caffeine Benchmark (without FloatTest):



Discussion

- Hit rates of about 90% comparable to a blocked cache with 4 blocks.
- Sliding window view on cache space is acceptable.
- Larger cache memory is only utilized by multi-slice implementation:
  - Slice size according to average call hierarchy.
  - Slices reduce thrashing in loops.
  - Number of comparators grows with slice number.
  - Slicing and coloring of slices may benefit multithreaded analysis.

Anomalies with Multithreading

StringTest regularly ran faster when forked into extra Thread.

Anomalies with Multithreading

StringTest regularly ran faster when forked into extra Thread.
Normal layout of method cache content (looped):

```
StringAtom.execute() ->
    StringBuffer.append() ->
        String.length() |
        String.getChars() -> System.arraycopy()
```

Anomalies with Multithreading

StringTest regularly ran faster when forked into extra Thread.
Normal layout of method cache content (looped):

```
StringAtom.execute() ->
    StringBuffer.append() ->
        String.length() |
        String.getChars() -> System.arraycopy()
```

Cache layout after destructive interruption of forking Thread yielding:

```
System.arraycopy() ->
    String.getChars() ->
        StringBuffer.append() ->
            StringAtom.execute() |
            String.length()
```

Mutual replacement now limited to small methods.

Thank you!