Function Splitting for the Method Cache

Stefan Hepp¹, Florian Brandner²

¹ Vienna University of Technology
² Technical University of Denmark

Accepted for CASES'14, October 2014
July 10, 2014
Motivation: The T-CREST Project

T-CREST: The quest for a time-predictable multicore platform
Motivation: The T-CREST Project

T-CREST: The quest for a time-predictable multicore platform
Motivation: The Method Cache

- Instruction cache that holds **blocks of code** of **variable size**
  - Functions or parts of functions (compiler defined)
  - Cache updates only at call, return and special branches
  - Fully associative tag memory

- More dynamic than I-SPM
  - Address translation, replacement done in hardware

- Less hardware costs than n-way LRU instruction cache
  - Instructions only stall in memory stage
  - Smaller tag memory

<table>
<thead>
<tr>
<th>Tag memory</th>
<th>main: 2</th>
<th>g: 10</th>
<th>f: 14</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>main()</td>
<td>g()</td>
<td>f()</td>
<td></td>
</tr>
</tbody>
</table>
Motivation: The Method Cache

- **But**: M$ can only load functions of up to a hardware-defined size
- **But**: Loading whole functions can cause large amounts of code to be evicted again before it is used
- **Compiler support to split functions into smaller subfunctions**
  - Basic blocks are too small (tag memory size, branch overhead)

Tag memory

<table>
<thead>
<tr>
<th></th>
<th>main: 2</th>
<th>g: 10</th>
<th>f: 14</th>
<th>-</th>
</tr>
</thead>
</table>

Cache

```
main()  g()  f()
```
Patmos Instruction Set Arch.

- From the method cache point of view:
  - Blocks of code

- From the ISA point of view:
  - Functions: `call`, `ret`, `xret`
    - Update return information (link registers)
  - Sub-functions: `brcf` (branch with cache fill)
    - Must (should) be single-entry regions in the CFG
    - Requires explicit `brcf` to jump to a different sub-function (no fall-through)
    - Return information not updated
Splitting Acyclic CFGs

- **Region header**: dominates all blocks in a region
- Extend regions if all predecessors are in the same region
Extend regions if all predecessors are in the same region
Extend regions if all predecessors are in the same region
Splitting Acyclic CFGs

- Extend regions if all predecessors are in the same region
Splitting Acyclic CFGs

- Extend regions if all predecessors are in the same region
Splitting Acyclic CFGs

- Extend regions if all predecessors are in the same region
Splitting Acyclic CFGs

- Extend regions if all predecessors are in the same region
Splitting Acyclic CFGs

- Extend regions if all predecessors are in the same region
Handling reducible loops

- The loop header of a reducible loop has to be region header unless all nodes in the loop are in the same code region.

- Either add whole strongly-connected components (SCCs) or make loop header a region header
Handling reducible loops

- The loop header of a reducible loop has to be region header unless all nodes in the loop are in the same code region.

- Either add whole strongly-connected components (SCCs) or make loop header a region header
Handling reducible loops

- The loop header of a reducible loop has to be region header unless all nodes in the loop are in the same code region.

- Either add whole strongly-connected components (SCCs) or make loop header a region header.

Add whole SCC (2 BBs)
Handling reducible loops

- The loop header of a reducible loop has to be region header unless all nodes in the loop are in the same code region.

- Either add whole strongly-connected components (SCCs) or make loop header a region header.
Handling reducible loops

- The loop header of a reducible loop has to be region header unless all nodes in the loop are in the same code region.

- Either add whole strongly-connected components (SCCs) or make loop header a region header.
Handling reducible loops

- The loop header of a reducible loop has to be a region header unless all nodes in the loop are in the same code region.

- Either add whole strongly-connected components (SCCs) or make loop header a region header.
Handling irreducible loops

- Add artificial loop header
- Redirect all edges to original loop headers over the artificial header
- Either add whole SCC or make all successors of the artificial header a region header

(a) An irreducible CFG.  
(b) The transformed acyclic CFG.
Handling irreducible loops

- Add **artificial loop header**
- Redirect all edges to original loop headers over the artificial header
- Either add whole SCC or make **all successors of the artificial header** a region header

(a) An irreducible CFG.  
(b) The transformed acyclic CFG.
Handling irreducible loops

- Add artificial loop header
- Redirect all edges to original loop headers over the artificial header
- Either add whole SCC or make all successors of the artificial header a region header
Handling computed branches

- Insert artificial headers, connect branch targets to form an SCC
- Form regions as before

(a) CFG with computed branch.  (b) The transformed CFG.
Handling computed branches

- Insert artificial headers, connect branch targets to form an SCC
- Form regions as before

(a) CFG with computed branch.  (b) The transformed CFG.
Handling computed branches

- Insert artificial headers, connect branch targets to form an SCC
- Form regions as before

(a) CFG with computed branch.  
(b) The transformed CFG.
Evaluation

Speedup of function splitting over 1k code blocks on a 4k 32 block method cache
Evaluation

Speedup of 4k 32 blocks M$ with variable bursts over 4-way LRU set-associative cache
Future Work

- Adapt region sizes to amount of control flow automatically
  - Merge small if-else blocks, split on diverging paths more aggressively

- Scope-based Method cache analysis uses similar approach to find single-entry scopes
  - Combine cache analysis and splitter to find good splits?
Conclusion

- Method cache requires function splitting to
  - support arbitrarily large functions
  - reduce amount of code loaded but not used

- Function splitter algorithm
  - Supports arbitrary CFGs and jumptables

- Implemented in patmos-clang, available at:
  - http://patmos.compute.dtu.dk/
  - http://github.com/t-crest/

Thanks! Questions?