Adaptive Memoization in the TreeCalc VM

Dynamic Compilation 185.A50 VU

SS 2012 24.7.2012

Stefan Neubauer

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

computer languages

Schedule

- Starting point
- Baseline
- Dynamic methods
- Results
- Lessons learned

Starting point

- TcVM as described in slides for Virtual machines
- Formula calls: 11 different instructions
  - Restructured, pulled call outside instruction-switch
- Tests
  - Regression tests
  - Model for performance tests
    - Life insurance calculations
    - premium, reserve values, indexation
    - nested recursive calls
    - ~200 different formulas called, 750 Mio. calls
    - very slow, GC very busy

Baseline

- Cache
  - LRU-Cache
  - Key (CacheKey object)
    - int id (instr nr), args V[], times-counter long[]
    - comparison: hashCode, equals
  - Value (V): object, might be big
  - Reset on changed input
- Formula: simple yes/no
  - Config 1: all non-simple
  - Config 2: compiler decides
- Main action (if formula not simple)
  - call formula: build cache key, lookup cache
  - found => done
  - not found

    => push on stack, call stack; **push cache key**, set PC

    => return instruction: **pop cache key**, write into cache

Adaptive methods

- Activate based on formula counter
- Activate based on formula runtime
- Deactivate based on cache statistics
- Run profiling, then decide

Adaptive - Counter

- Counter for each formula
- Start: no memoization
- Exclusion: simple formulas
- Threshold 10, 100 to activate
- Implementation
  - add to callstack: caching yes/no
  - otherwise problems with active calls

Adaptive - Runtime

- Runtime inklusive child calls for each formula
- Start: no memoization
- Exclusion: simple formulas
- Threshold: >0 ms, >10 ms for formula in total
- Implementation
  - cheap: System.currentTimeMillis()
  - add start-timestamp to callstack
  - remember top caller for each formula

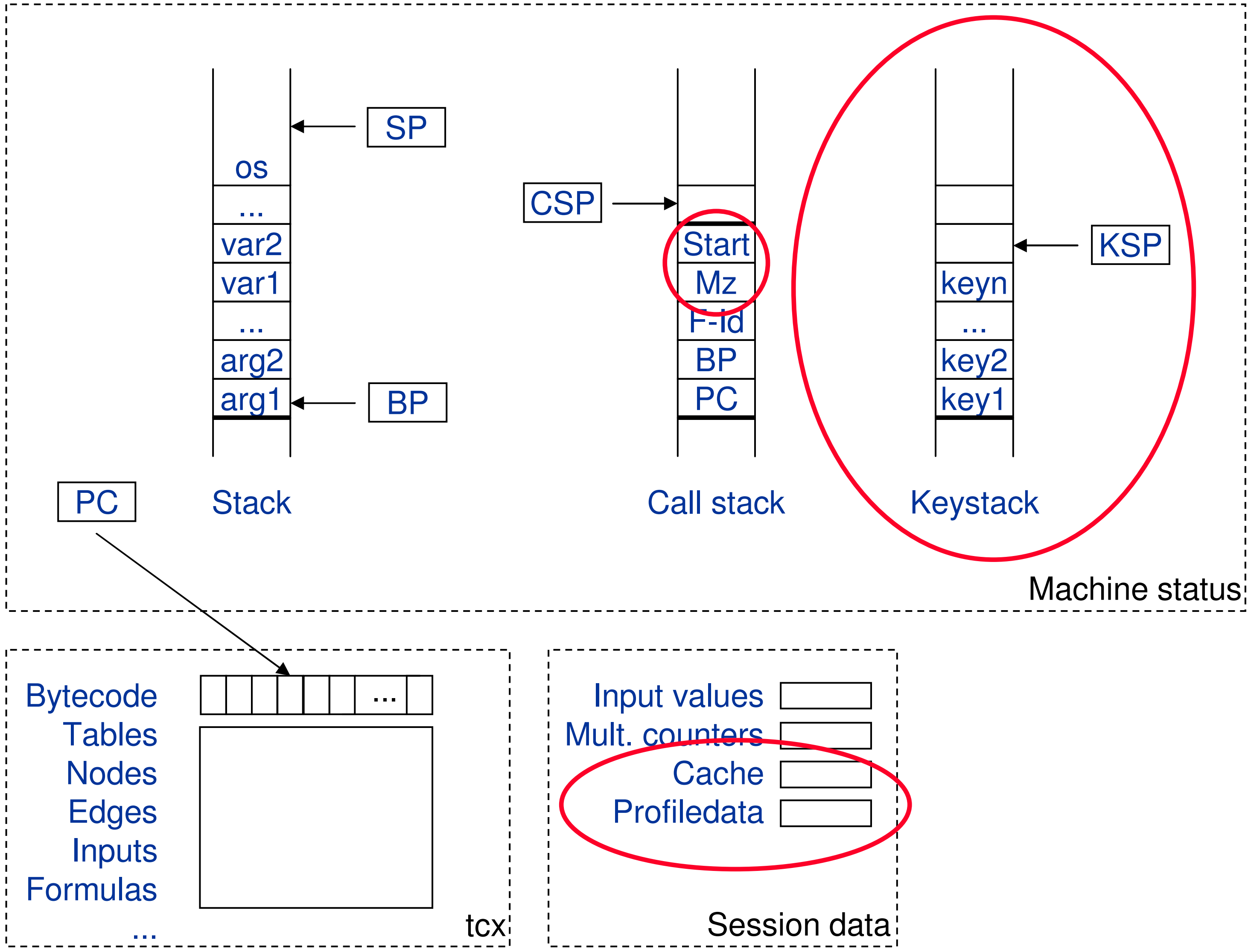
Adaptive – Cache statistics

- Cache hit, Cache miss per formula
- Start: all with memoization
- Exclusion: none
- Threshold: rate 10 %, 30 %, 50 %
  - check in interval of 100 calls
- Implementation
  - cache not reset
  - active calls of formula still cached
    - otherwise callstack/keystack update needed

Adaptive – Profiling

- Counter + runtime + cache statistics
- Calls 1-10: no memoization
- Calls 11-50: with memoization
- Exclusion: none
- Threshold:
  - cache hit-rate > 30%, and
  - runtime > 10 ms
- One check after 50 calls
- Implementation
  - check counters on formula call

TcVM – status

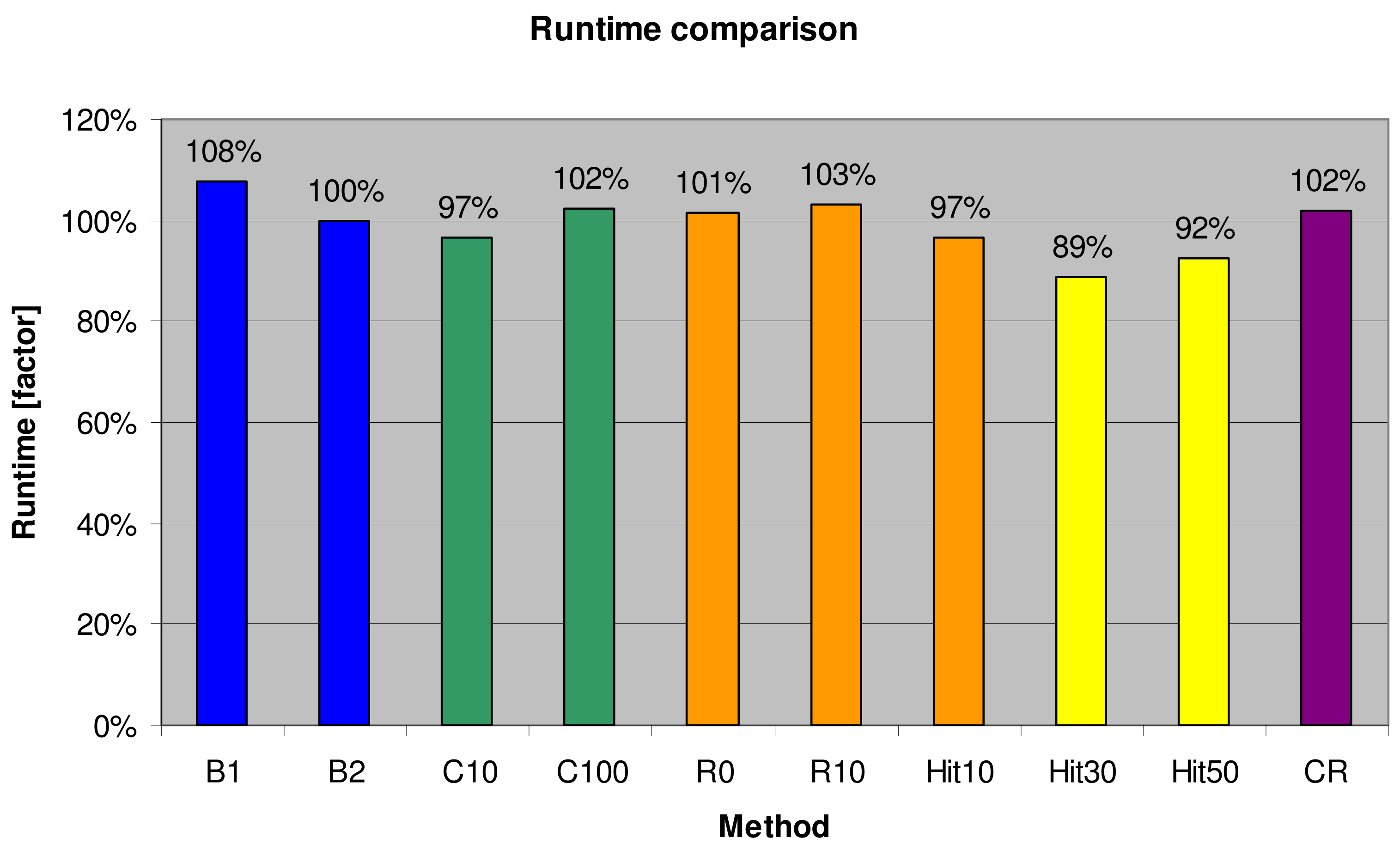


SP
os
...
var2
var1
...
arg2
arg1
BP
PC
Stack
CSP
Start
Mz
F-Id
BP
PC
Call stack
KSP
keyn
...
key2
key1
Keystack
Machine status
Bytecode
Tables
Nodes
Edges
Inputs
Formulas
...
tcx
Input values
Mult. counters
Cache
Profiledata
Session data

Results

- Starting point: one night
- Baseline
  - config1: 3,531 sec, config2: **3,281** sec
- Adaptive
  - Counter 10: **3,172** sec, Counter 100: 3,359 sec
- Runtime
  - >0 ms: 3,329; >10 ms: 3,390
- Cache hit rate
  - <10%: 3,172; <30%: **2,906**; <50%: 3,031
- Cache and Runtime
  - 3,343 sec

Results



Runtime comparison
120%
100%
80%
60%
40%
20%
0%
Runtime [factor]
108%
100%
97%
102%
101%
103%
97%
89%
92%
102%
B1
B2
C10
C100
R0
R10
Hit10
Hit30
Hit50
CR
Method

Lessons learned

- Proper infrastructure (profiling class etc.) needed
- Most of work
  - comparisions
  - working out heuristics
  - handling of recursive calls
- Performance
  - Naive caching (with LRU) already very useful
  - Improvements with simple methods reached
  - Overhead not that bad
  - One bottleneck: tree access (by macroprogram)