# Return Value Prediction in a Java Virtual Machine

Christopher J.F. Pickett Clark Verbrugge

{cpicke,clump}@sable.mcgill.ca

School of Computer Science, McGill University

Montréal, Québec, Canada H3A 2A7

# Overview

- Introduction and Related Work
- Contributions
- Design
- Benchmark Properties
- Size Variation
- Memory Usage
- Hybrid Performance
- Conclusions and Future Work

# Introduction and Related Work

- Speculative method-level parallelism (SMLP) allows for dynamic parallelisation of single-threaded programs
  - speculative threads are forked at callsites
  - suitable for Java virtual machines

- Perfect return value prediction can double performance of SMLP (Hu *et al.*, 2003)

- Goals
  - Implement Hu's predictors in SableVM
  - Achieve higher accuracy

# Speculative Method-Level Parallelism

```
// execute foo non-speculatively
r = foo (a, b, c);

// execute past return point
// speculatively in parallel with foo()
if (r > 10)
  {
    s = o1.f;    // buffer head reads
    o2.f = r;    // buffer heap writes
  }
...
```

# Impact of Return Value Prediction

| RVP strategy | return value | SMLP speedup |
|---|---|---|
| none | arbitrary | 1.52 |
| best | predicted | 1.92 |
| perfect | correct | 2.76 |

- 26% speedup over no RVP with Hu's best predictor
- 82% speedup over no RVP with perfect prediction
  - Improved hybrid accuracy is highly desirable


- S. Hu., R. Bhargava, and L. K. John. The role of return value prediction in exploiting speculative method-level parallelism. *Journal of Instruction-Level Parallelism*, 5:1–21, Nov. 2003.

# Contributions (1)

- Expand on previous data collected
  - Use S100 instead of S1 (size 1) for SPEC JVM98
  - Report all return types, not just boolean, int, ref
  - Explicitly account for exceptions (run `jack`)
  - Predict all method calls (no inlining)
- Implement existing predictors in JVM
  - last value, stride, 2-delta stride
  - parameter stride
  - finite context method (FCM)
  - hybrid

# Contributions (2)

- New memoization predictor
  - Table-based, like context predictor
  - Hashes together method arguments
  - Performs well in a hybrid
- Explore predictor performance limits
  - Allocate storage until accuracy no longer improves
  - Reduce memory requirements
    - Dynamically expand hashtables
    - Exploit VM info about value widths

# Design

- Implement all predictors in software JVM
  - not trace-based
  - not simulated
- Fixed size predictors:
  - **L**ast value – last callsite return value
  - **S**tride – prediction = *r1 + (r1 - r2)*
  - **2**-delta stride – update after two identical strides
  - **P**arameter stride – search for and capture stride between *r* and one parameter
- Focus on predictors with variable size

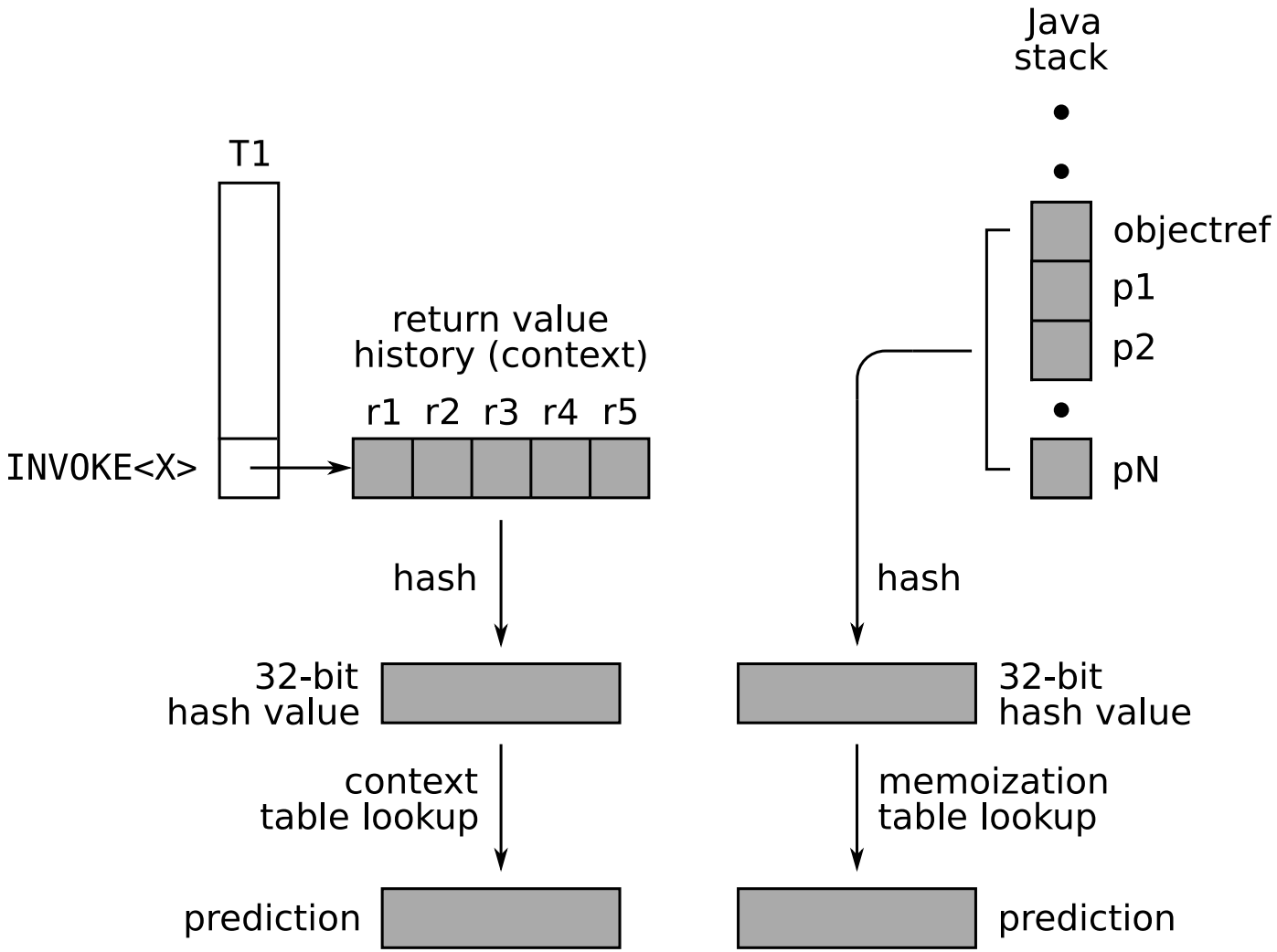
# Design

- Variable-memory table-based predictors:
  - **C**ontext – inputs are return value history
  - **M**emoization – inputs are method parameters
- Hash values together, use extra bits as tag
  - Rehash on tag collisions
  - Use direct addressing, not chaining
  - Use Jenkins' fast hash to get even distribution
- Attach context and memoization tables per callsite
- Expand tables if load > 75%, up to a fixed maximum

# Design

- Hybrid predictor
  - Chooses best sub-predictor over last 32 values
  - **LS2PC** – all previous predictors
  - **LS2PCM** – all previous predictors + memoization

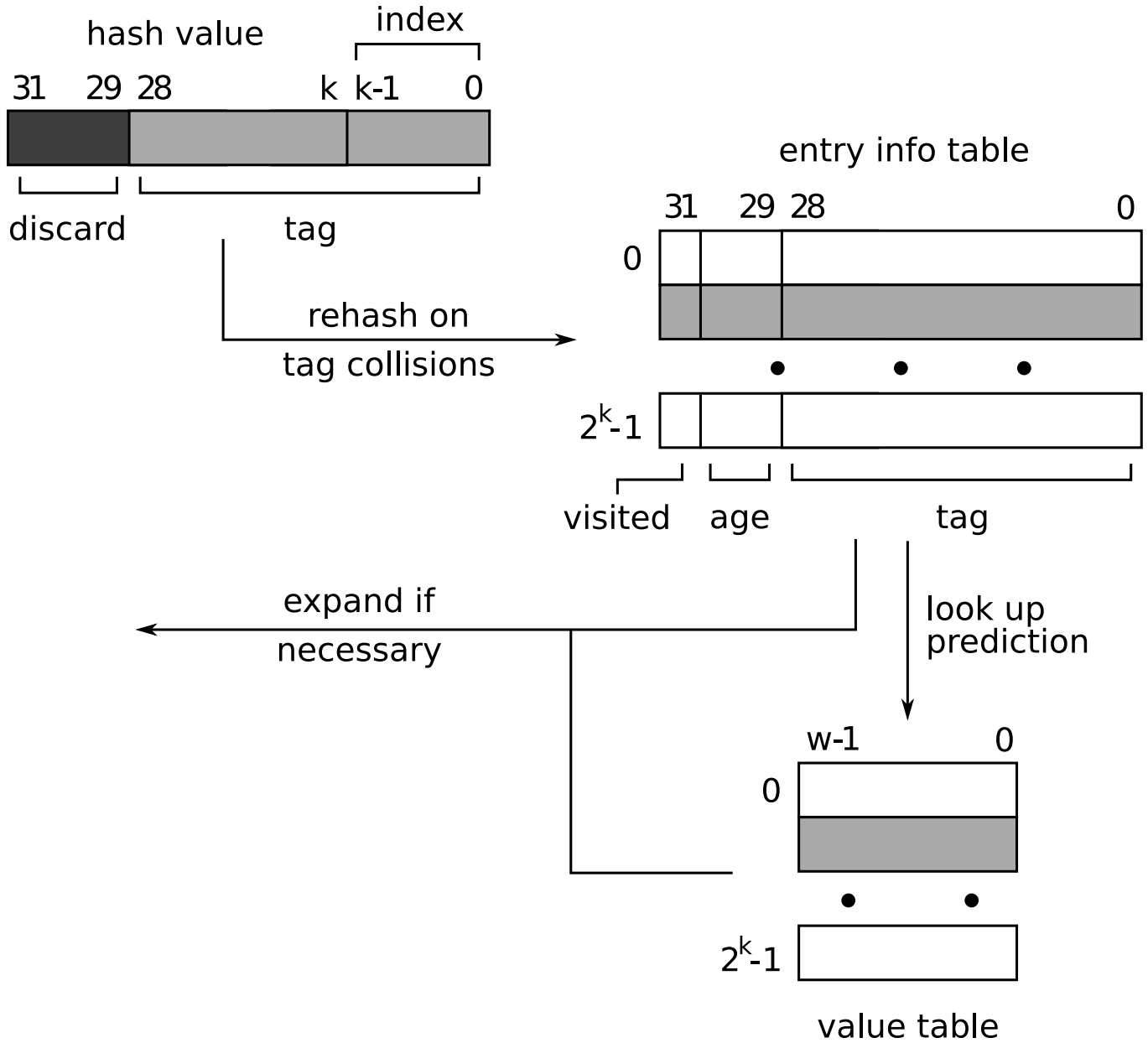
# Context and Memoization Predictors




(a) Context Predictor

(b) Memoization Predictor

# Hashtable Lookup and Expansion



# SPEC JVM98 Dynamic Properties

| property | comp | db | jack | javac | jess | mpeg | mtrt |
|---|---|---|---|---|---|---|---|
| callsites | 1.72K | 1.89K | 3.60K | 5.12K | 3.04K | 2.17K | 2.90K |
| forked | 226M | 170M | 59.4M | 127M | 125M | 111M | 288M |
| aborted | 36 | 18 | 608K | 41.8K | 290 | 114 | 62 |
| void | 93.4M | 54.4M | 24.4M | 45.3M | 23.3M | 34.1M | 20.5M |
| verified | 133M | 115M | 34.4M | 81.5M | 102M | 76.9M | 267M |
| boolean Z | 3.75K | 11.1M | 9.38M | 17.5M | 35.8M | 24.3M | 3.06M |
| byte B | 0 | 0 | 580K | 39.3K | 0 | 0 | 0 |
| char C | 935 | 1.73K | 1.55M | 3.70M | 6.65K | 2.11K | 9.84K |
| short S | 0 | 0 | 0 | 73.3K | 0 | 18.0M | 0 |
| int I | 133M | 48.0M | 11.5M | 36.5M | 20.8M | 34.6M | 4.54M |
| long J | 477 | 152K | 1.23M | 845K | 101K | 15.8K | 2.13K |
| float F | 0 | 0 | 0 | 96 | 280 | 7.81K | 162M |
| double D | 0 | 0 | 0 | 156 | 1.77M | 56 | 188K |
| reference R | 15.8K | 56.2M | 10.2M | 22.9M | 43.5M | 32.7K | 97.5M |

# SPEC JVM98 Dynamic Properties

| property | comp | db | jack | javac | jess | mpeg | mtrt |
|---|---|---|---|---|---|---|---|
| callsites | 1.72K | 1.89K | 3.60K | 5.12K | 3.04K | 2.17K | 2.90K |
| forked | 226M | 170M | 59.4M | 127M | 125M | 111M | 288M |
| aborted | 36 | 18 | 608K | 41.8K | 290 | 114 | 62 |
| void | 93.4M | 54.4M | 24.4M | 45.3M | 23.3M | 34.1M | 20.5M |
| verified | 133M | 115M | 34.4M | 81.5M | 102M | 76.9M | 267M |
| boolean Z | 3.75K | 11.1M | 9.38M | 17.5M | 35.8M | 24.3M | 3.06M |
| byte B | 0 | 0 | 580K | 39.3K | 0 | 0 | 0 |
| char C | 935 | 1.73K | 1.55M | 3.70M | 6.65K | 2.11K | 9.84K |
| short S | 0 | 0 | 0 | 73.3K | 0 | 18.0M | 0 |
| int I | 133M | 48.0M | 11.5M | 36.5M | 20.8M | 34.6M | 4.54M |
| long J | 477 | 152K | 1.23M | 845K | 101K | 15.8K | 2.13K |
| float F | 0 | 0 | 0 | 96 | 280 | 7.81K | 162M |
| double D | 0 | 0 | 0 | 156 | 1.77M | 56 | 188K |
| reference R | 15.8K | 56.2M | 10.2M | 22.9M | 43.5M | 32.7K | 97.5M |

# SPEC JVM98 Dynamic Properties

| property | comp | db | jack | javac | jess | mpeg | mtrt |
|---|---|---|---|---|---|---|---|
| callsites | 1.72K | 1.89K | 3.60K | 5.12K | 3.04K | 2.17K | 2.90K |
| forked | 226M | 170M | 59.4M | 127M | 125M | 111M | 288M |
| aborted | 36 | 18 | 608K | 41.8K | 290 | 114 | 62 |
| void | 93.4M | 54.4M | 24.4M | 45.3M | 23.3M | 34.1M | 20.5M |
| verified | 133M | 115M | 34.4M | 81.5M | 102M | 76.9M | 267M |
| boolean Z | 3.75K | 11.1M | 9.38M | 17.5M | 35.8M | 24.3M | 3.06M |
| byte B | 0 | 0 | 580K | 39.3K | 0 | 0 | 0 |
| char C | 935 | 1.73K | 1.55M | 3.70M | 6.65K | 2.11K | 9.84K |
| short S | 0 | 0 | 0 | 73.3K | 0 | 18.0M | 0 |
| int I | 133M | 48.0M | 11.5M | 36.5M | 20.8M | 34.6M | 4.54M |
| long J | 477 | 152K | 1.23M | 845K | 101K | 15.8K | 2.13K |
| float F | 0 | 0 | 0 | 96 | 280 | 7.81K | 162M |
| double D | 0 | 0 | 0 | 156 | 1.77M | 56 | 188K |
| reference R | 15.8K | 56.2M | 10.2M | 22.9M | 43.5M | 32.7K | 97.5M |

# SPEC JVM98 Dynamic Properties

| property | comp | db | jack | javac | jess | mpeg | mtrt |
|---|---|---|---|---|---|---|---|
| callsites | 1.72K | 1.89K | 3.60K | 5.12K | 3.04K | 2.17K | 2.90K |
| forked | 226M | 170M | 59.4M | 127M | 125M | 111M | 288M |
| aborted | 36 | 18 | 608K | 41.8K | 290 | 114 | 62 |
| void | 93.4M | 54.4M | 24.4M | 45.3M | 23.3M | 34.1M | 20.5M |
| verified | 133M | 115M | 34.4M | 81.5M | 102M | 76.9M | 267M |
| boolean Z | 3.75K | 11.1M | 9.38M | 17.5M | 35.8M | 24.3M | 3.06M |
| byte B | 0 | 0 | 580K | 39.3K | 0 | 0 | 0 |
| char C | 935 | 1.73K | 1.55M | 3.70M | 6.65K | 2.11K | 9.84K |
| short S | 0 | 0 | 0 | 73.3K | 0 | 18.0M | 0 |
| int I | 133M | 48.0M | 11.5M | 36.5M | 20.8M | 34.6M | 4.54M |
| long J | 477 | 152K | 1.23M | 845K | 101K | 15.8K | 2.13K |
| float F | 0 | 0 | 0 | 96 | 280 | 7.81K | 162M |
| double D | 0 | 0 | 0 | 156 | 1.77M | 56 | 188K |
| reference R | 15.8K | 56.2M | 10.2M | 22.9M | 43.5M | 32.7K | 97.5M |

# SPEC JVM98 Dynamic Properties

| property | comp | db | jack | javac | jess | mpeg | mtrt |
|---|---|---|---|---|---|---|---|
| callsites | 1.72K | 1.89K | 3.60K | 5.12K | 3.04K | 2.17K | 2.90K |
| forked | 226M | 170M | 59.4M | 127M | 125M | 111M | 288M |
| aborted | 36 | 18 | 608K | 41.8K | 290 | 114 | 62 |
| void | 93.4M | 54.4M | 24.4M | 45.3M | 23.3M | 34.1M | 20.5M |
| verified | 133M | 115M | 34.4M | 81.5M | 102M | 76.9M | 267M |
| boolean Z | 3.75K | 11.1M | 9.38M | 17.5M | 35.8M | 24.3M | 3.06M |
| byte B | 0 | 0 | 580K | 39.3K | 0 | 0 | 0 |
| char C | 935 | 1.73K | 1.55M | 3.70M | 6.65K | 2.11K | 9.84K |
| short S | 0 | 0 | 0 | 73.3K | 0 | 18.0M | 0 |
| int I | 133M | 48.0M | 11.5M | 36.5M | 20.8M | 34.6M | 4.54M |
| long J | 477 | 152K | 1.23M | 845K | 101K | 15.8K | 2.13K |
| float F | 0 | 0 | 0 | 96 | 280 | 7.81K | 162M |
| double D | 0 | 0 | 0 | 156 | 1.77M | 56 | 188K |
| reference R | 15.8K | 56.2M | 10.2M | 22.9M | 43.5M | 32.7K | 97.5M |

# SPEC JVM98 Dynamic Properties

| property | comp | db | jack | javac | jess | mpeg | mtrt |
|---|---|---|---|---|---|---|---|
| callsites | 1.72K | 1.89K | 3.60K | 5.12K | 3.04K | 2.17K | 2.90K |
| forked | 226M | 170M | 59.4M | 127M | 125M | 111M | 288M |
| aborted | 36 | 18 | 608K | 41.8K | 290 | 114 | 62 |
| void | 93.4M | 54.4M | 24.4M | 45.3M | 23.3M | 34.1M | 20.5M |
| verified | 133M | 115M | 34.4M | 81.5M | 102M | 76.9M | 267M |
| boolean Z | 3.75K | 11.1M | 9.38M | 17.5M | 35.8M | 24.3M | 3.06M |
| byte B | 0 | 0 | 580K | 39.3K | 0 | 0 | 0 |
| char C | 935 | 1.73K | 1.55M | 3.70M | 6.65K | 2.11K | 9.84K |
| short S | 0 | 0 | 0 | 73.3K | 0 | 18.0M | 0 |
| int I | 133M | 48.0M | 11.5M | 36.5M | 20.8M | 34.6M | 4.54M |
| long J | 477 | 152K | 1.23M | 845K | 101K | 15.8K | 2.13K |
| float F | 0 | 0 | 0 | 96 | 280 | 7.81K | 162M |
| double D | 0 | 0 | 0 | 156 | 1.77M | 56 | 188K |
| reference R | 15.8K | 56.2M | 10.2M | 22.9M | 43.5M | 32.7K | 97.5M |

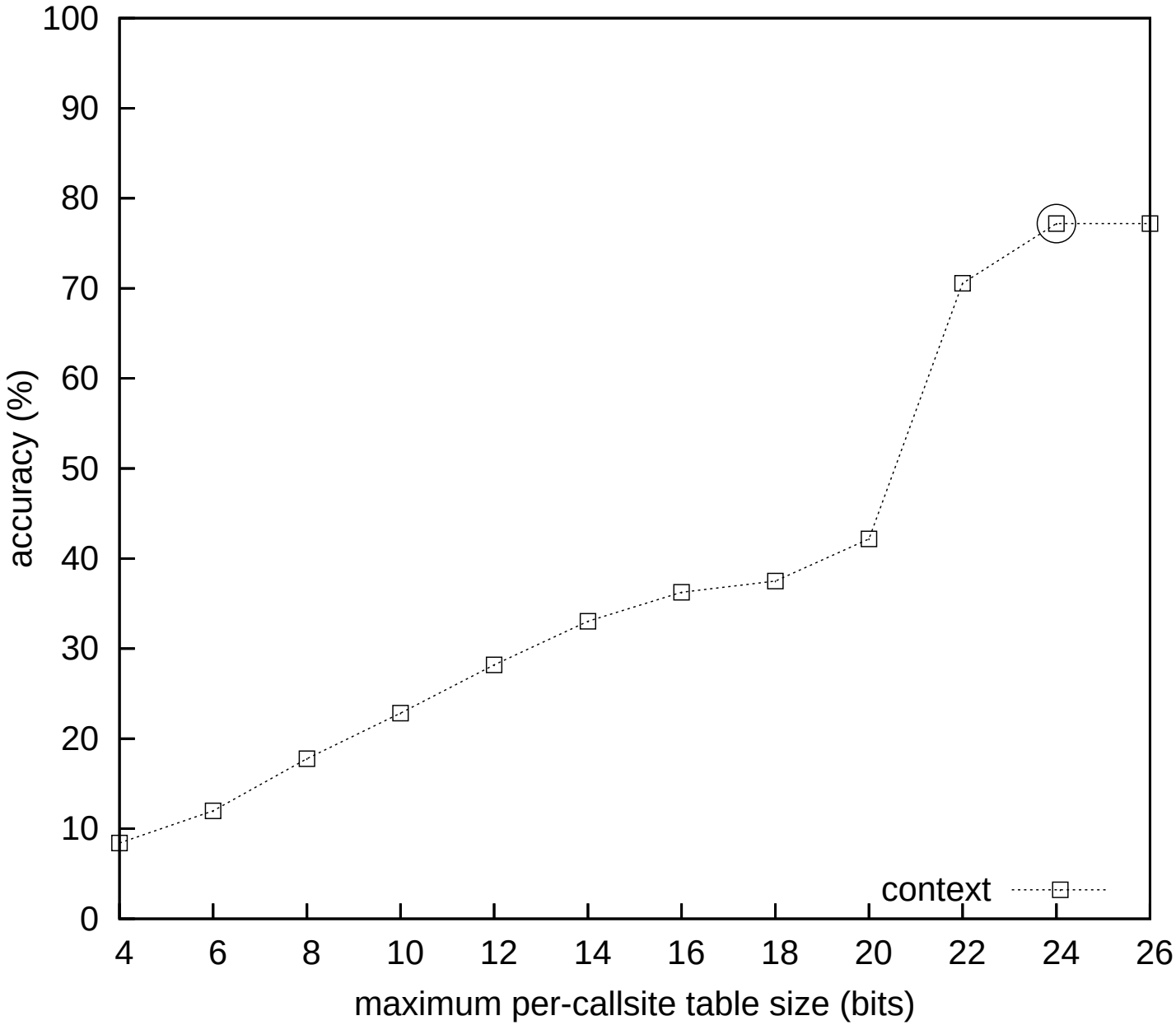
# SPEC JVM98 Dynamic Properties

| property | comp | db | jack | javac | jess | mpeg | mtrt |
|---|---|---|---|---|---|---|---|
| callsites | 1.72K | 1.89K | 3.60K | 5.12K | 3.04K | 2.17K | 2.90K |
| forked | 226M | 170M | 59.4M | 127M | 125M | 111M | 288M |
| aborted | 36 | 18 | 608K | 41.8K | 290 | 114 | 62 |
| void | 93.4M | 54.4M | 24.4M | 45.3M | 23.3M | 34.1M | 20.5M |
| verified | 133M | 115M | 34.4M | 81.5M | 102M | 76.9M | 267M |
| boolean Z | 0% | 10% | 27% | 21% | 35% | 32% | 1% |
| byte B | 0% | 0% | 2% | 0% | 0% | 0% | 0% |
| char C | 0% | 0% | 5% | 5% | 0% | 0% | 0% |
| short S | 0% | 0% | 0% | 0% | 0% | 23% | 0% |
| int I | 100% | 42% | 33% | 45% | 20% | 45% | 2% |
| long J | 0% | 0% | 4% | 1% | 0% | 0% | 0% |
| float F | 0% | 0% | 0% | 0% | 0% | 0% | 61% |
| double D | 0% | 0% | 0% | 0% | 2% | 0% | 0% |
| reference R | 0% | 49% | 30% | 28% | 43% | 0% | 37% |

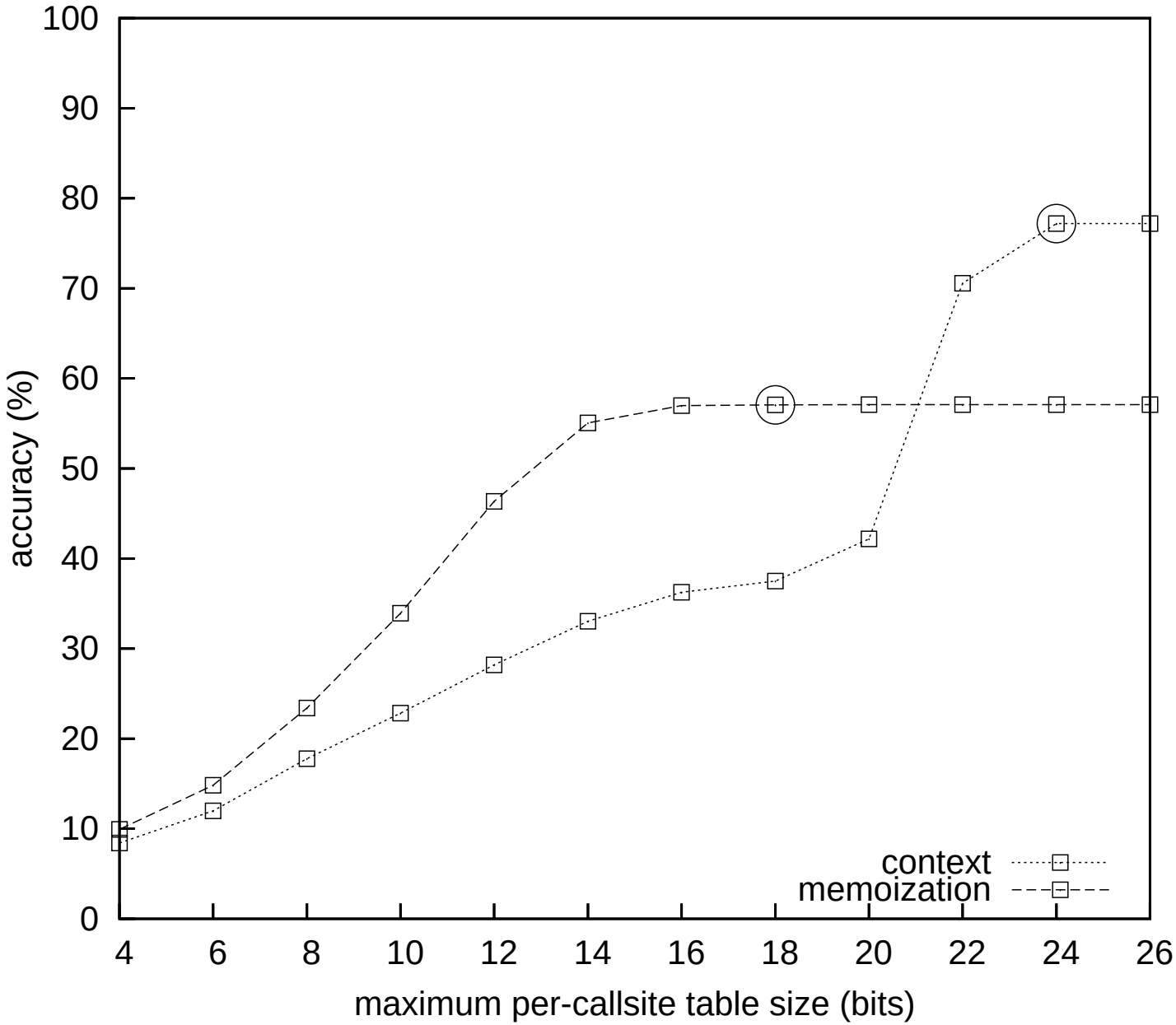
# Size Variation

- Vary hashtable maximum size from 4 to 26 bits
- Graph accuracy against size for:
  - Context
  - Memoization
  - LS2PCM hybrid (all sub-predictors)
- Choose optimal points for context and memoization
  - Use these in future hybrid experiments
  - Future: try to do this profiling dynamically

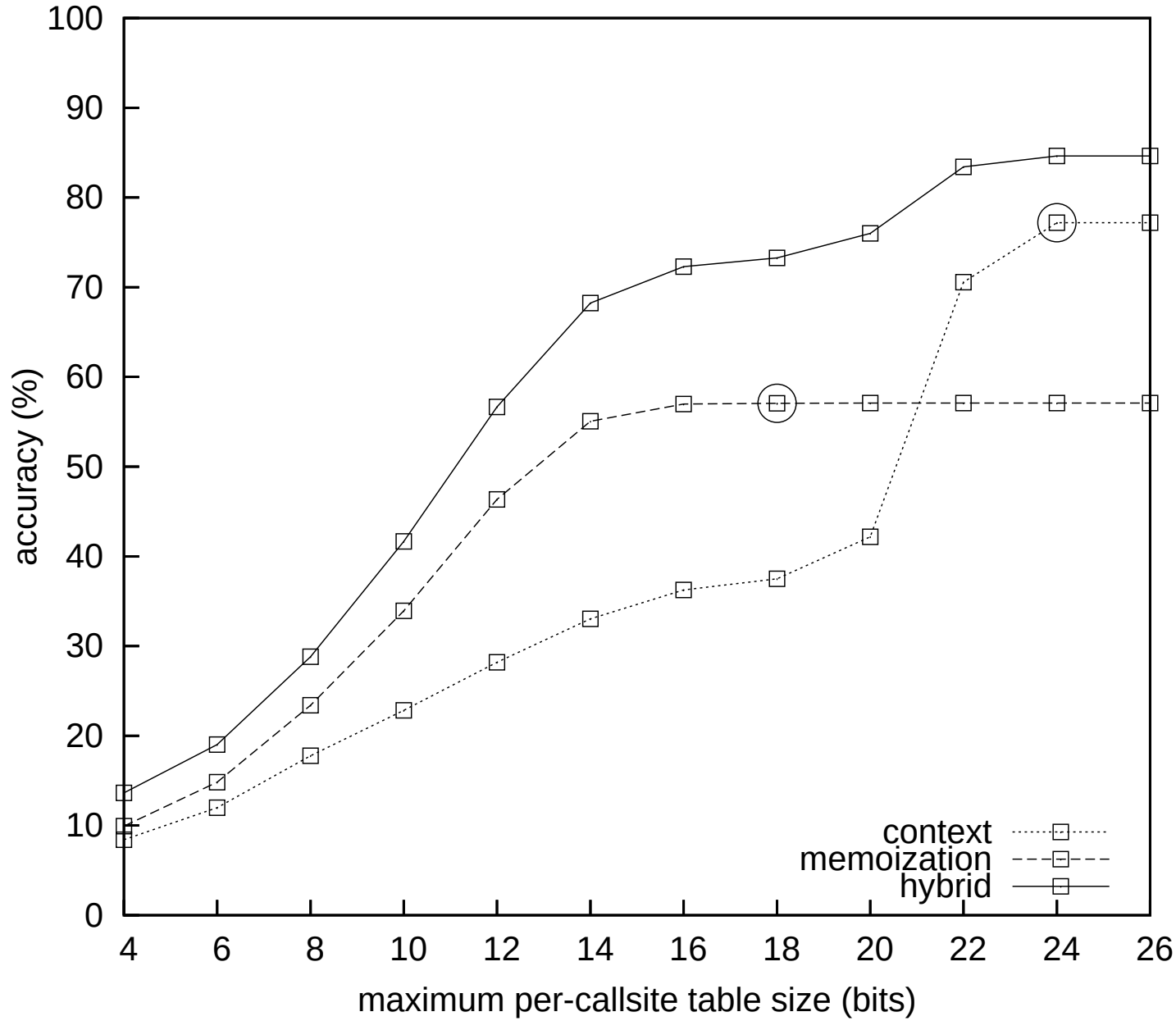
# comp Size Variation



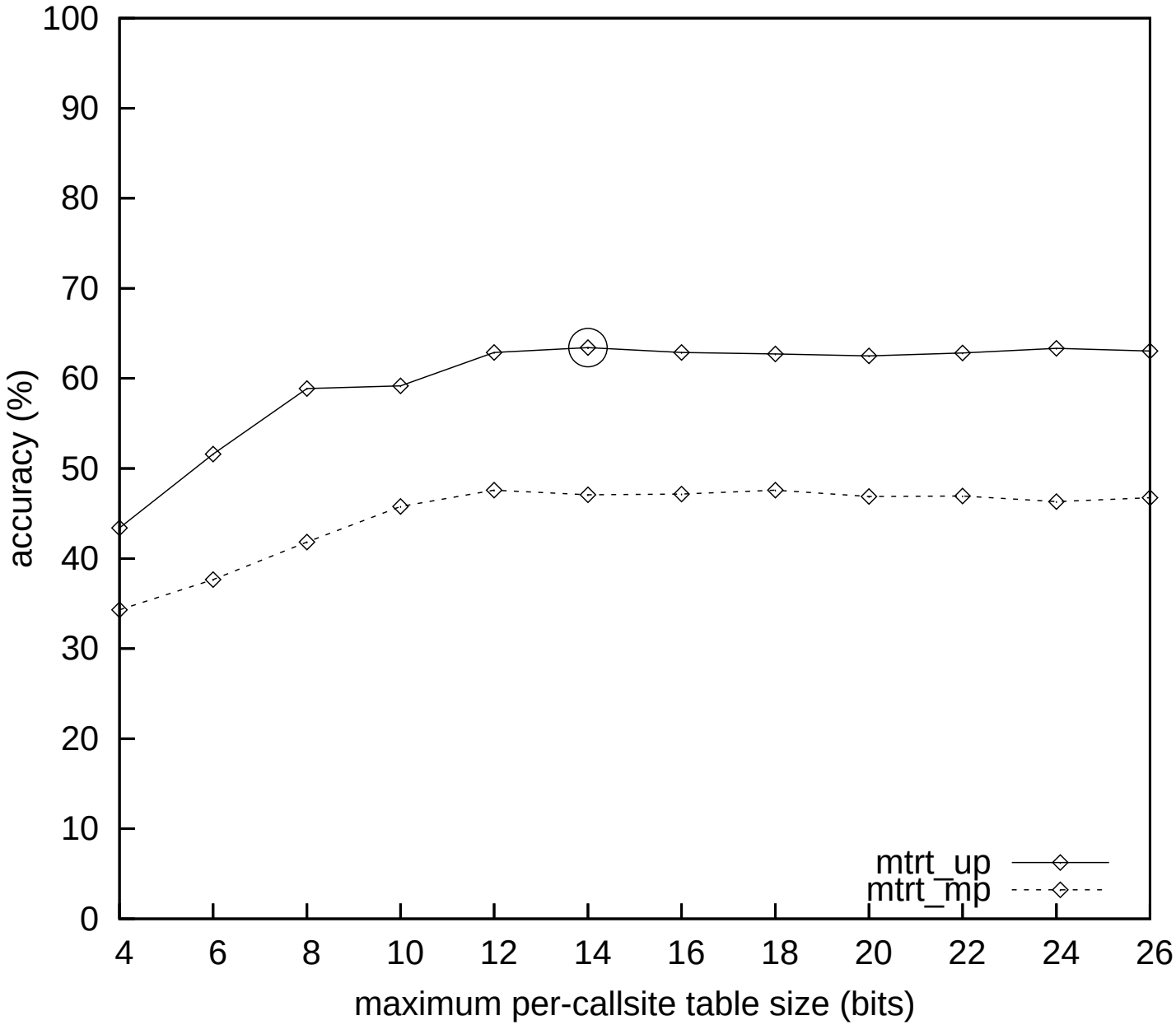
# comp Size Variation



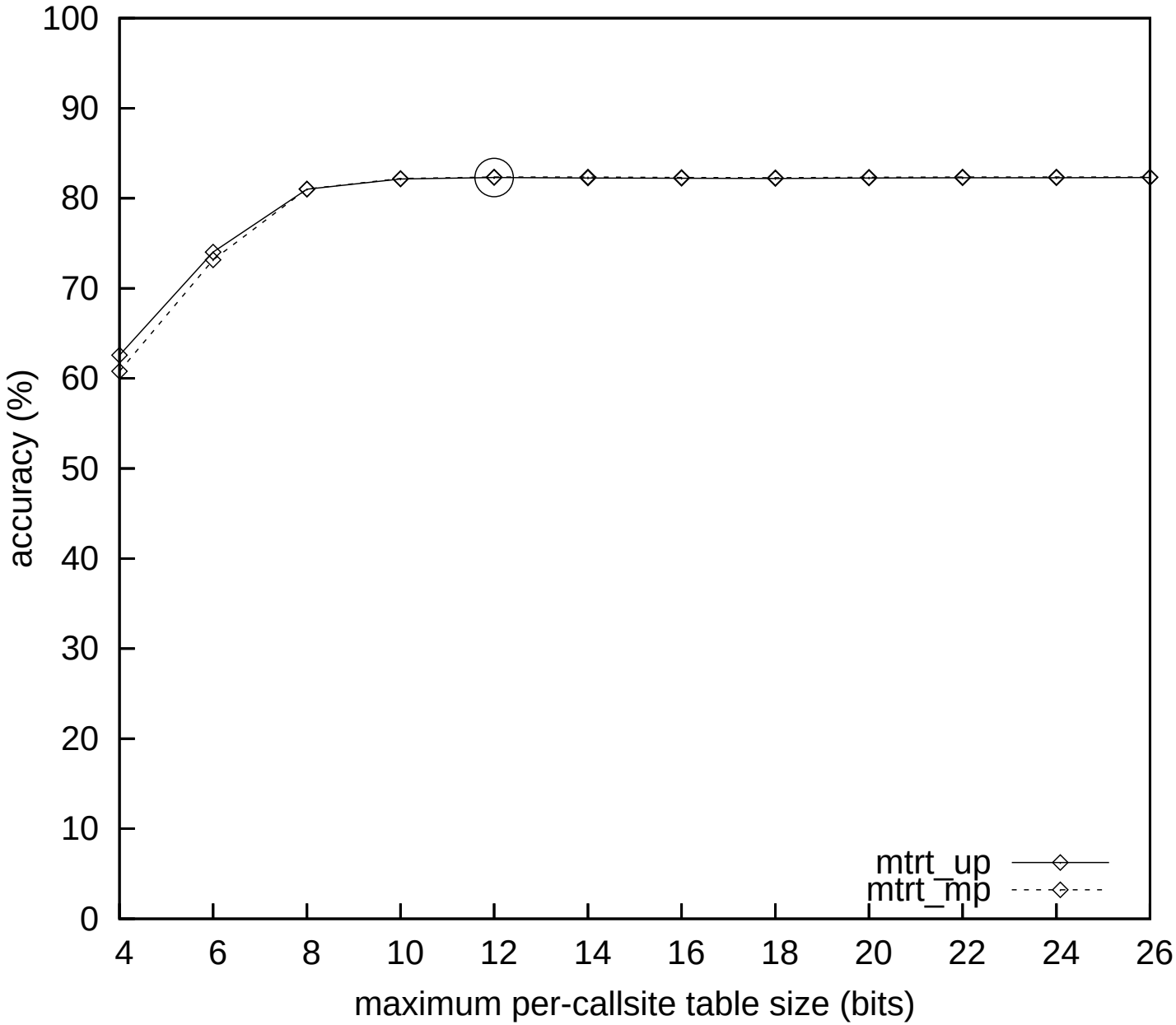
# comp Size Variation



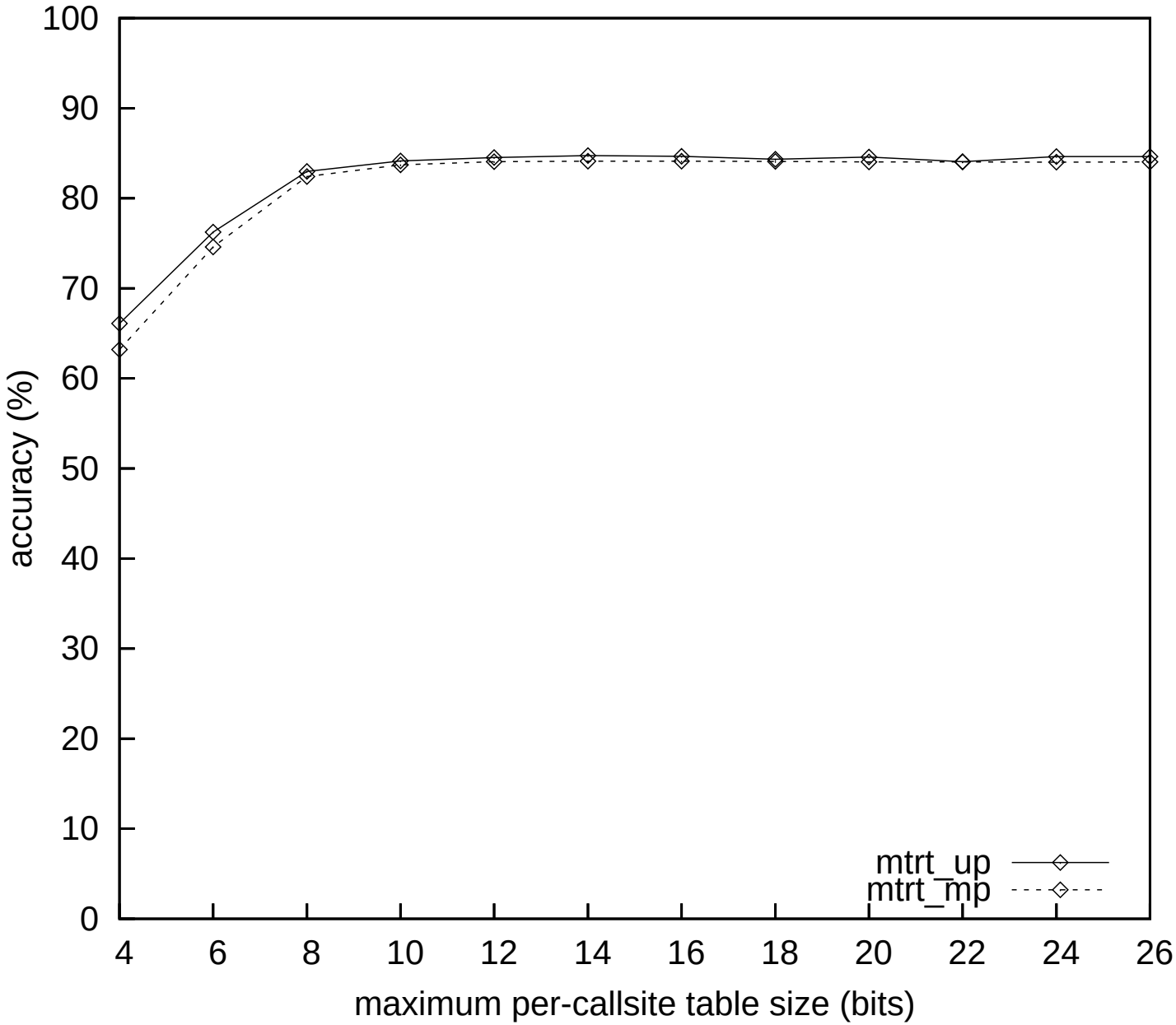
# `mtrt` Context Size Variation



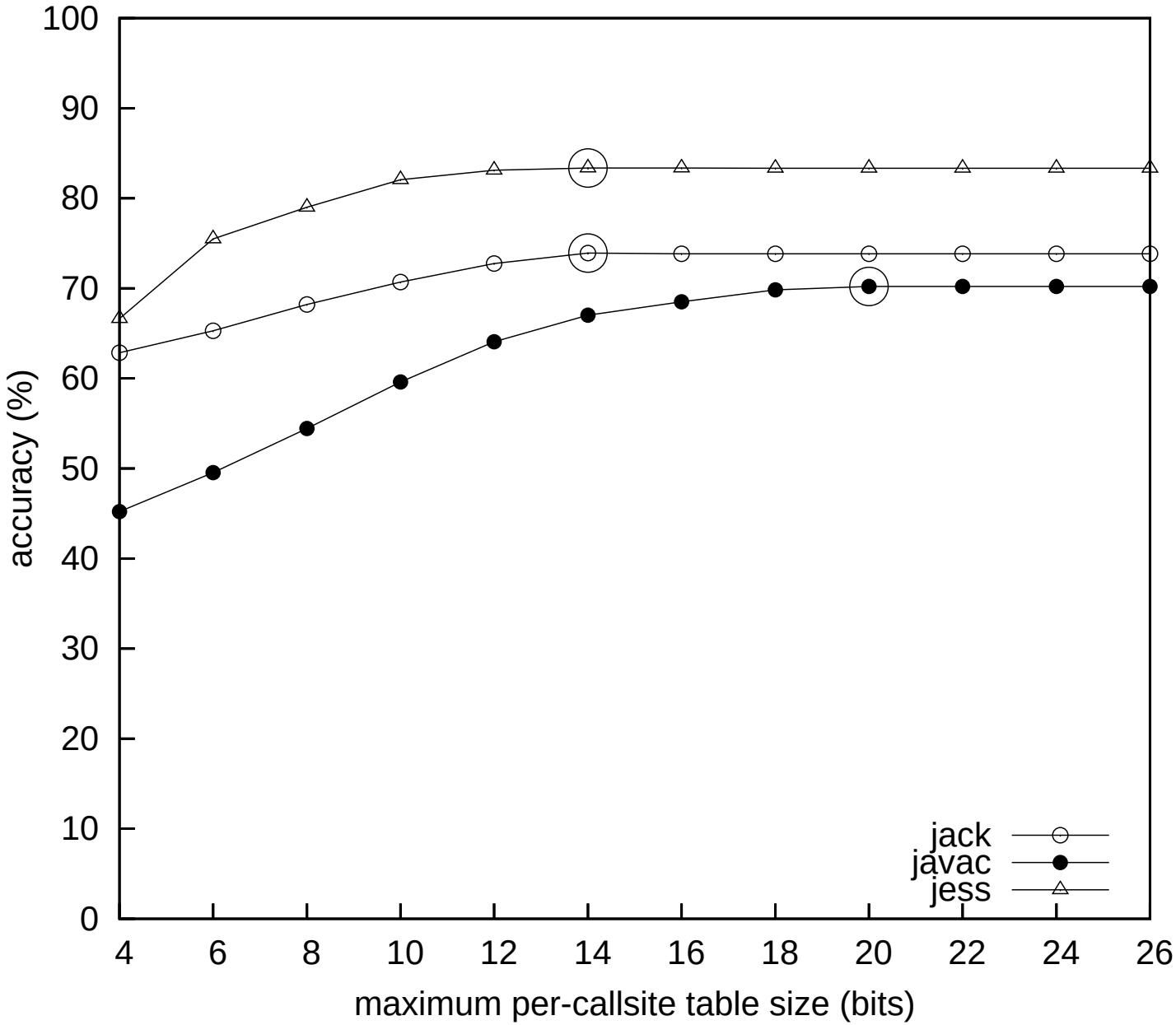
# `mtrt` Memoization Size Variation



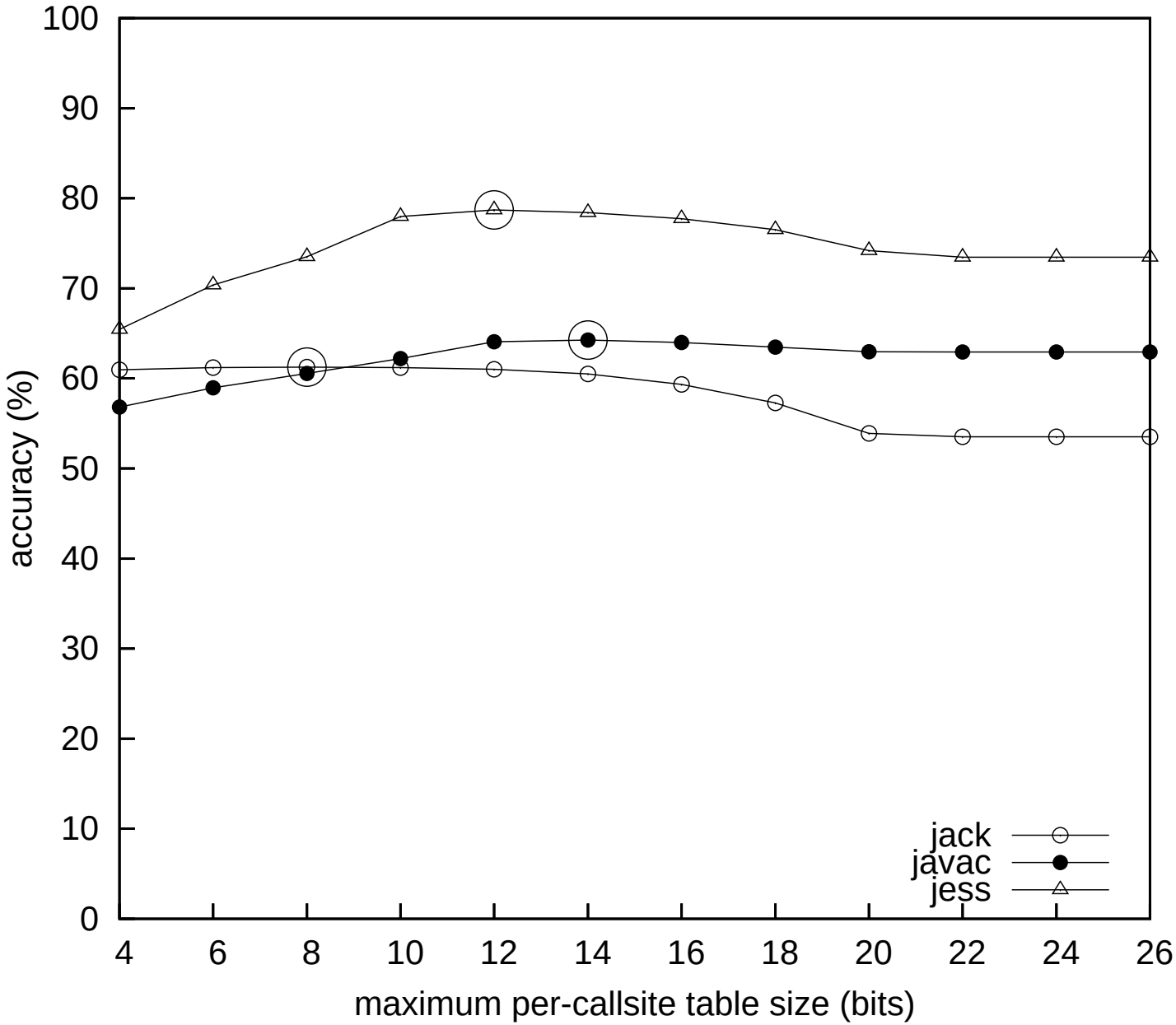
# `mtrt` Hybrid Size Variation



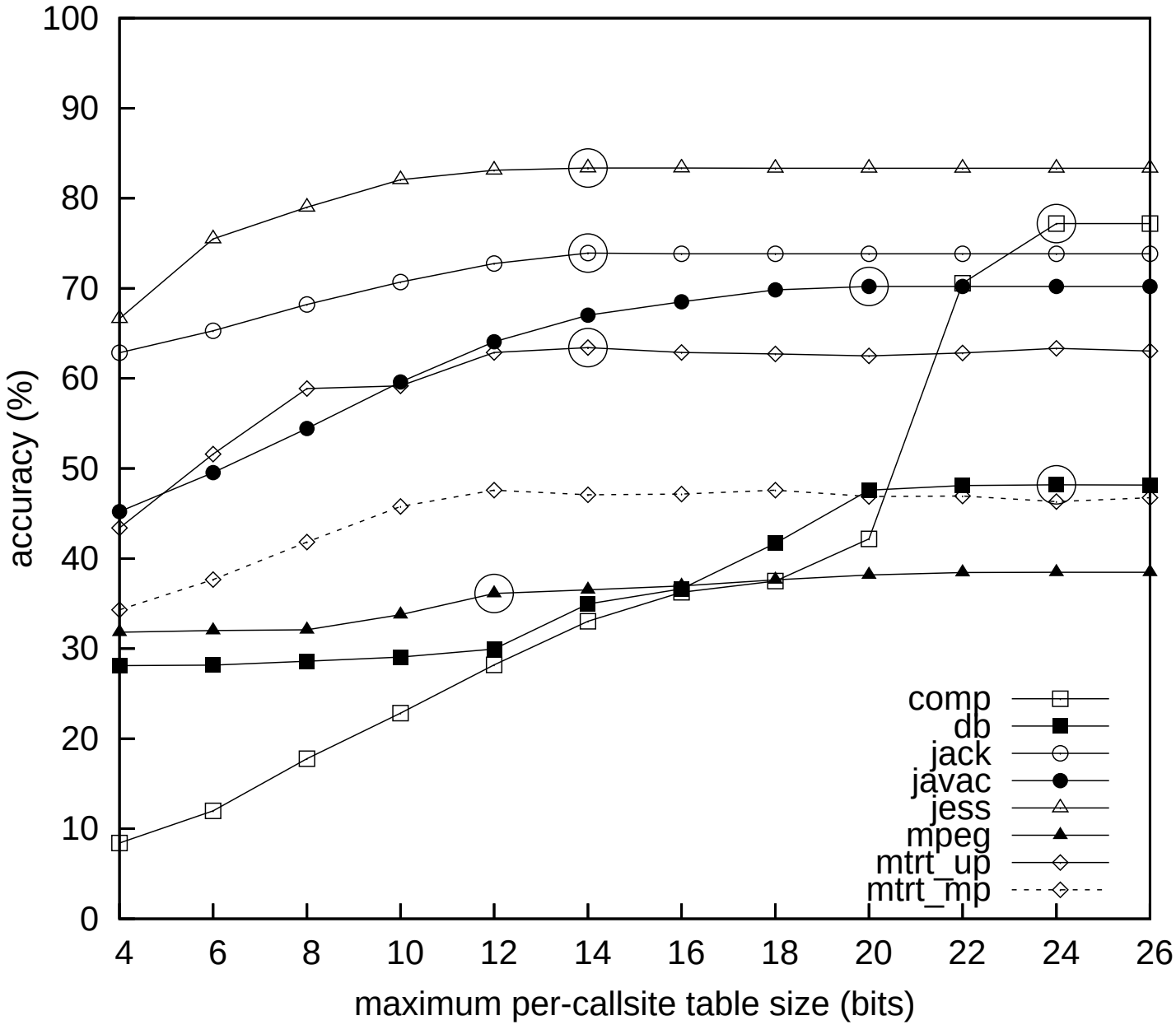
# {jack,javac,jess} Context



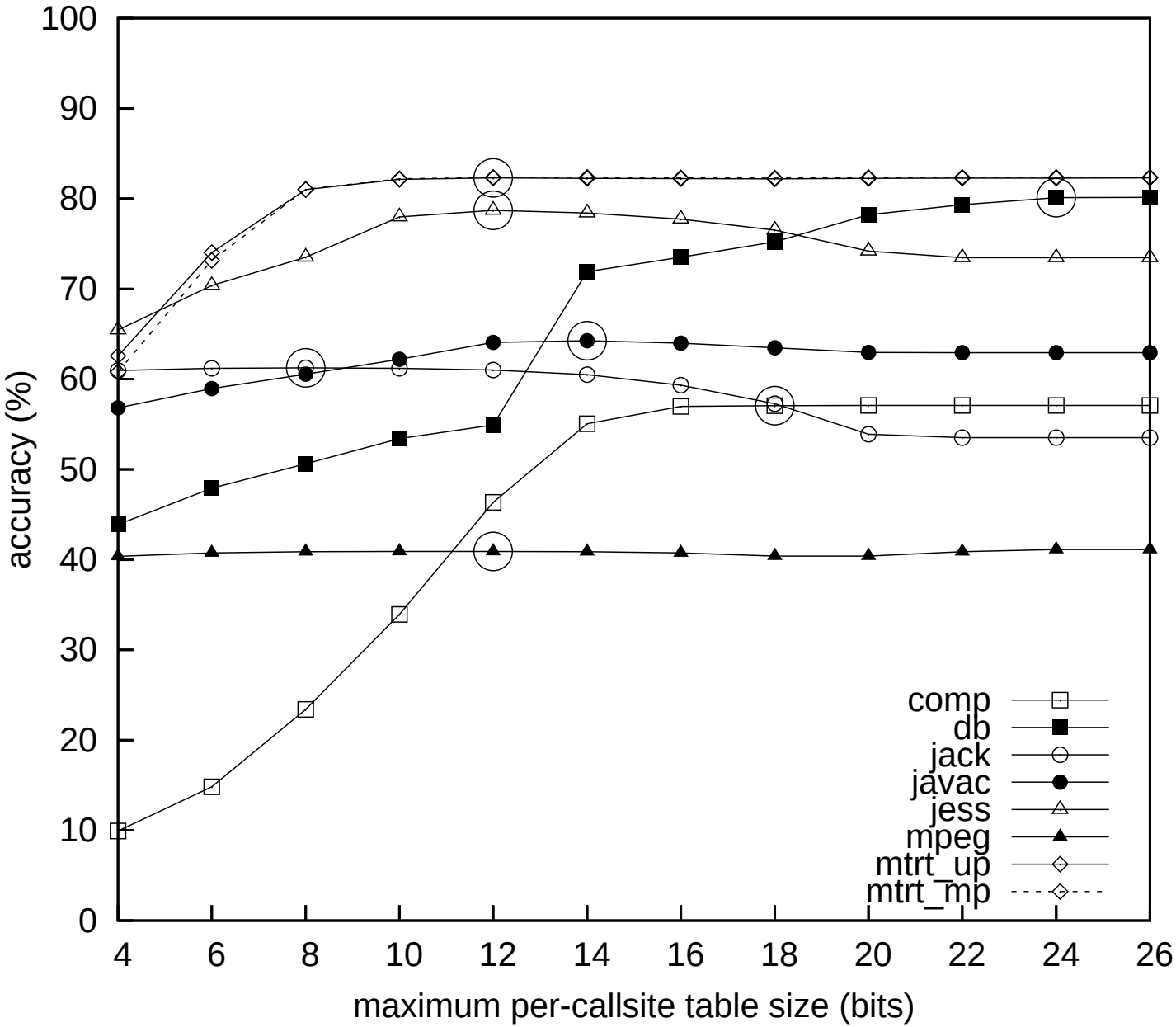
# {jack,javac,jess} Memoization



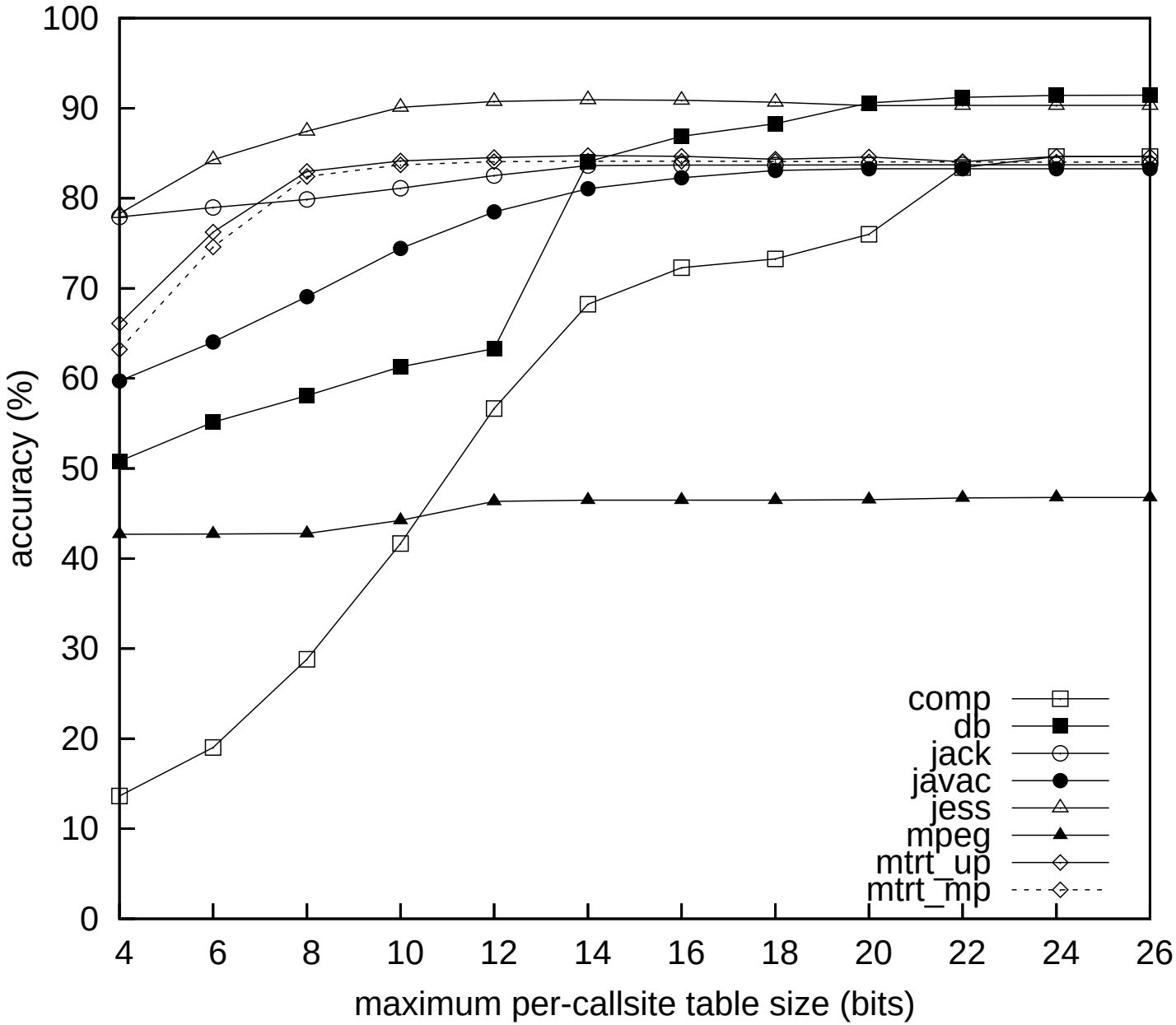
# Context Size Variation (all)



# Memoization Size Variation (all)



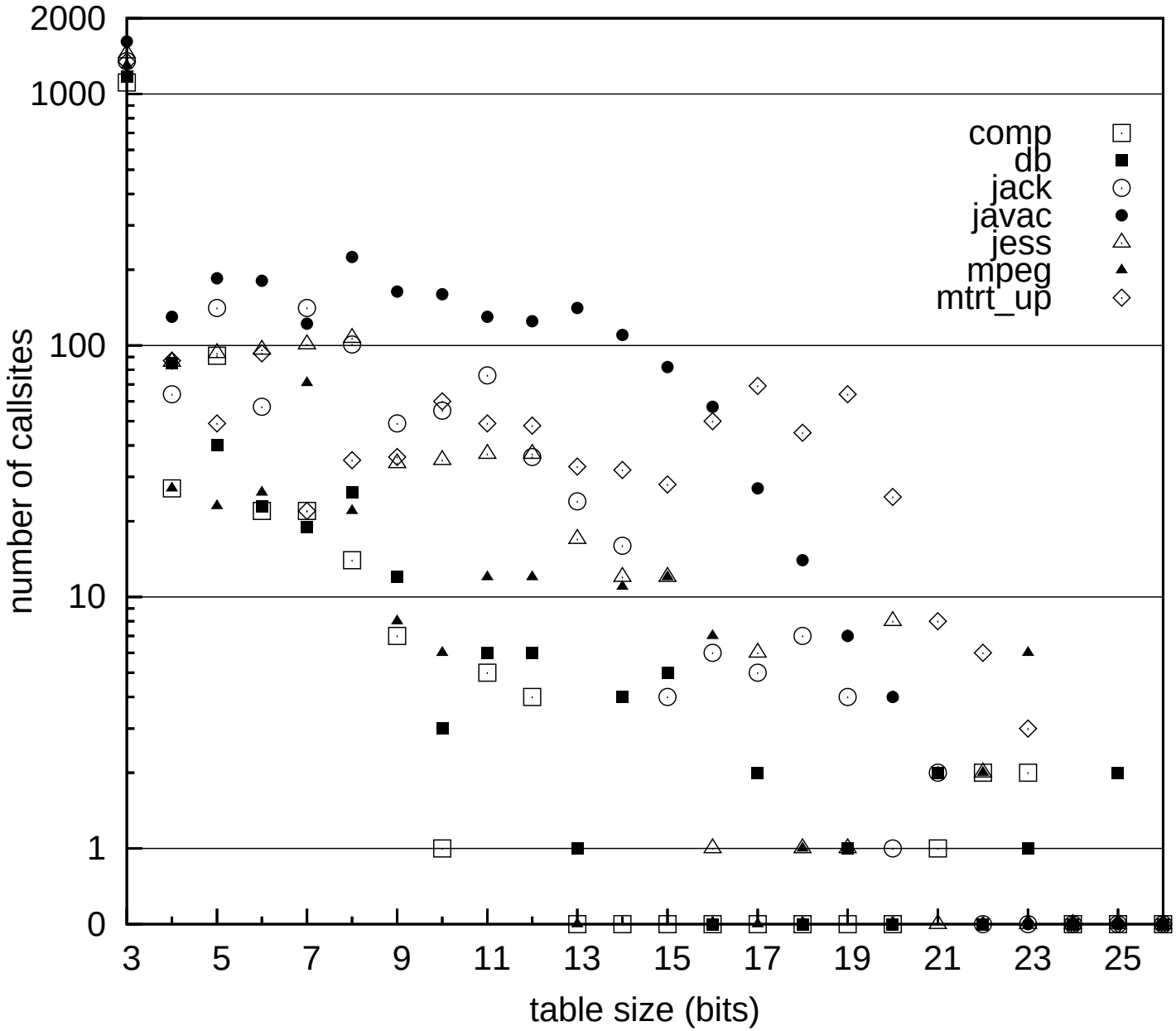
# Hybrid Size Variation (all)



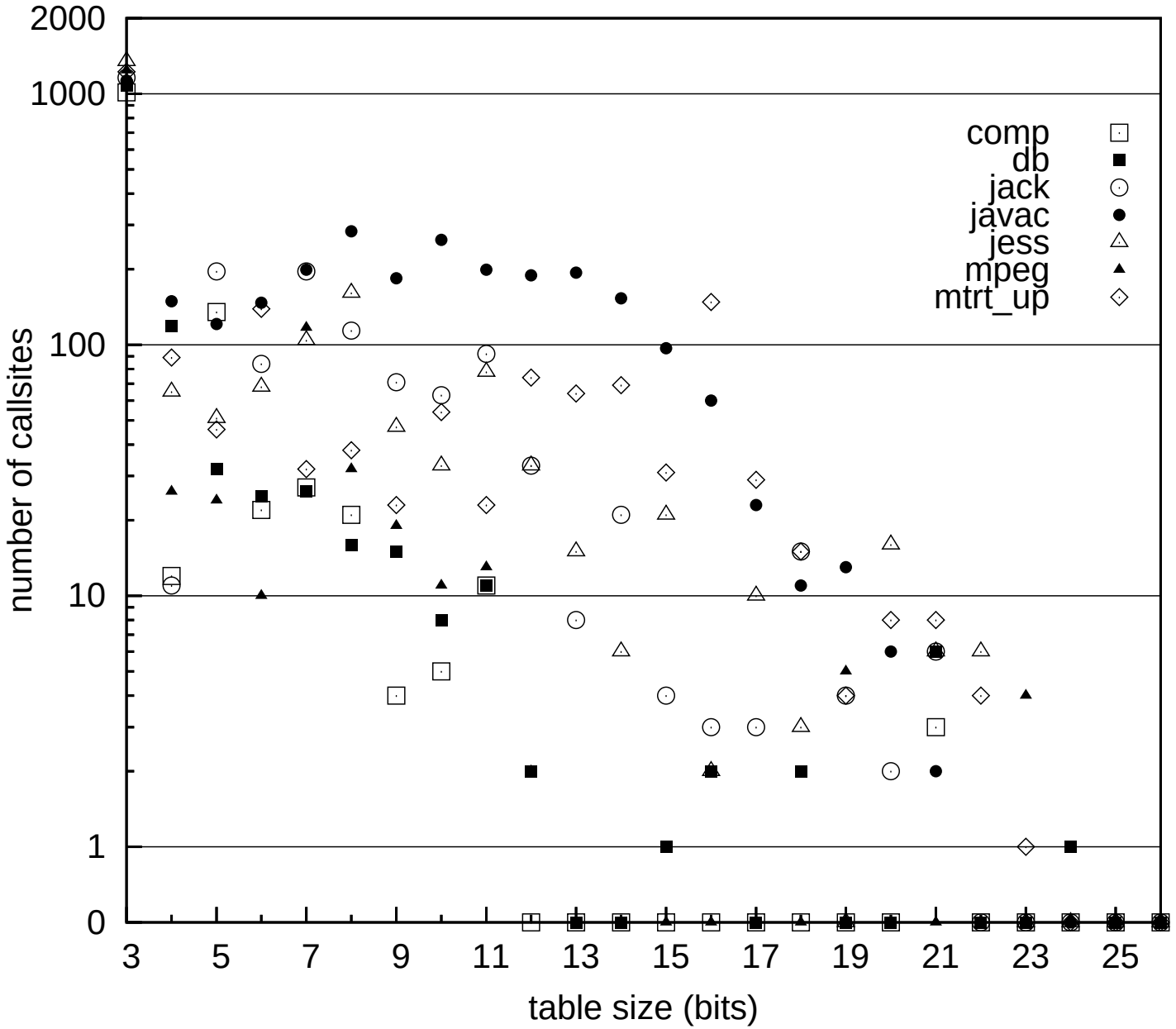
# Memory Usage

- Allow hashtables to expand freely up to 26 bits
  - Look at final distributions
  - On average, 87% never expand beyond 8 bits
  - Expansion $\alpha$ hash function input variability
- Exploit VM level value width info to conserve memory
  - Compare against using full 64-bit table values
- Memoization requires less space than context
  - Indicates suitability for hardware designs

# Context Table Distribution



# Memoization Table Distribution



# Table Memory

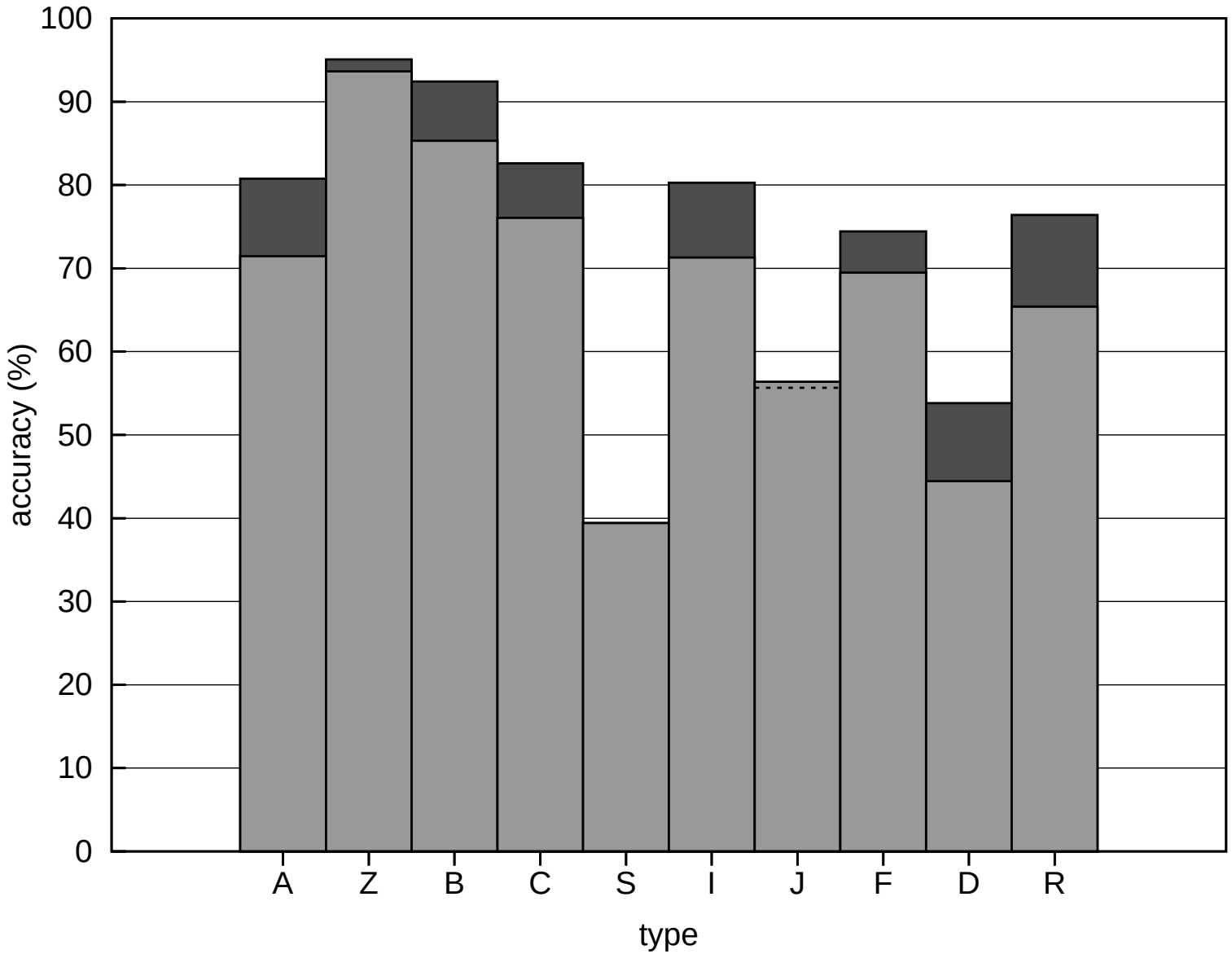
| bench-mark | context | | | memoization | | |
|---|---|---|---|---|---|---|
| | size | original | reduced | size | original | reduced |
| `comp` | 24 | 313M | 208M | 18 | 9.60M | 6.38M |
| `db` | 24 | 541M | 361M | 24 | 345M | 206M |
| `jack` | 14 | 15.8M | 10.7M | 8 | 1.79M | 1.11M |
| `javac` | 20 | 291M | 195M | 14 | 103M | 64.5M |
| `jess` | 14 | 13.5M | 9.59M | 12 | 8.83M | 5.62M |
| `mpeg` | 12 | 3.72M | 2.49M | 12 | 1.46M | 856K |
| `mtrt` | 14 | 69.4M | 46.4M | 12 | 23.0M | 15.3M |
| average | 17 | 178M | 119M | 14 | 70.4M | 42.8M |

- value width optimizations yield 35% space reduction

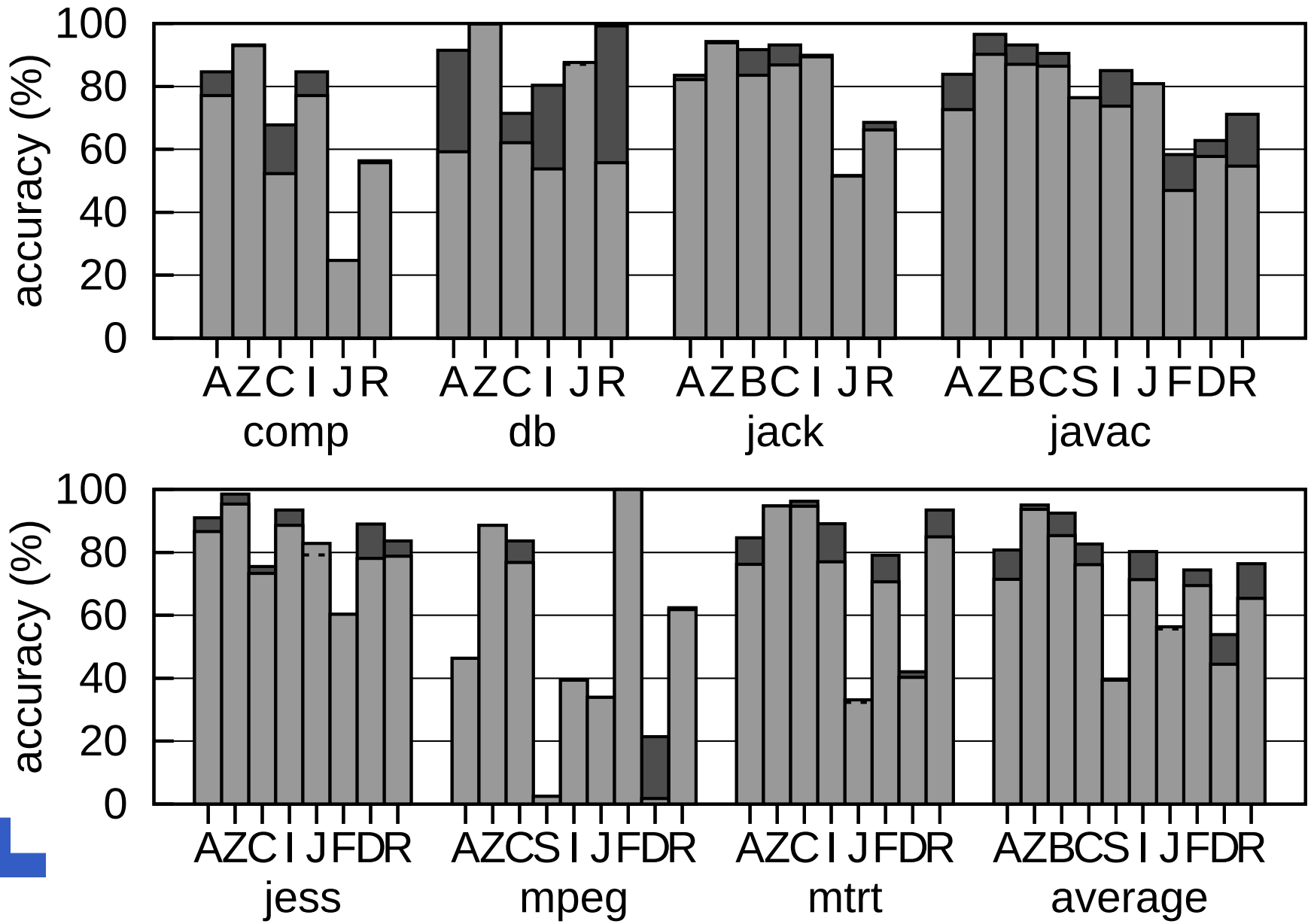
# Hybrid Performance

- Compare hybrid LS2PC and LS2PCM predictors
- Omit types with zero calls
- Difficult to compare with directly with Hu's results
  - S100 vs. S1
  - non-inlined vs. inlined
- Memoization complements context nicely in hybrid
  - 72% average accuracy with LS2PC
  - 81% average accuracy with LS2PCM
- Hybrid fitness: 96%
  - Ability to capture correct sub-predictions

# LS2PC (light) vs. LS2PCM (dark)



# LS2PC (light) vs. LS2PCM (dark)



# Conclusions

- Reported comprehensive data over all method calls
- Achieved high prediction accuracy in software JVM
- Introduced powerful memoization predictor
- Cut memory costs without sacrificing accuracy

# Future Work

- Extend framework with new predictors (e.g. DFCM)
- Generalised load predictors
- Implement compiler analyses in Soot
  - Parameter dependence analysis
  - Return value use analysis
- Determine extent to which memoization compensates for concurrent update problems with context predictors
- Expand hashtables only if accuracy increased on last expansion
- Finish SMLP implementation in SableVM
  - Study costs and benefits of RVP in this system