Tail Call Elimination in Opensmalltalk

Matthew Ralston   Dave Mason
Department of Computer Science
Ryerson University

RYERSON UNIVERSITY

Agenda

- What is a Tail Call?
- Tail Call Elimination
- Stack Interpreter Implementation
- Cog VM JIT Implementation
- Results
- Conclusions and Future Work

What is a Tail Call?

- Call followed by a return
- Smalltalk Example

```
Array class>>new: sizeRequested
   ^ self basicNew: sizeRequested
```

- Call to basicNew: is a tail call
- Immediately followed by a return

What is a Tail Call?

- Call followed by a return
- Smalltalk Example

```
Array class>>new: sizeRequested
   ^ self basicNew: sizeRequested
```

- Call to basicNew: is a tail call
- Immediately followed by a return

What is a Tail Call?

- Call followed by a return
- Smalltalk Example

```
Array class>>new: sizeRequested
   ^ self basicNew: sizeRequested
```

- Call to basicNew: is a tail call
- Immediately followed by a return

What is a Tail Call?

- Call followed by a return
- Smalltalk Example

```
Array class>>new: sizeRequested
   ^ self basicNew: sizeRequested
```

- Call to basicNew: is a tail call
- Immediately followed by a return

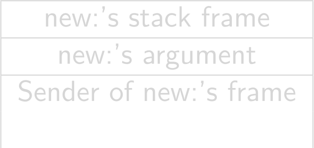
Tail call in Bytecode

- Bytecode for Array class»new:

```
self
pushTemp: 0
send: basicNew:
returnTop
```

Calling new without TCE

- Stack after calling new:



new:'s stack frame

new:'s argument

Sender of new:'s frame

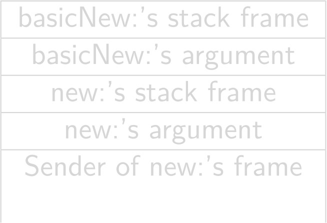
Calling new without TCE

- Stack after calling new:

new:'s stack frame
new:'s argument
Sender of new:'s frame

Calling new without TCE - cont.

- Stack after calling basicNew:



basicNew:'s stack frame

basicNew:'s argument

new:'s stack frame

new:'s argument

Sender of new:'s frame

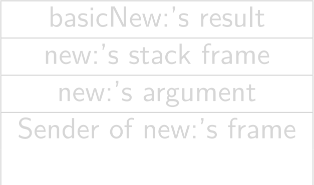
Calling new without TCE - cont.

- Stack after calling basicNew:

basicNew:'s stack frame
basicNew:'s argument
new:'s stack frame
new:'s argument
Sender of new:'s frame

Calling new without TCE - cont.

- Stack after returning from basicNew:



basicNew:'s result

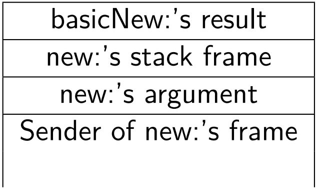
new:'s stack frame

new:'s argument

Sender of new:'s frame

Calling new without TCE - cont.

- Stack after returning from basicNew:



basicNew:'s result

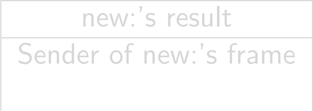
new:'s stack frame

new:'s argument

Sender of new:'s frame

Calling new without TCE - cont.

- Stack after returning from new:

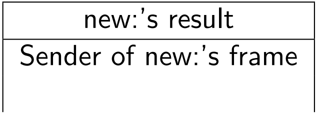


new:'s result

Sender of new:'s frame

Calling new without TCE - cont.

- Stack after returning from new:



new:'s result

Sender of new:'s frame

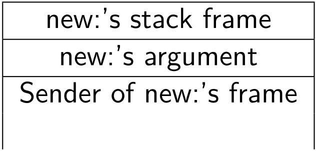
Tail Call Elimination

- Why return to new:?
- Why keep new:'s stack frame?

Tail Call Elimination

- Why return to new:?
- Why keep new:'s stack frame?

Calling new with TCE

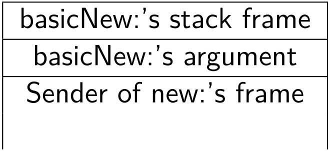


new:'s stack frame

new:'s argument

Sender of new:'s frame

Calling new with TCE - cont'd

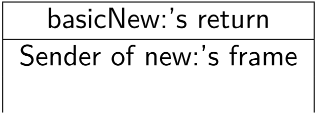


basicNew:'s stack frame

basicNew:'s argument

Sender of new:'s frame

Calling new with TCE - cont'd



basicNew:'s return

Sender of new:'s frame

Tail Recursion Elimination

- Special Case of Tail Call Elimination
- Recursive Call is also a Tail Call

Tail Recursion Elimination

- Special Case of Tail Call Elimination
- Recursive Call is also a Tail Call

Motivation

- Well Known Optimization
- Support in functional languages, CLR, etc.
- Not supported in JVM, Python
- Necessary for functional languages
- Can be useful for OO as well
- In common patterns like Visitor Pattern
- Iteration in Smalltalk

Motivation

- Well Known Optimization
- Support in functional languages, CLR, etc.
- Not supported in JVM, Python
- Necessary for functional languages
- Can be useful for OO as well
- In common patterns like Visitor Pattern
- Iteration in Smalltalk

Motivation

- Well Known Optimization
- Support in functional languages, CLR, etc.
- Not supported in JVM, Python
- Necessary for functional languages
- Can be useful for OO as well
- In common patterns like Visitor Pattern
- Iteration in Smalltalk

Motivation

- Well Known Optimization
- Support in functional languages, CLR, etc.
- Not supported in JVM, Python
- Necessary for functional languages
- Can be useful for OO as well
- In common patterns like Visitor Pattern
- Iteration in Smalltalk

Motivation

- Well Known Optimization
- Support in functional languages, CLR, etc.
- Not supported in JVM, Python
- Necessary for functional languages
- Can be useful for OO as well
- In common patterns like Visitor Pattern
- Iteration in Smalltalk

Motivation

- Well Known Optimization
- Support in functional languages, CLR, etc.
- Not supported in JVM, Python
- Necessary for functional languages
- Can be useful for OO as well
- In common patterns like Visitor Pattern
- Iteration in Smalltalk

Motivation

- Well Known Optimization
- Support in functional languages, CLR, etc.
- Not supported in JVM, Python
- Necessary for functional languages
- Can be useful for OO as well
- In common patterns like Visitor Pattern
- Iteration in Smalltalk

Frequency of Tail Calls

- Static Frequency

Platform Packages	Tail Calls	Total	Percentage
Squeak - All	25162	407971	6.17
Squeak - Compiler	863	8747	9.87

- Higher Dynamic Frequency

Action	Tail Calls	Total	Percentage
Startup	47669	219054	21.76
Recompile	92041349	551250016	16.70

Frequency of Tail Calls

- Static Frequency

Platform Packages	Tail Calls	Total	Percentage
Squeak - All	25162	407971	6.17
Squeak - Compiler	863	8747	9.87

- Higher Dynamic Frequency

Action	Tail Calls	Total	Percentage
Startup	47669	219054	21.76
Recompile	92041349	551250016	16.70

Implementations

- Stack Interpreter
- Cog VM

Implementations

- Stack Interpreter
- Cog VM

Stack Interpreter

- Interpreter only
- Look ahead to next bytecode for return
- Switch to tail call eliminating implementation
- Remove the existing stack frame and arguments
- Pushes the new arguments and calls the next method

Stack Interpreter

- Interpreter only
- Look ahead to next bytecode for return
- Switch to tail call eliminating implementation
- Remove the existing stack frame and arguments
- Pushes the new arguments and calls the next method

Stack Interpreter

- Interpreter only
- Look ahead to next bytecode for return
- Switch to tail call eliminating implementation
- Remove the existing stack frame and arguments
- Pushes the new arguments and calls the next method

Stack Interpreter

- Interpreter only
- Look ahead to next bytecode for return
- Switch to tail call eliminating implementation
- Remove the existing stack frame and arguments
- Pushes the new arguments and calls the next method

Stack Interpreter

- Interpreter only
- Look ahead to next bytecode for return
- Switch to tail call eliminating implementation
- Remove the existing stack frame and arguments
- Pushes the new arguments and calls the next method

Cog JIT Compiler

- Not optimizing interpreted calls
- JIT compile tail calls as jumps
- Cost of tail call check moved to JIT compile time

Cog JIT Compiler

- Not optimizing interpreted calls
- JIT compile tail calls as jumps
- Cost of tail call check moved to JIT compile time

Cog JIT Compiler

- Not optimizing interpreted calls
- JIT compile tail calls as jumps
- Cost of tail call check moved to JIT compile time

Cog VM - Inline Caching

- Cog VM - levels of inline caching
- No Inline Cache
- Monomorphic Send Sites
- Polymorphic Send Sites
- Megamorphic Send Sites

Cog VM - Inline Caching

- Cog VM - levels of inline caching
- No Inline Cache
- Monomorphic Send Sites
- Polymorphic Send Sites
- Megamorphic Send Sites

Cog VM - Inline Caching

- Cog VM - levels of inline caching
- No Inline Cache
- Monomorphic Send Sites
- Polymorphic Send Sites
- Megamorphic Send Sites

Cog VM - Inline Caching

- Cog VM - levels of inline caching
- No Inline Cache
- Monomorphic Send Sites
- Polymorphic Send Sites
- Megamorphic Send Sites

Cog VM - Inline Caching

- Cog VM - levels of inline caching
- No Inline Cache
- Monomorphic Send Sites
- Polymorphic Send Sites
- Megamorphic Send Sites

Cog VM - Inline Caching - cont.

- Bypass for unlinked send sites
- Activate for monomorphic send sites
- Bypass for polymorphic and megamorphic send sites
- Need tail call and non-tail call JIT code for each send
- Copy method lookup code to sender in tail calls

Cog VM - Inline Caching - cont.

- Bypass for unlinked send sites
- Activate for monomorphic send sites
- Bypass for polymorphic and megamorphic send sites
- Need tail call and non-tail call JIT code for each send
- Copy method lookup code to sender in tail calls

Cog VM - Inline Caching - cont.

- Bypass for unlinked send sites
- Activate for monomorphic send sites
- Bypass for polymorphic and megamorphic send sites
- Need tail call and non-tail call JIT code for each send
- Copy method lookup code to sender in tail calls

Cog VM - Inline Caching - cont.

- Bypass for unlinked send sites
- Activate for monomorphic send sites
- Bypass for polymorphic and megamorphic send sites
- Need tail call and non-tail call JIT code for each send
- Copy method lookup code to sender in tail calls

Cog VM - Inline Caching - cont.

- Bypass for unlinked send sites
- Activate for monomorphic send sites
- Bypass for polymorphic and megamorphic send sites
- Need tail call and non-tail call JIT code for each send
- Copy method lookup code to sender in tail calls

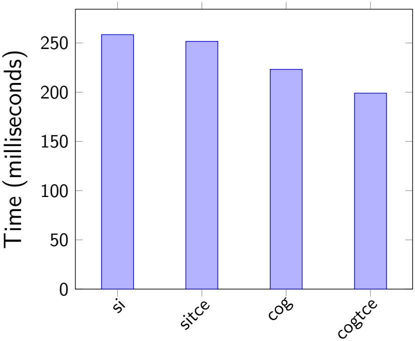
Results

- Tail Recursive Tests
- Real World Tests

Results

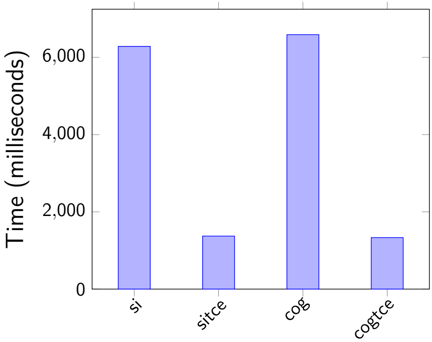
- Tail Recursive Tests
- Real World Tests

Factorial 500 x 1000



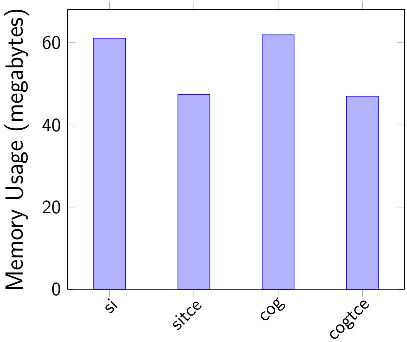
Version	Mean	%Imp	%SD	Std Dev	Median
si	258.43			1.41	258.00
sitce	251.61	2.6	0.7	1.02	251.00
cog	223.18			15.46	217.00
cogtce	199.01	10.8	6.9	1.04	199.00

Factorial 5000



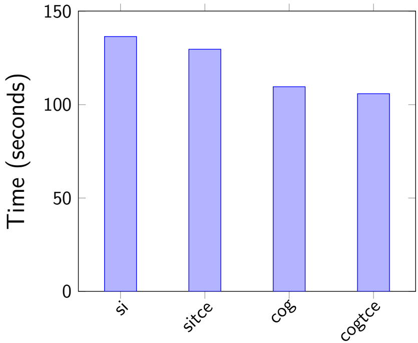
Version	Mean	%Imp	%SD	Std Dev	Median
si	6284.60			12.52	6283.00
sitce	1372.40	78.2	0.4	21.88	1356.00
cog	6587.72			16.45	6591.00
cogtce	1333.28	79.8	0.5	27.36	1314.00

Factorial 5000 Memory



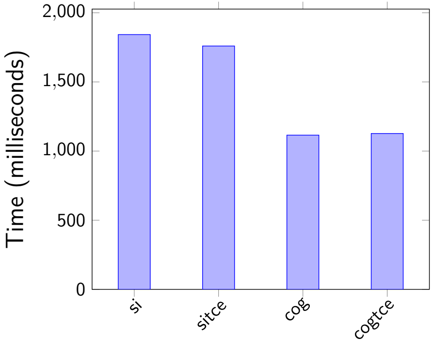
Version	Mean	%Imp	%SD	Std Dev	Median
si	61.11			3.28	60.33
sitce	47.37	22.5	7.7	3.35	47.22
cog	61.92			6.17	59.50
cogtce	46.99	24.1	11.3	3.33	46.89

Compile All Execution



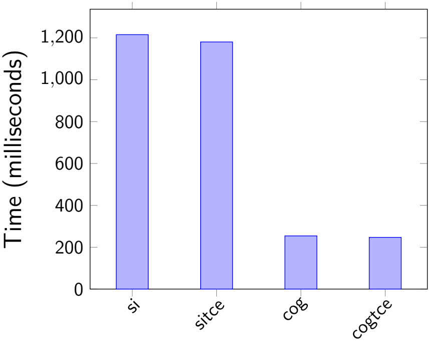
Version	Mean	%Imp	%SD	Std Dev	Median
si	136.49			2.30	135.52
sitce	129.68	5.0	1.7	0.33	129.60
cog	109.59			0.90	109.29
cogtce	105.86	3.4	0.9	0.45	105.84

Browse Number Class Execution



Version	Mean	%Imp	%SD	Std Dev	Median
si	1842.60			51.99	1850.00
sitce	1759.90	4.5	4.2	56.60	1754.00
cog	1115.40			62.26	1124.00
cogtce	1127.20	-1.1	7.5	55.21	1124.00

Method Analyzer Execution



Version	Mean	%Imp	%SD	Std Dev	Median
si	1215.38			29.02	1200.00
sitce	1180.62	2.9	3.1	23.67	1169.00
cog	254.14			38.37	243.00
cogtce	247.10	2.8	21.1	37.33	237.00

Conclusions

- Significant improvements in execution time for tail recursive cases
- Improvements in execution time for most general applications
- Memory usage is only significantly affected for deep recursive call chains
- Stack Interpreter outperforms Cog in some tests
- Stack Interpreter supports polymorphic calls
- Increased JIT compiled method size leads to overhead

Conclusions

- Significant improvements in execution time for tail recursive cases
- Improvements in execution time for most general applications
- Memory usage is only significantly affected for deep recursive call chains
- Stack Interpreter outperforms Cog in some tests
- Stack Interpreter supports polymorphic calls
- Increased JIT compiled method size leads to overhead

Conclusions

- Significant improvements in execution time for tail recursive cases
- Improvements in execution time for most general applications
- Memory usage is only significantly affected for deep recursive call chains
- Stack Interpreter outperforms Cog in some tests
- Stack Interpreter supports polymorphic calls
- Increased JIT compiled method size leads to overhead

Conclusions

- Significant improvements in execution time for tail recursive cases
- Improvements in execution time for most general applications
- Memory usage is only significantly affected for deep recursive call chains
- Stack Interpreter outperforms Cog in some tests
- Stack Interpreter supports polymorphic calls
- Increased JIT compiled method size leads to overhead

Conclusions

- Significant improvements in execution time for tail recursive cases
- Improvements in execution time for most general applications
- Memory usage is only significantly affected for deep recursive call chains
- Stack Interpreter outperforms Cog in some tests
- Stack Interpreter supports polymorphic calls
- Increased JIT compiled method size leads to overhead

Conclusions

- Significant improvements in execution time for tail recursive cases
- Improvements in execution time for most general applications
- Memory usage is only significantly affected for deep recursive call chains
- Stack Interpreter outperforms Cog in some tests
- Stack Interpreter supports polymorphic calls
- Increased JIT compiled method size leads to overhead

Conclusions and Future Work

- Support polymorphic caches (partially complete!)
- Reduce redundant code generation for Cog

Conclusions and Future Work

- Support polymorphic caches (partially complete!)
- Reduce redundant code generation for Cog

Questions?