The State of TclQuadcode

Kevin B. Kenny
Donal K. Fellows
Tcl Core Team

24th Annual Tcl/Tk Conference
16-20 October 2017

QUAD
CODE

What TclQuadcode is:

Native code compiler for Tcl

- Procedures only
- Not yet methods, λ-forms
- Probably never global scripts

Running ahead of time

- Too slow for JIT!

Using advanced technology

- Many recent papers
- Data flow analysis in Static Single Assigment (SSA)

Multi-year collaboration

- Kevin Kenny, Donal Fellows, Jos Decoster, others

45k lines of Tcl, 3k lines of C++

- And ≈10k lines of generated code

Still a work in progress

- But a piece of software is never "done!"

Why TclQuadCode?

Bytecode interpreter is too slow

- Delicate: changes make it slower!
- Unmaintainable: maze of goto
- Close to achievable speed

Making it much faster needs native code.

Discussed among Tcl'ers for years

- Donal Fellows
- Kevin Kenny
- Don Porter
- Miguel Sofer
- Jos Decoster
- Others...

Very hard problem

Limited time to devote

Getting started

2010: Ozgur Ugurlu (GSoC student) implements bytecode assembler

- Shows that bytecode can be manipulated without compromising safety.

≈2011: Compiler backend embeddings in Tcl appear

- llvm, tcc
- Generate code without leaving Tcl

2012: Karl Lehenbauer issues the FlightAware challenges

- 2× and 10× performance bogeys
- Got everyone moving!

2013: TclQuadcode project launched

Early progress

2014: Kevin studies translation of bytecode to quadcode

- Easier to analyze and manipulate
- Explicit variables rather than stack

Kevin studies data flow analysis

- No SSA yet
- Datalog implemented to aid in difficult analysis
- Datalog paper at Tcl conference pre-announces TclQuadcode

Donal works out translation of quadcode to LLVM IR

- Machine-focused rather than Tcl-focused
- Huge amount of 'glue' needed

Kevin and Donal integrate code at 2014 conference

- Successfully run the first program: [fib]

The long slog

2015: Add bytecode operations and builtin commands, one by one.

Implement SSA and eliminate Datalog

- Datalog not quite fast enough
- SSA enabled analysis with *relatively* simple algorithms

Donal announces project formally at Tcl conference

2016: Largely spend consolidating and refactoring

- Limited developer time

2017: Big gains:

- Node splitting/loop peeling
- Global/namespace variables
- [upvar]
- Near-complete support for ordinary built-in commands (≈200 non-bytecode commands)

QUAD CODE

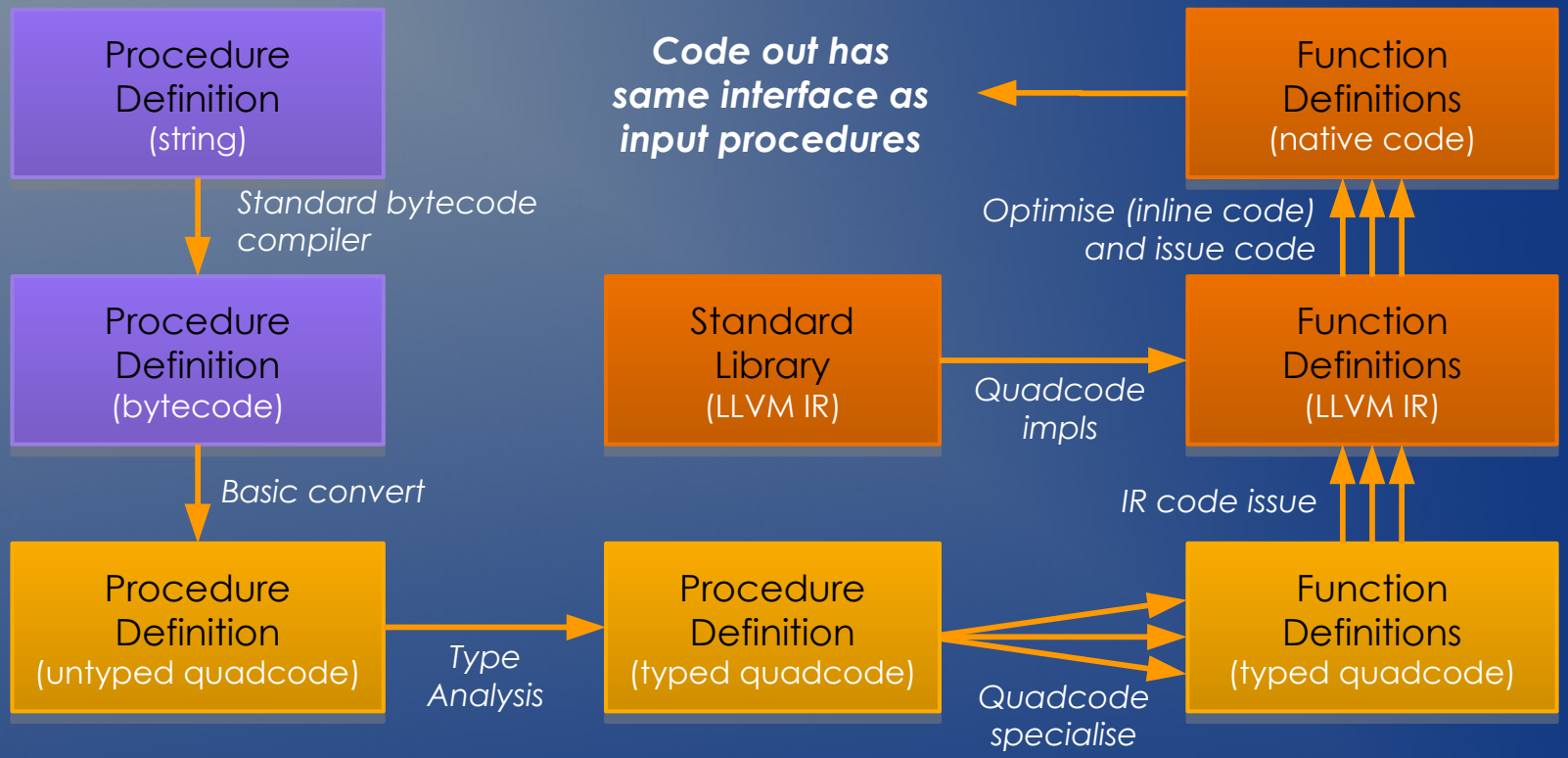
Measured results

Name	Description	Speedup
fib 85	Test simple loops	24.6×
cos 1.2	Test simple floating point	10.9×
wordcounter3 $sentence	Dicts, string operations	5.4×
H9fast $longWord	Compute a hash code on a string	4.9×
mrtest::calc $tree	Recursive tree traversal and arithmetic on nodes	10.8×
impure-caller	Best-case numeric code	66.1×
linesearch::getAllLines2 $size	Larger numeric-intensive code, collinearity testing	10.3×
flightawarebench::test $size	Karl's first benchmark: geographic calculations	15.5×

Typical: 3-6× for general code, 10× and beyond for numeric-intensive code
Little or no speedup for string and I/O operations (Tcl is pretty good at strings)

QUAD CODE

How it works


Procedure Definition (string)
Standard bytecode compiler
Procedure Definition (bytecode)
Basic convert
Procedure Definition (untyped quadcode)
Type Analysis
Procedure Definition (typed quadcode)
Quadcode specialise
Function Definitions (typed quadcode)
IR code issue
Function Definitions (LLVM IR)
Quadcode impls
Standard Library (LLVM IR)
Optimise (inline code) and issue code
Function Definitions (native code)
Code out has same interface as input procedures

Why it works

Avoid overheads

- Memory management, type checking, value conversion

Enabled by type analysis

- int64_t, double, bool
- Check with [string is]
- Propagate through operations such as +

Control flow analysis

- Some code paths exclude others
- After [expr {$x + 1}] succeeds, we know $x is numeric!

Cross-procedure analysis

- Including specialization by type
- One implementation always string-based

Path splitting

Path splitting

```
proc x {a} {
    set y 0
    for {set i $a} {$i <= 10} {incr i} {
        incr y $i
    }
    return $y
}
```

Look at x when called from Tcl

- $a is a string
- $i is a string
- ($i <= 10) is complicated
- [incr y $i] has to extract the integer from a Tcl_Obj
- Bottom of loop has to put the integer back in a Tcl_Obj

Path Splitting, continued

y ← 0

i ← $a

complicated: (i > 10)?

is $i numeric?

i ← IntFromObj($i)

y ← $y + $i

i ← $i + 1

i ← NewIntObj($i)

goto

throw error

return $y

Path Splitting, continued

```
y ← 0
i ← $a
complicated: (i > 10)?
is $i numeric?
i ← IntFromObj($i)
y ← $y + $i
i ← $i + 1
i ← NewIntObj($i)
goto
```

```
throw error
```

```
complicated: (i > 10)?
is $i numeric?
i ← IntFromObj($i)
y ← $y + $i
i ← $i + 1
i ← NewIntObj($i)
goto
```

```
return $y
```

Path Splitting, continued

y ← 0

i ← $a

complicated: (i > 10)?

is $i numeric?

i ← IntFromObj($i)

y ← $y + $i

i ← $i + 1

~~i ← NewIntObj($i)~~

goto

throw error

return $y

integer

~~complicated~~: (i > 10)?

~~is $i numeric?~~

~~i ← IntFromObj($i)~~

y ← $y + $i

i ← $i + 1

~~i ← NewIntObj($i)~~

goto

QUAD CODE

Nonlocal Variable Access

What's done:

- [namespace upvar]
- [variable], [global]
- [upvar 1 $arg name] – gets special handling
- [upvar 1 constantName name] – gets special handling
- [upvar $n ...]
- [upvar #0 ...]
- $::path::to::variable

What's not done:

- Non-constant local names
- [upvar #n], n>0
- [upvar 0]
- $namespace::variable

Why?

- Potential to create aliases for local vars
- Aliases wreck assumptions!

Also: Access to nonlocal variables is still slow!

May have to change code to take best advantage

Slower:

```
proc accum {list} {
  global n; global s; global ss

  foreach a $args {
    incr n
    set s [expr {$s + $a}]
    set ss [expr {$ss + $a}]
  }

}
```

Faster:

```
proc accum {list} {
  global n; global s; global ss
  set n_ $n; set s_ $s; set ss_ $ss
  foreach a $args {
    incr n_
    set s_ [expr {$s_ + $a}]
    set ss_ [expr {$ss_ + $a}]
  }
  set n $n_; set s $s_; set ss $ss_
}
```

QUAD CODE

There's still a lot to do!

Long compilation time

- LLVM is slow
- TclQuadcode is slower
  - Written in Tcl

Large generated code volumes

- Many copies of procedures after type specialization
- Long procedures
  - Stresses downstream compiler

Incomplete language support

- Many things we think we know how to do
- Some things are too dynamic to compile
- Interpreter will always be available

Next steps

[uplevel]

- Limited initially to constant scripts and constant args in a caller
- Limited initially to [uplevel 1]

Better alias treatment

- Lift most of the penalty on nonlocal variables

NRE

- Coroutines, unbounded recursion

Non-hacky arrays

- Currently, arrays are implemented as dicts.

Procedure inlining

- May be required for [uplevel]

Get user experience!

QUAD CODE

Would language changes help?

TIP 283: "Fix variable name resolution quirks"

- Ambiguity in how $namespace::variable resolved
- Current behaviour absolutely insane, source of bugs
- Current behaviour also insanely difficult to implement in compiled code

Help from the programmer about aliases and types

- tcl::pragma::type int $value
- tcl::pragma::noalias var1 var2 ...
- Maybe others...

QUAD CODE

tcl::pragma::type

- Works on values, not variables.
- Asserts that at a given point in execution, a value has a given type.
- Throws error on wrong type
- Useful for documenting API's and parameter checking

Simplifies compiled code called from Tcl.

- Forward type analysis on args possible
- Type checking outside loops
- Much less node splitting – simpler and smaller code.

tcl::pragma::noalias

Asserts that a given set of variable names refer to distinct variables

- Can make exceptions for known aliases.
- Throws a runtime error if the constraint is violated
- Useful check – few procs can survive unexpected aliasing!

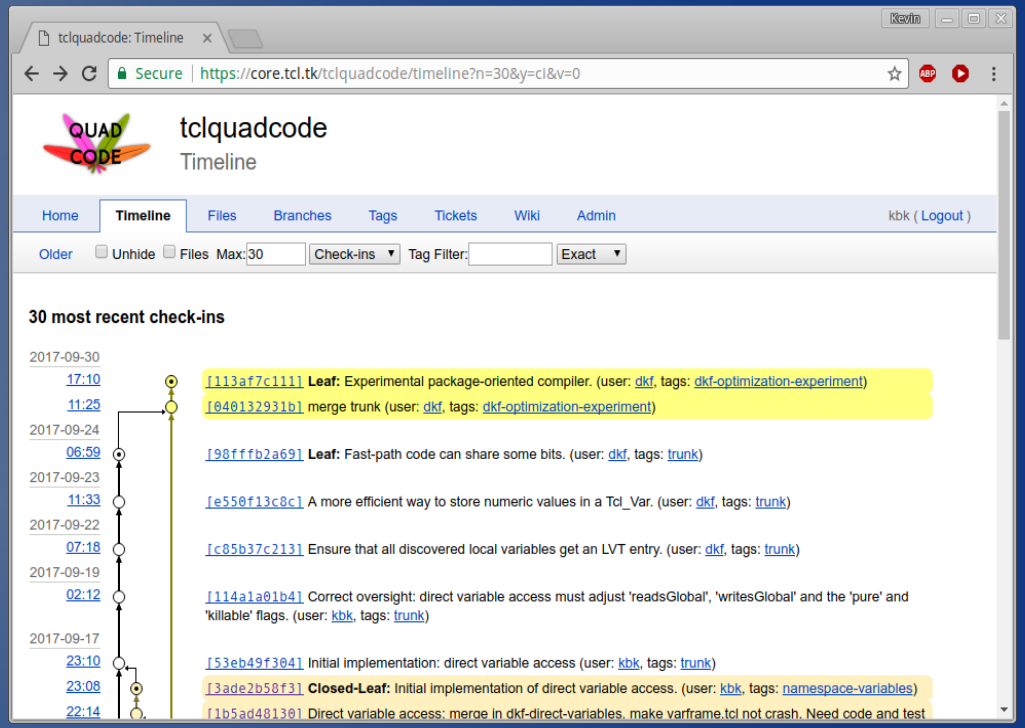
Cannot analyze in general without help – Turing-complete problem!

Can compile much better code

- Uncontrolled aliases are all strings (because types are unknown)
- Changing any potentially aliased variable requires converting all potential aliases back from strings
- Aliasing therefore has pervasive effects.

Thank you!

Where TclQuadcode is:



Source code repository:
https://core.tcl.tk/tclquadcode/

Mailing list:
https://sourceforge.net/p/tcl/mailman/tcl-quadcode/