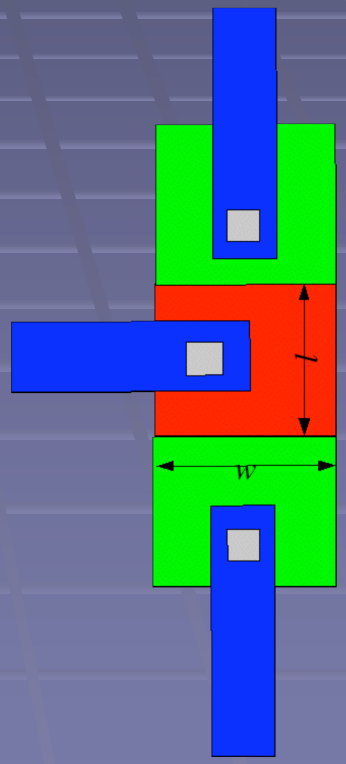
Pulling Out All the Stops

Application of Tcl user extensions in a high performance multi-threaded electronic design application

by Phil Brooks

Calibre Design Verification

- Integrated Circuit Design
- Geometric Analysis Tool

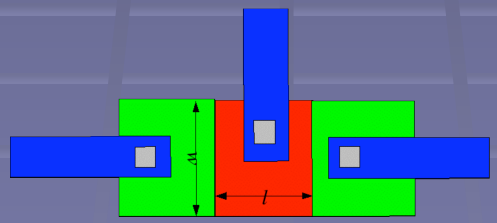


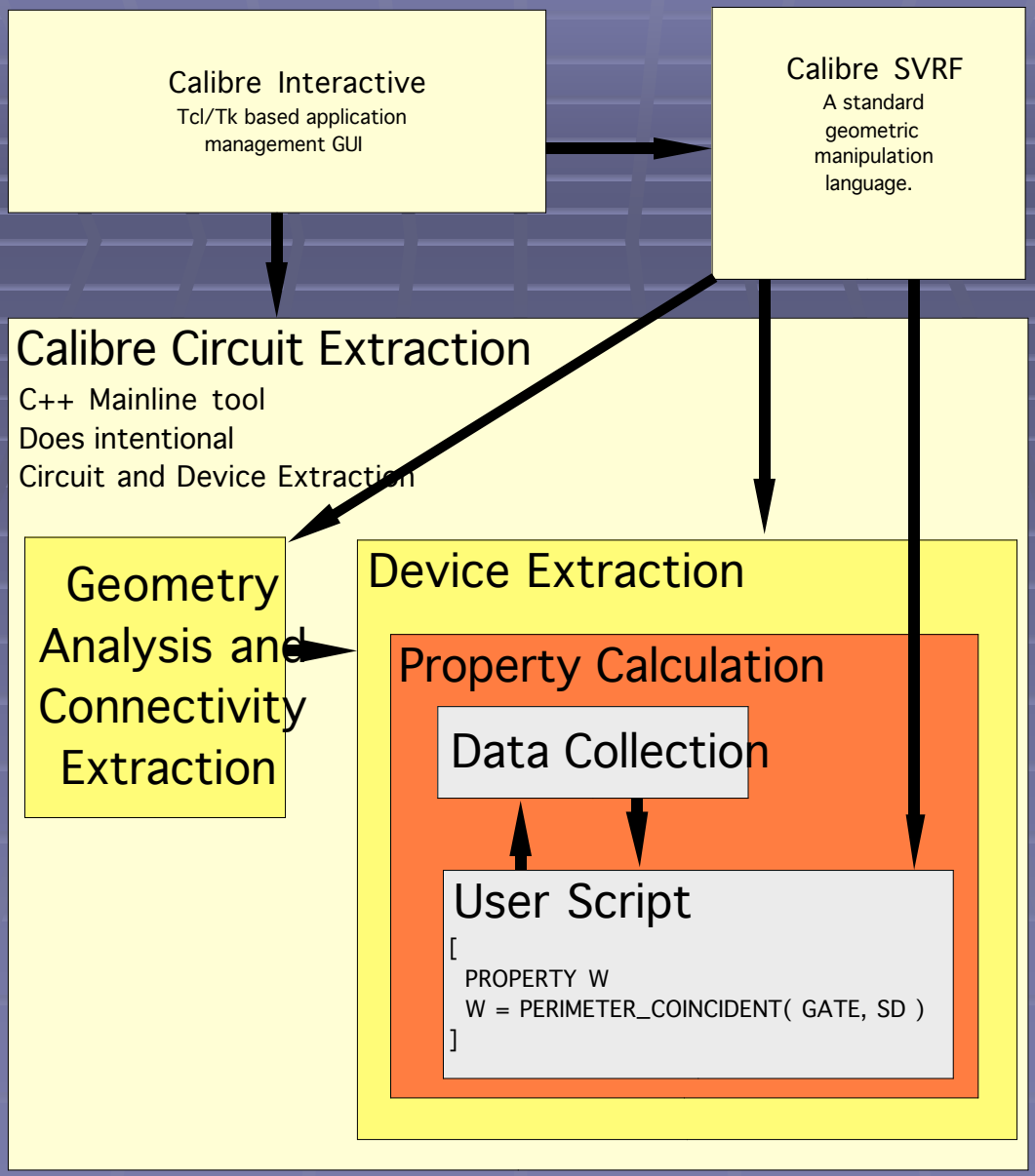
Layout vs. Schematic (LVS)

- Schematic – logical design of a circuit
- Layout – physical design of a circuit
- LVS makes sure Layout and Schematic designs describe the same circuit by comparing them

Device Recognition

- Device Recognition allows the user to turn geometric shapes from the layout into logical devices:




Calibre Interactive
Tcl/Tk based application management GUI
Calibre SVRF
A standard geometric manipulation language.
Calibre Circuit Extraction
C++ Mainline tool
Does intentional
Circuit and Device Extraction
Geometry Analysis and Connectivity Extraction
Device Extraction
Property Calculation
Data Collection
User Script
[
PROPERTY W
W = PERIMETER_COINCIDENT(GATE, SD)
]

Built-in Script Language

- Very Simple
- Very High Performance

Built-in Script Language

- byte compiled interpreted user program
- program configures data collection before execution starts
- Very limited capabilities
  - no loops, no allocation
  - just data collection, arithmetic, and conditionals

Built-in Script Language

- Data and temporary variables are pre-allocated
- Byte code runs for each device that is recognized

Built-in Script Language

```
DEVICE MP PGATE PGATE(G) SD(S) SD(D) DIFF
[
  property w, l
  w = 0.5 * ( perim_co( S, G )
            + perim_in( S, G )
        + perim_co( D, G )
        + perim_in( D, G ) )
  l = area( G ) / w
  if( ( bends( G ) != 0 ))
  {
    if( w > l )
      w = w – 0.5 * bends( G ) * l
    else l = l – 0.5 * bends( G ) * w
  }
]
```

Built-in Script Language

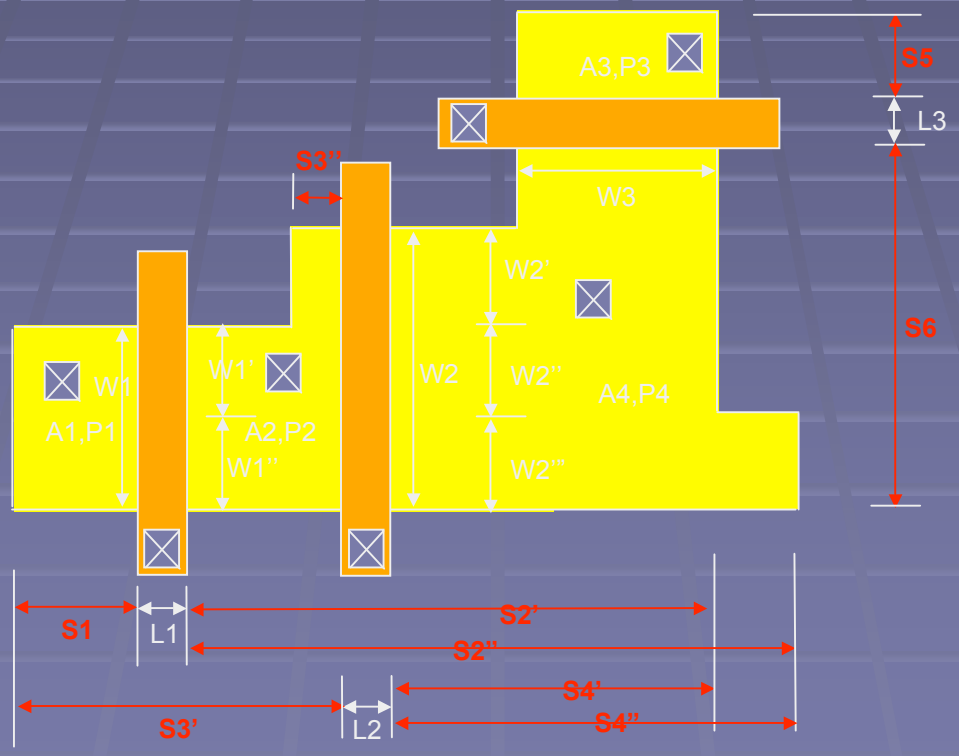
Variable Name	Value
w	0.0
l	0.0
temp1	0.0
temp2	0.0
temp3	0.0
perim_co(S,G)	4
perim_in(S,G)	6
perim_co(D,G)	4
perim_in(D,G)	6
area(G)	12
bends(G)	2

Byte Code

```
ADD   temp3  perim_co(S,G)  perim_in(S,G)
ADD   temp2  temp3  perim_co(D,G)
ADD   temp1  temp2  perim_in(D,G)
MUL   w  0.5  temp1
DIV   l  area(G)  w
NE    bends(G)  0 [31] [70]
GT    w  l [37] [55]
MUL   temp2  0.5  bends(G)
MUL   temp1  temp2  l
SUB   w  w  temp1
GOTO [70]
MUL   temp2  0.5  bends(G)
MUL   temp1  temp2  w
SUB   l  l  temp1
HALT
```

Reaching the Limits of Built-in

Enclosure Functions



Reaching the Limits of Built-in

- Complex measurements
- No simple number representation
- Summation capability required

Reaching the Limits of Built-in

```
DEVICE MP PGATE PGATE(G) SD(S) SD(D) DIFF
[
  property W, L, SEFFA
  W = 0.5 * ( perim_co(S,G) +
     perim_in(S,G) + perim_co(D,G)
     + perim_in(D,G) )
  L = area(G) / W
  if( ( bends(G) != 0 ))
  {
    if( W > L )
      W = W - 0.5 * bends(G) * L
    else L = L - 0.5 * bends(G) * W
  }
  S = ENC_PER (PGATE, DIFF, SD, 25)
  SEFFA = W / SUM(S::W / (S::A + 0.5*L) ) - 0.5*L
]
```

Reaching the Limits of Built-in

- Sum functionality satisfied specific requirements
- Soon a more general solution was needed
  - min/max
  - arbitrary calculations on array elements
  - a more complex measurement array
    - Multi Finger Enclosure measurements

Scripting Language Requirements

- Describe data collection prior to execution
  - Built-in language data retrieval function configures data collection algorithms
- Leverage existing Built-in script engine
  - Customers have extensive existing usage
  - Built-in engine has 31 data collection routines
- Get data collected to the script
- Get calculated results back
- Overall Performance can’t slow down too much

Why Tcl?

- Calibre already has Tcl interfaces inside our geometric manipulation language

How Tcl?

- Add a function interface to the Built-in language – TVF_NUMERIC_FUNCTION

  (TVF_NUM_FUN for short)

- Allows calling a Tcl Function from within the Built-in language

How Tcl?

```
DEVICE MP PGATE PGATE(G) SD(S) SD(D) DIFF
[
  property W, L, SEFFA
  W = 0.5 * ( perim_co(S,G) +
     perim_in(S,G) + perim_co(D,G)
     + perim_in(D,G) )
  L = area(G) / W
  if( ( bends(G) != 0 ))
  {
    if( W > L )
      W = W - 0.5 * bends(G) * L
    else L = L - 0.5 * bends(G) * W
  }
  S = ENC_PER (PGATE, DIFF, SD, 25)
  SEFFA = TVF_NUM_FUN (
        "calc_eff_a",
        "device_func ", S, W, L
  )
]
```

How Tcl?

```
TVF FUNCTION device_func [/*
  proc calc_eff_a { enc W L } {
     ... tcl code ...
  }
*/]
```

Requirements Check

- ✓ Describe data collection before script runs
  - Existing built-in script engine still provides the data collection functions
- ✓ Leverage existing script engine and customer scripts
  - Add Tcl calls to Existing built-in scripts
- ✓ Get data collected to Tcl
  - Function call arguments
- ✓ Get data calculations back to Calibre
  - Return values of Tcl functions called
- ? Overall Performance goal

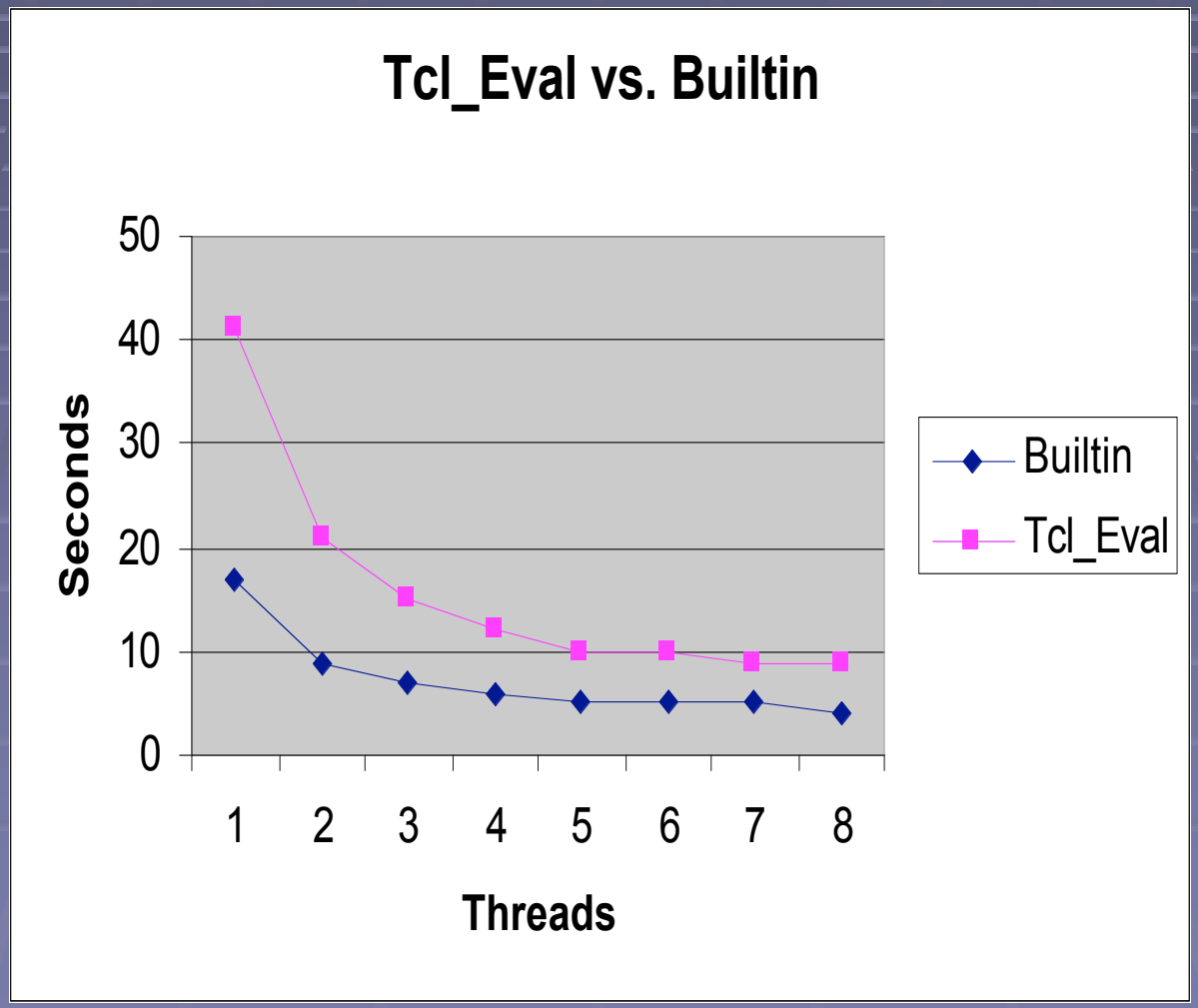
Performance Testing

- Test Circuits: Emphasize scripting language performance (not a real world test case)
  - 500,000 devices that run through the calculation script
    - 3 Builtin->Tcl calls for each device (1.5 million total)
    - 10 Tcl->Calibre object callbacks (5 million total)
  - Realistic Device Calculations
  - Partitions into 18 equal partitions for Multi-Threaded execution
  - Can be expressed as Tcl function or Built-In SUM

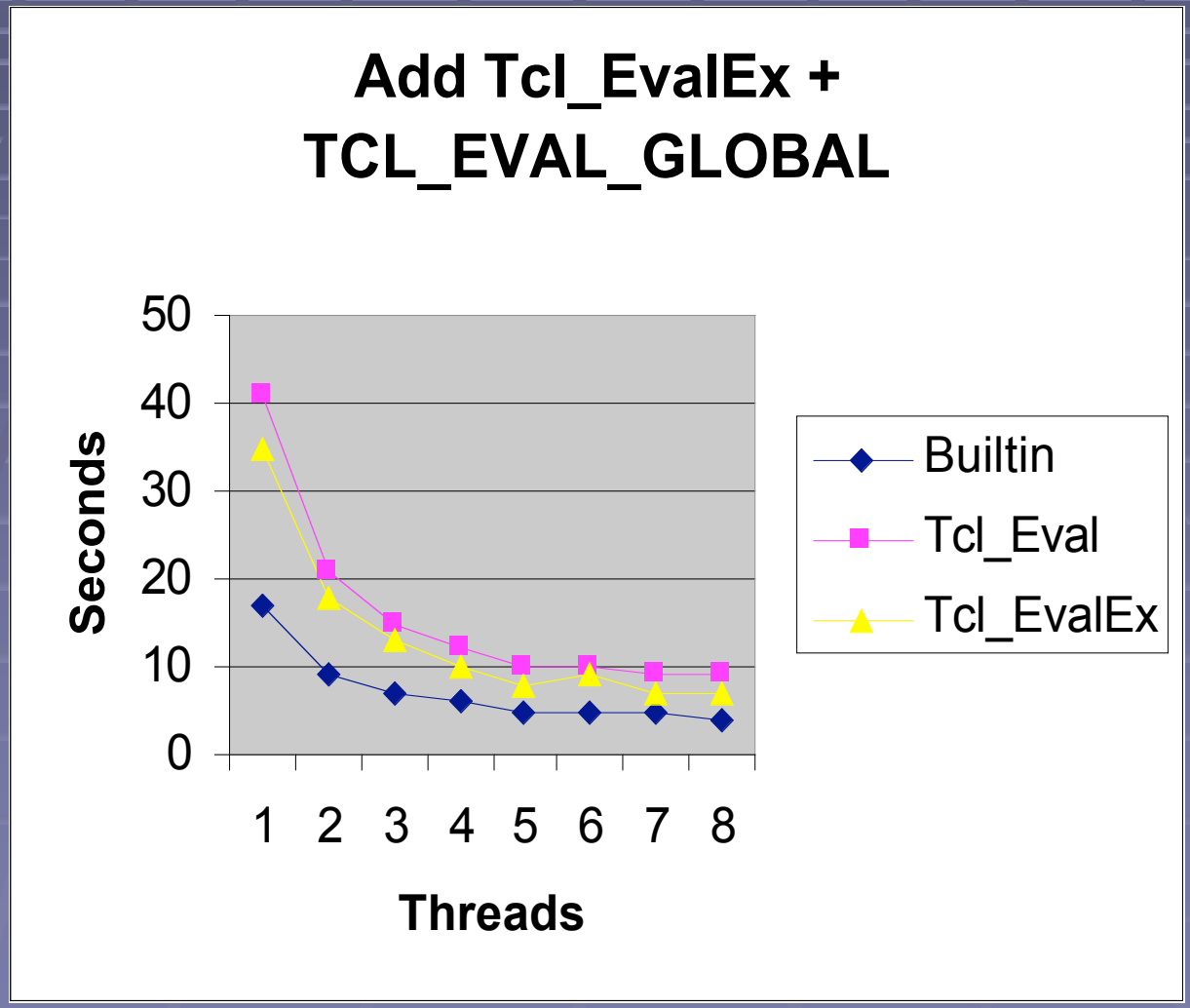
Efficiently Calling Tcl

- What call interface to choose:
  - Tcl_Eval
  - Tcl_EvalEx
  - Tcl_EvalObjEx
  - Tcl_EvalObjv

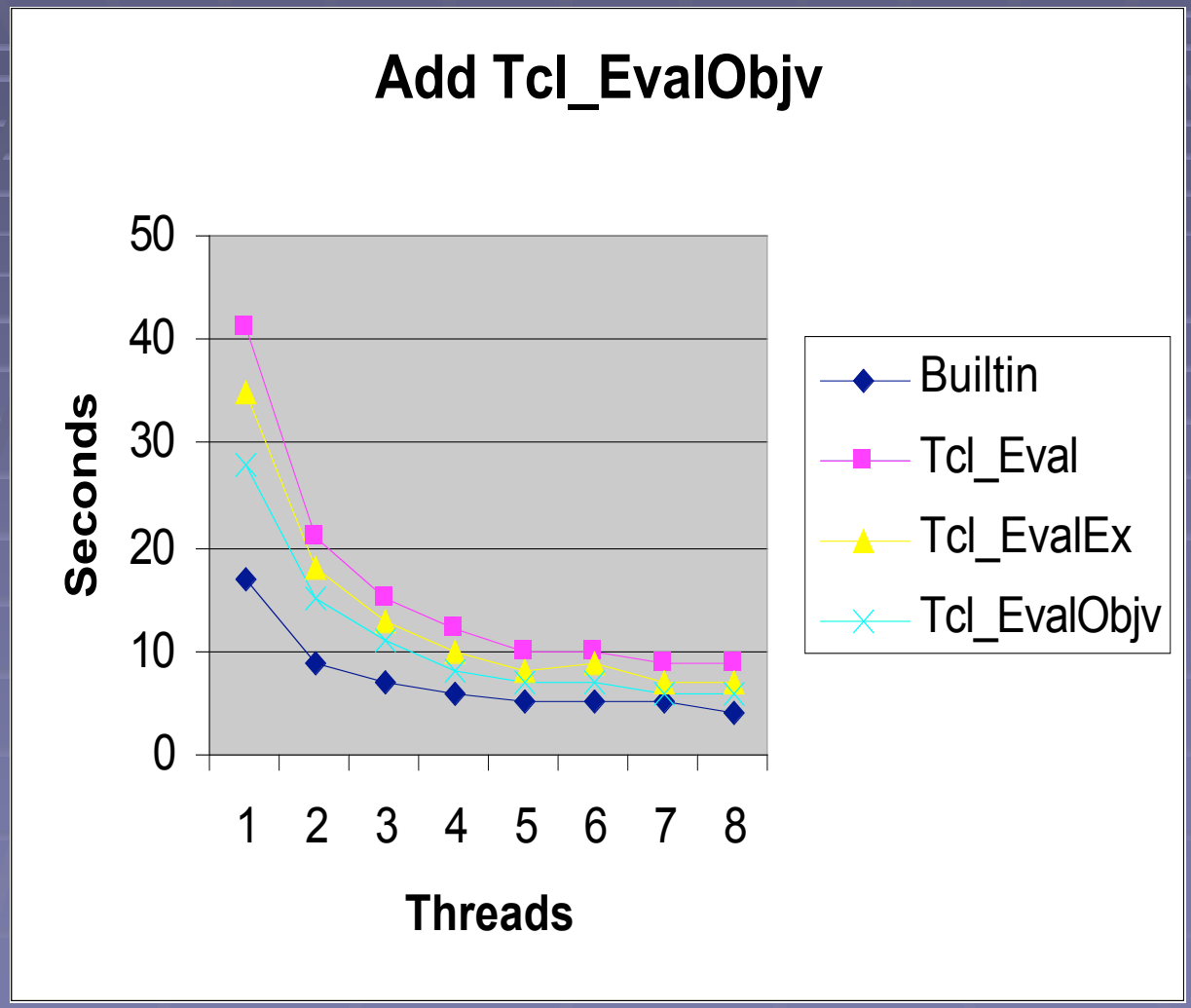
Efficiently Calling Tcl



Efficiently Calling Tcl



Efficiently Calling Tcl



Efficiently Calling Tcl

- Use of Tcl_EvalObjEx didn't differ significantly from Tcl_EvalObjv
- Argument Handling
  - Set up as much as possible before execution
  - Data passed through objects (Tcl_CreateObjCommand) to avoid construction of argument objects

Getting Back to C++

- Data is passed back through a Tcl_DoubleObj set as the “result”
- Tcl handles the result objects efficiently

Writing Efficient Tcl

- Beware the use of expr without {}
- changing expr { ... } to expr ...
  - reduced performance by 300%

Writing Efficient Tcl

- Consider object interfaces that do the bookkeeping operations:

```
set slice_count [ $enc slice_count ]
for { set i 0 } { $i<$slice_count } { incr i } {
    # slice specific code here uses index $i to access data
    # from the $enc command object by passing it in as an argument
}
```

Writing Efficient Tcl

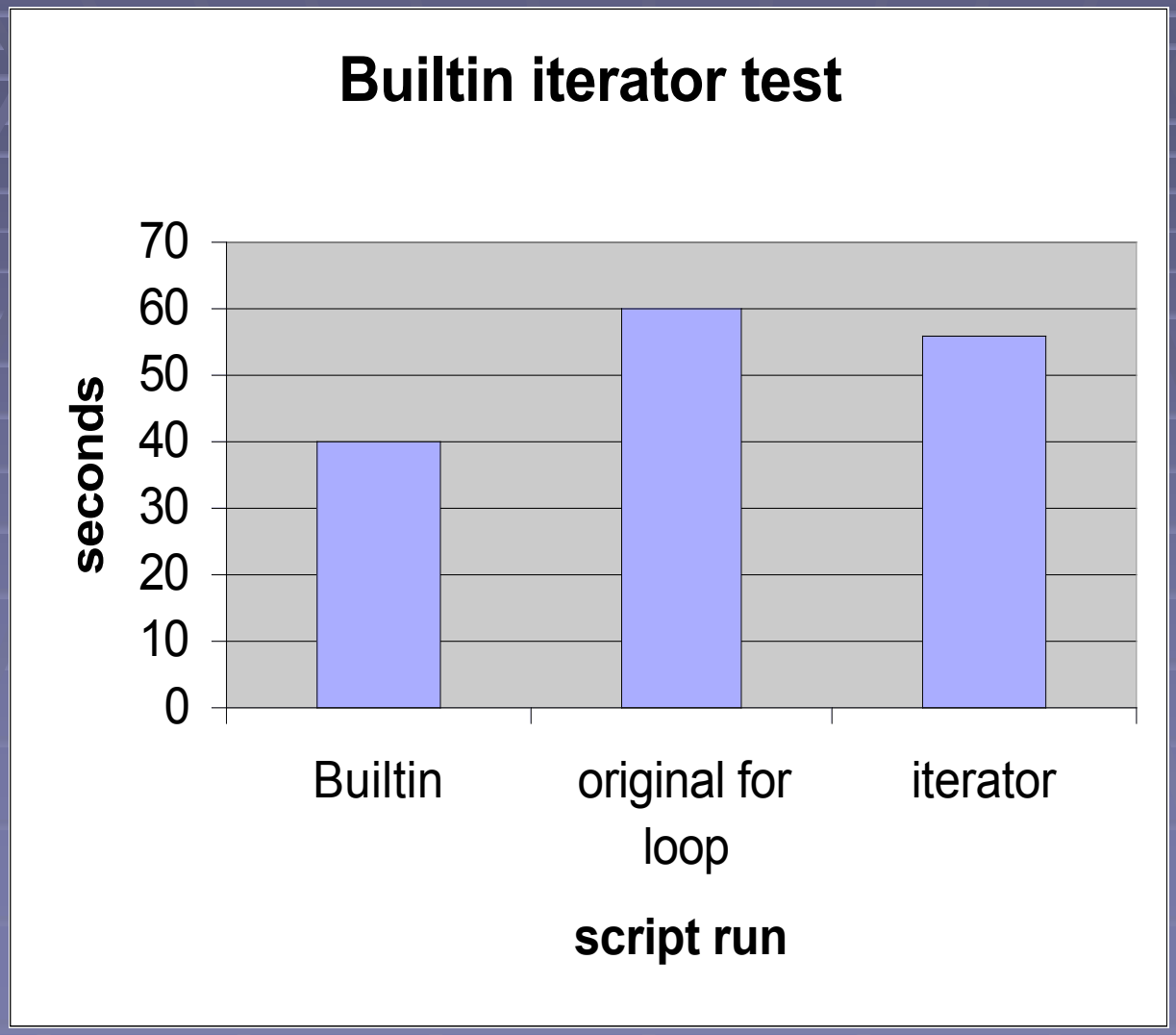
- Consider this interface where the $enc object performs index calculations

```
set i [ $enc first_slice ]
while { $i > 0 }  {
  # slice specific code here index is implicit $enc will
  # access data from the current array slice.
  set i [ $enc next_slice ]
}
```

- Or, if you want to get fancy:

```
$rtval = $enc eval {
    # some sort of code that executes once for each slice
}
```

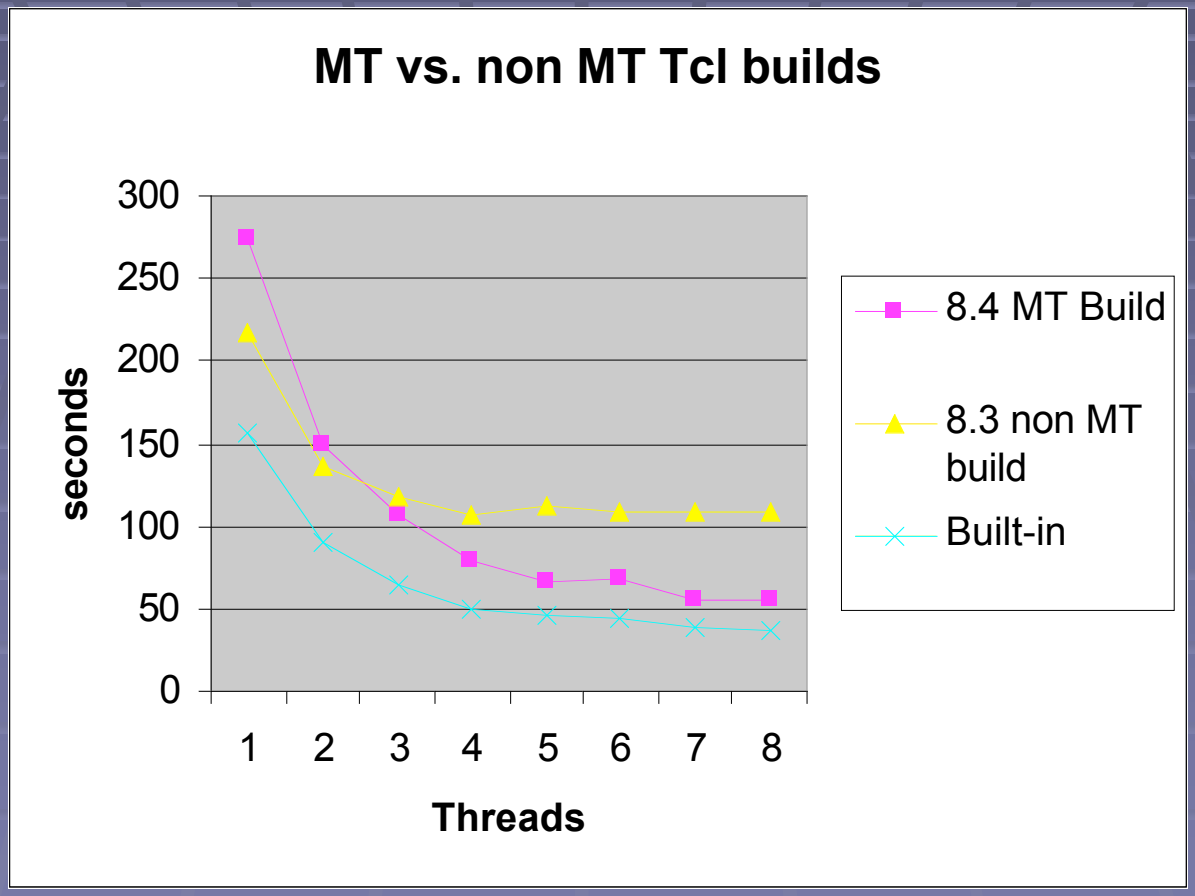
Writing Efficient Tcl



Threads

- Data is partitioned for parallel execution
  - Each thread recognizes devices independently
  - Each thread runs its own copy of the built-in language engine
- How do threads perform with Tcl?
- Better to use MT Tcl or Single Threaded Tcl?

Threads


MT vs. non MT Tcl builds
seconds
300
250
200
150
100
50
0
1
2
3
4
5
6
7
8
Threads
8.4 MT Build
8.3 non MT build
Built-in

Final Performance Numbers

Real chips showed no increase in required CPU time for Device Recognition