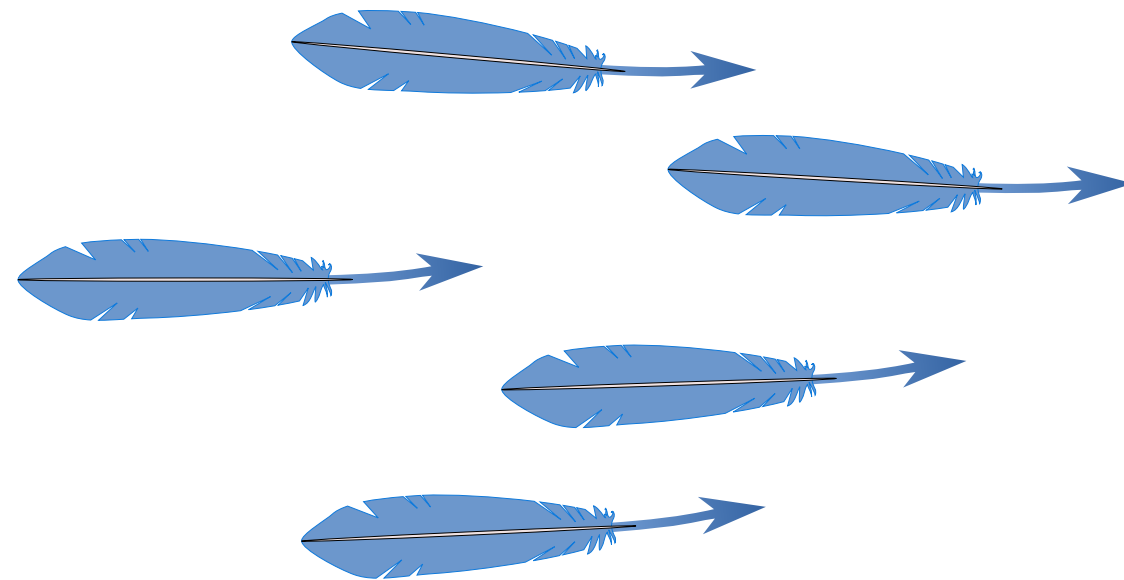


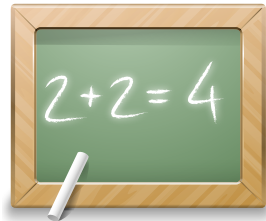
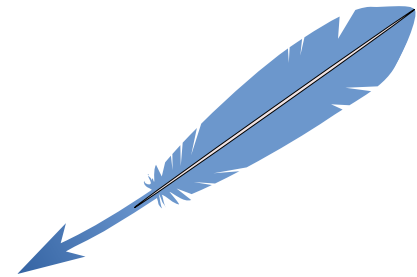
EIAS and numerical math — introducing VecTcl



Christian Gollwitzer

EuroTcl 2014

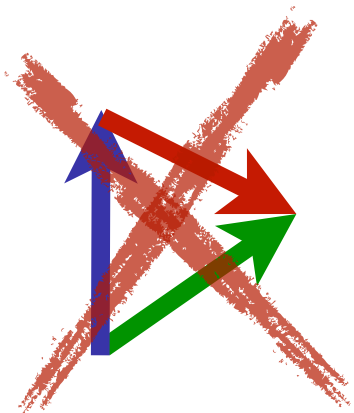
What is numerical / vector math?



Tcl has (scalar) math in the core:

$$x = \frac{1}{2a} \left(b \pm \sqrt{b^2 - 4ac} \right)$$

```
set x [expr {($b+sqrt($b**2-4*$a*$c))/(2*$a)}]
```



There is no direct support for vector math:

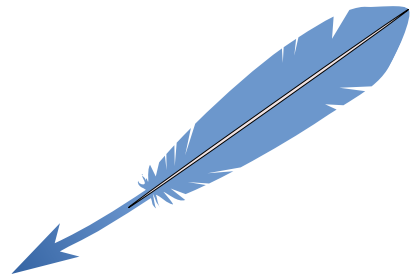
$$x = \vec{a} \cdot \vec{b} = \sum_i a_i b_i$$

```
set x 0.0
foreach ai $a bi $b {
    set x [expr {$x+$a*$b}]
}
```



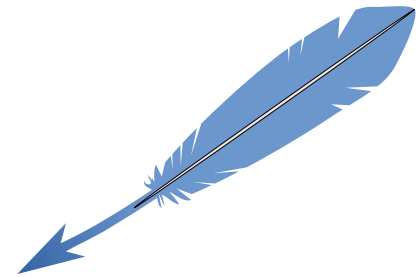
Tcl is more complicated / uglier than textbook math

VecTcl demo / Why do we need it?



- Python has it ;)
- Cool applications: Data fitting, image processing, Physical modelling, 3D graphics
- Tcl is often too slow

Linear regression



Math

$$\bar{x} = \frac{1}{N} \sum x_i$$

$$\bar{y} = \frac{1}{N} \sum y_i$$

$$\beta = \frac{\sum_i (x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$

$$\alpha = \bar{y} - \beta \bar{x}$$

VecTcl

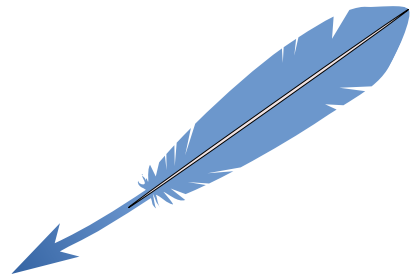
```
vexpr {  
  xm=mean(xv)  
  ym=mean(yv)  
  beta=sum((xv-xm) .* (yv-ym)) ./ sum((xv-xm) .^2)  
  alpha=ym-beta*xm  
}
```

Tcl

```
set xsum 0.0; set ysum 0.0  
foreach x $xv y $yv {  
  set xsum [expr {$xsum + $x}]  
  set ysum [expr {$ysum + $y}]  
}  
set xm [expr {$xsum/[llength $xv]}]  
set ym [expr {$ysum/[llength $xv]}]  
set xsum 0.0; set ysum 0.0  
foreach x $xv y $yv {  
  set dx [expr {$x - $xm}]  
  set dy [expr {$y - $ym}]  
  set xsum [expr {$xsum + $dx * $dy}]  
  set ysum [expr {$ysum + $dx * $dx}]  
}  
set b [expr {$xsum / $ysum}]  
set a [expr {$ym - $b * $xm}]
```



Linear Least Squares



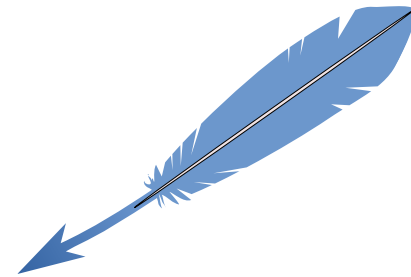
Even easier: the matrix way

$$A = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix} \Rightarrow \text{solve } Ap = y \Rightarrow p = (\alpha \quad \beta)$$

VecTcl

```
vexpr {  
    A=hstack(1, xv)  
    alpha, beta = A \ yv  
}
```

Linear Least Squares

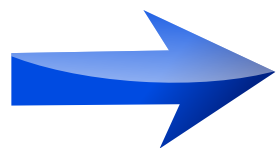


Even easier: the matrix way

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{pmatrix} \Rightarrow \text{solve } Ap = y \Rightarrow p = (\alpha \quad \beta \quad \gamma)$$

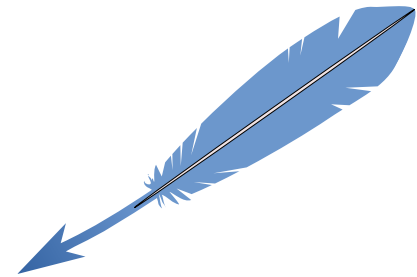
VecTcl

```
vexpr {  
    A=hstack(1, xv, xv.^2)  
    alpha, beta, gamma = A \ yv  
}
```



no way to do it in pure Tcl
(short of writing LLS matrix decomposition)

Why a new package?



Q: There is TIP 363 , TIP 420, and a couple of extensions.
Who needs a new vector package?

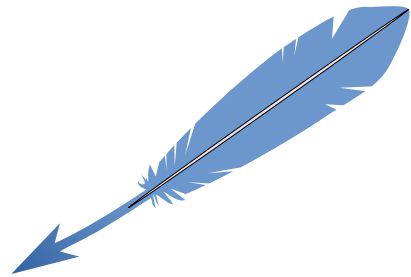
A: VecTcl was designed to be

- Easy to use
- Interoperable
- General
- Easy to build and run
- Fast and memory efficient



VecTcl beats all competitors in all of these points

A word about the TIPs...



TIP #363: Vector Math in the Tcl Core

```
set ListA {1 2 3}
set b "$ListA";# -> "{1 2 3}"
set b "{*}$ListA";# -> "1 2 3"
```

„The proposed approach will enable *eventual* incorporation of vector-math to the Tcl engine“



```
expr $ListA + {4 5 6};# -> "{5 7 9}"
```

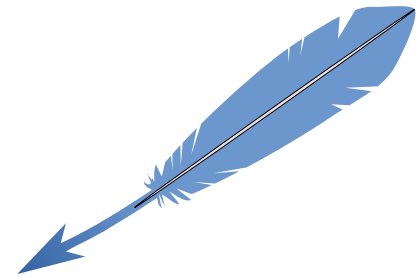
Q: Why on earth break everything to enable „eventual“ incorporation of vector-math to the Tcl engine?

```
set name "Mr. John Ousterhout"
puts "Hello, $name"
# Hello, {Mr. John Ousterhout}
```



breaks „Hello world!“

A word about the TIPs...



TIP #363: Vector Math in the Tcl Core

```
set ListA {1 2 3}
set b "$ListA";# -> "{1 2 3}"
set b "{*}$ListA";# -> "1 2 3"
```

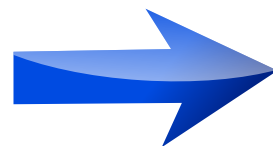
„The proposed approach will enable *eventual* incorporation of vector-math to the Tcl engine“



```
expr $ListA + {4 5 6};# -> "{5 7 9}"
```

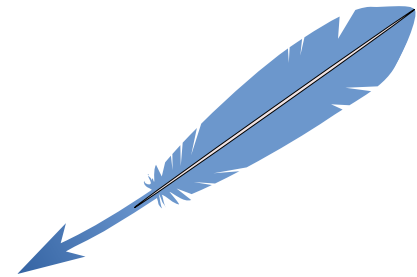
Q: Why on earth break everything to enable „eventual“ incorporation of vector-math to the Tcl engine?

```
set ListA {1 2 3}
vexpr {ListA + {4 5 6}}
```



works and doesn't break anything

A word about the TIPs...



TIP #420: Vector Math in the Tcl Core

```
set a {1 2 3}
set b {4 5 6}
vexpr $a $b + 2 *
# {10.0 14.0 18.0}
```



It actually works — there is an implementation for 3D

*„vexpr is a vector expression parser. It operates using **reverse-polish notation** [...] Why? Well **mostly for ease of implementation**. Partly because there is no PEMDAS equivalent order of operation for matrices and vectors.“*

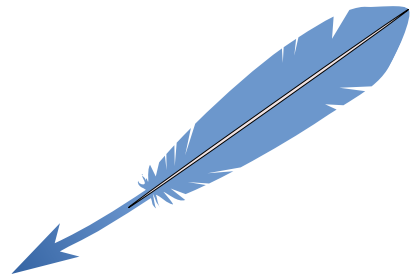
Q: Only 3D? No operator precedence for ease of implementation? Use a real parser generator!

```
set a {1 2 3}
set b {4 5 6}
vexpr {2*(a+b)}
```



Textbook-compatible
operator precedence in N-D

Ease of use



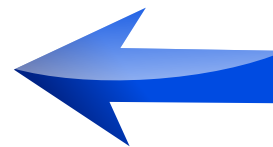
VecTcl

```
vexpr {  
  xm=mean( xv )  
  ym=mean( yv )  
  beta=sum( ( xv-xm ) . * ( yv-ym ) ) . / sum( ( xv-xm ) . ^ 2 )  
  alpha=ym-beta*xm  
}
```

Input/Output: just use
xv, yv, alpha, beta

BLT / rbc

```
vector create x  
vector create y  
vector create alpha  
vector create beta  
x set $xv  
y set $yv  
beta expr { (mean(x*y)-mean(x)*mean(y)) / (mean(x^2)-mean(x)^2) }  
alpha expr { mean(y)-beta*mean(x) }  
set alpha [alpha index 0]  
set beta [beta index 0]
```

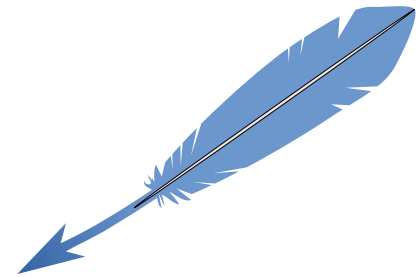


Input from Tcl

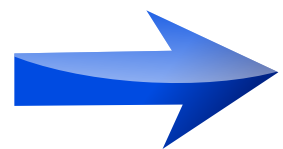


Output to Tcl

Interoperability

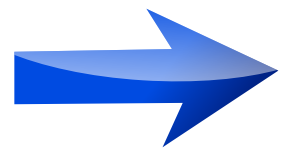


- Tcl has a type for sequences of numbers: lists
- VecTcl operates on lists (and lists-of-lists)
- VecTcl uses Tcl variables and commands (functions)



Simply use your variable in pure Tcl, VecTcl, math::linearalgebra, odielib, lapack...

```
package require math::linearalgebra
namespace import math::linearalgebra::*
set X [mkDingdong 3]
vexpr { X \ { 1 2 3 } }
```



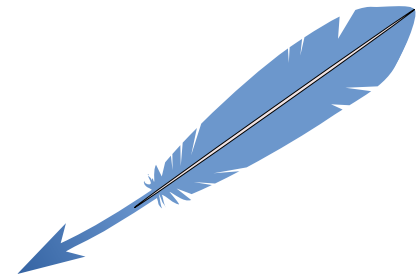
Simply use other commands in VecTcl

```
vexpr { orthonormalizeColumns(X) }
```



Different from NAP, BLT, tcl-tna, narray, tensor... but like the TIPs

Generality



VecTcl supports:

- integer, floating point and complex numbers
- scalars, vectors, matrices and N-rank tensors
- string rep carefully designed to comply with lists

```
set x {1 2 3} ;# an integer vector of length 3
```

```
set y {{1.0 3.0} {3.0 5.0}} ;# a 2x2 floating-point matrix
```

```
set y {{1.0 3.0 5.0}} ;# a 1x3 floating-point matrix (row vector)
```

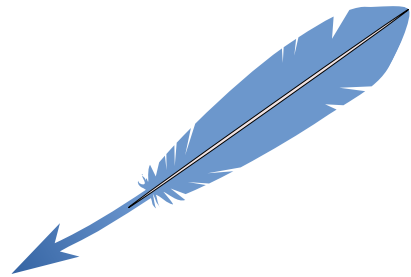
```
set z {0+1i 2+3.5i 3.0+0i} ;# a complex vector of length 3
```

```
set u {{1 2 3}} ;# a 1x3 integer matrix (a row vector)
```

```
set v {{{1 2} {3 4}} {{5 6} {7 8}}} ;# a 2x2x2 integer tensor
```

```
set e {1.0 2 3} ;# a floating point vector of length 3
```

No external dependencies



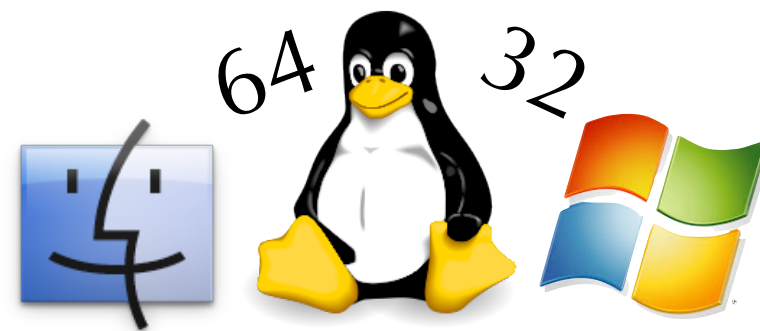
To compile VecTcl you need:

- a C compiler
- Tcl

~~FORTAN
C++ GPL©~~

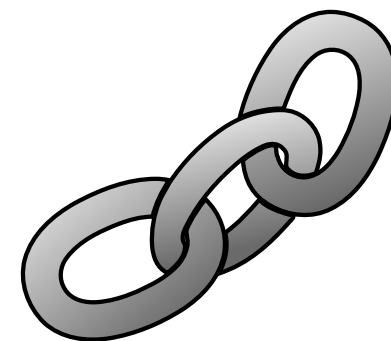
To run VecTcl you need:

- VecTcl
- TclOO

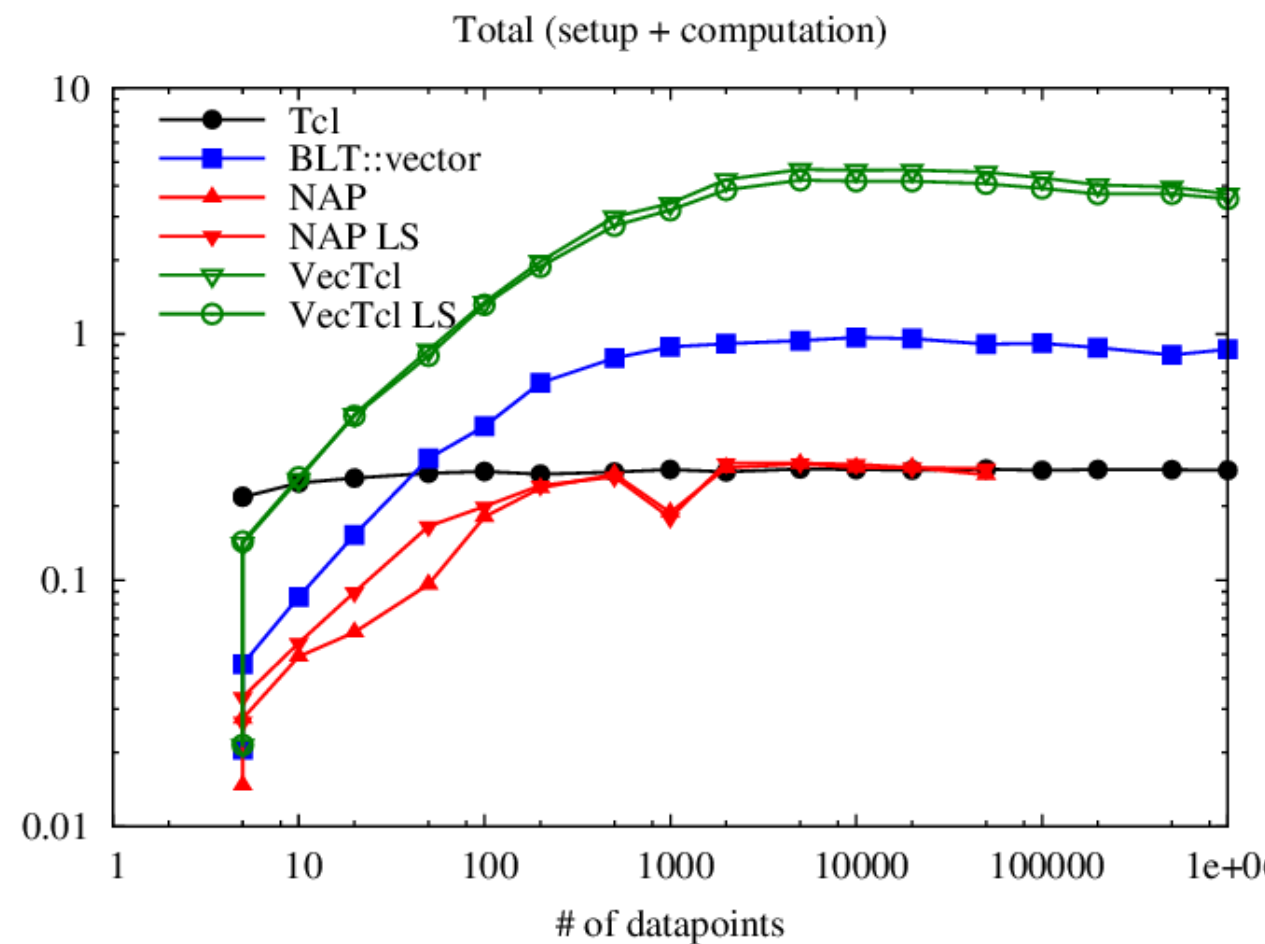
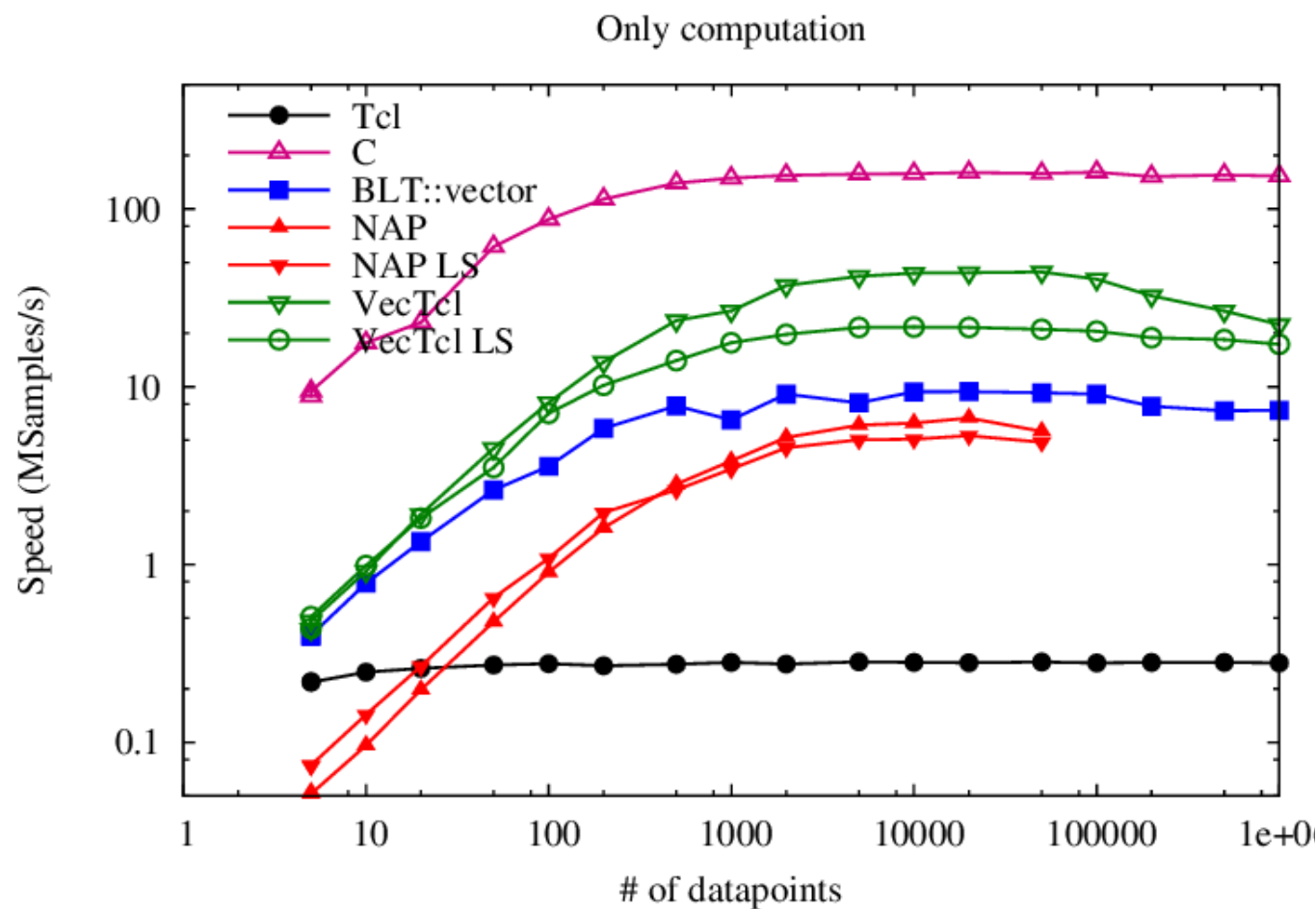
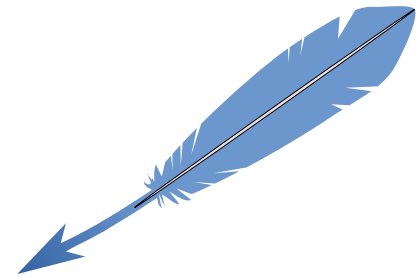


To rebuild VecTcl from scratch:

- autoconf
- tcllib::parsertools
- CLAPACK

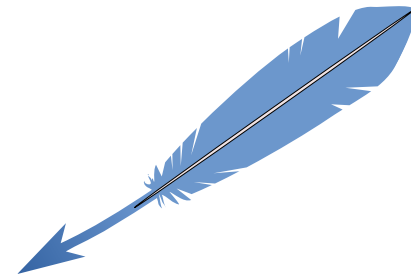


Benchmarks - linear regression



- VecTcl is 4x slower than C, but still faster than the others
- Shimmering is 5x slower than actual computation
- Competitors are still slower there

How does it work?



VecTcl is a 2-layered system

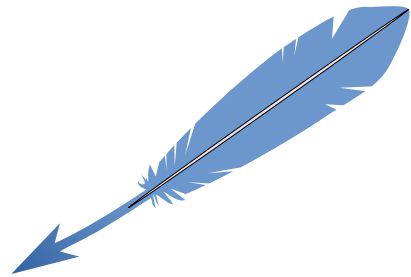
```
vexpr {  
    a={1 2 3}  
    c=2*(a+{4 5 6})  
}
```

Compiler, written in Tcl

Runtime, written in C

```
proc numarray::compiledexpressionXX {} {  
    upvar 1 a a  
    upvar 1 c c  
    set a {1 2 3}  
    set c [numarray::* 2 [numarray::+ [set a] {4 5 6}]]  
}  
numarray::compiledexpressionXX
```

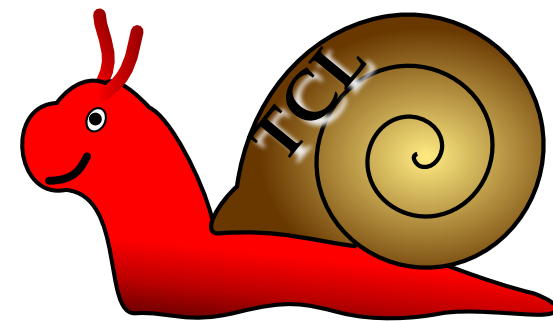

Why or when is it slow?



Tightly coded loops:

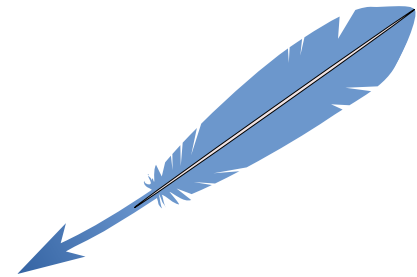
```
vexpr {  
  a=zeros(1000);  
  for i=0:999 {a[i]=2*i}  
}
```

```
set a [zeros 1000]  
set __temp1 999  
for {set i 0} {$i <= $__temp1} {incr i 1} {  
numarray::= a [list [list [set i] [set i] 1]]  
[numarray::* 2 [set i]]  
}
```

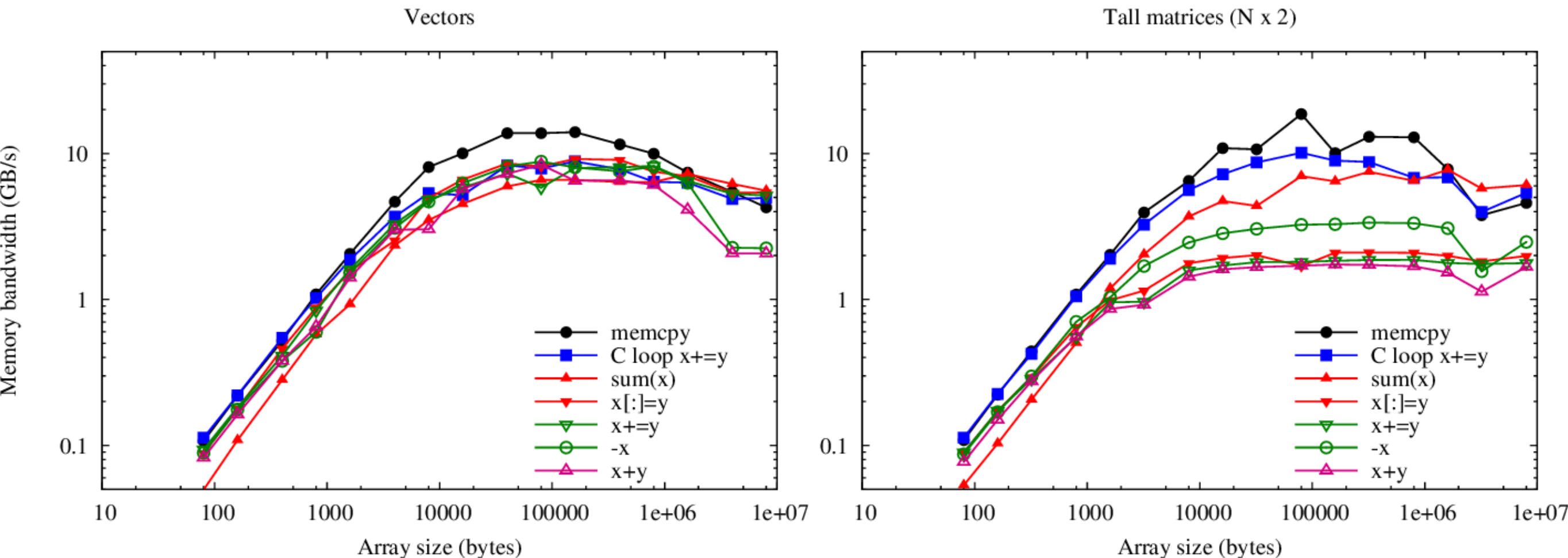


- Avoid if possible: `vexpr { a=2*linspace(0,999,1) }`
- Huge speed-up possible by JIT compilation (tcc4tcl?)

Why or when is it slow?

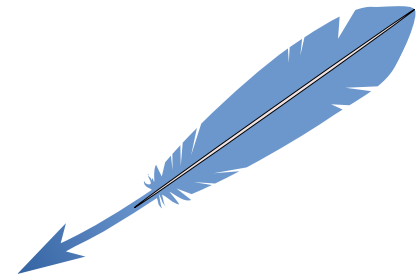


Speed of the elementary operations



- Vector operations close to the memory bandwidth
- Until ~10kbytes, the command dispatch dominates
- Matrix shape (currently) has a strong effect
- Improvement by OpenMP, BLAS, better iterators

Why or when is it slow?



Speed of complex operations

```
vexpr {  
    r=a.*a + b.*b  
}
```



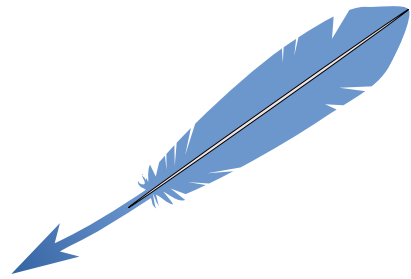
```
for (int i=0; i<N; i++) {  
    r[i] = a[i]*a[i] + b[i]*b[i];  
}
```

- $2N$ Flops
- $2N$ temporary storage
- 3 passes over the data

- $2N$ Flops
- 2 temporary registers
- 1 pass over the data

- 🌐 Not easy to solve (short of JIT)
- 🌐 More complex operations
- 🌐 Blocking

Conclusion & The Future



- VecTcl provides an easy interface to numeric math in Tcl
- Value semantics (Tcl_Obj) in contrast to other packages
- Performance superior to other packages, but worse than C
- Performance could suffer by changes in Tcl 9
- JIT compilation possible using tcc4tcl

