

When Perl is not Quite Fast Enough

**Nicholas Clark
London Perl Mongers**

Why optimise?

- *Premature optimisation is the root of all evil*
- Don't optimise until you absolutely have to
- Optimise when your existing implementation
 - ◆ doesn't **quite** work well enough on the hardware you have
 - ◆ doesn't **quite** work well enough on hardware as cheap as you'd like
 - ◆ doesn't **quite** scale as well as you'd like

Why optimise?

- *Premature optimisation is the root of all evil*
- Don't optimise until you absolutely have to
- Optimise when your existing implementation
 - ◆ doesn't **quite** work well enough on the hardware you have
 - ◆ doesn't **quite** work well enough on hardware as cheap as you'd like
 - ◆ doesn't **quite** scale as well as you'd like
- First make it work, **then** make it work fast

Why don't optimise?

- trading (?:clarity|style) for speed
 - ◆ makes maintenance harder
 - ◆ makes extension harder
 - ◆ tunes implementation - changes will de-tune

Why don't optimise?

- trading (?:clarity|style) for speed
 - ◆ makes maintenance harder
 - ◆ makes extension harder
 - ◆ tunes implementation - changes will de-tune
- effort could be spent better elsewhere?

Remember the 80/20 rule

- 80/20 rule
 - 90/10
 - 70/30
 - 60/40
- Pareto's Principle
- Law of diminishing returns

Cheating

- Find better algorithm

Cheating

- Find better algorithm
- Throw more hardware at it
 - ❑ hardware cheap, programmers well paid
 - ❑ saves time
 - ❑ predictable result
 - ❑ keeps code clean

Cheating

- Find better algorithm
- Throw more hardware at it
 - ❑ hardware cheap, programmers well paid
 - ❑ saves time
 - ❑ predictable result
 - ❑ keeps code clean
- mod_perl
 - ❑ FastCGI
 - ❑ PPerl

Cheating

- Find better algorithm
- Throw more hardware at it
 - ❑ hardware cheap, programmers well paid
 - ❑ saves time
 - ❑ predictable result
 - ❑ keeps code clean
- mod_perl
 - ❑ FastCGI
 - ❑ PPerl
- Rewrite in **C**, er **C++**, sorry **Java**, I mean **C#**, oops no ...

Compromises

- XS

Compromises

- XS
- Inline
 - For scripts and for modules
 - Needs temporary directory

Compromises

- XS
- Inline
 - For scripts and for modules
 - Needs temporary directory
- Compile your own perl?

Build your own perl

- get source from CPAN
 - ◆ build in source directory

Build your own perl

- get source from CPAN
 - ◆ build in source directory
- ./Configure

Build your own perl

- get source from CPAN
 - ◆ build in source directory
- ./Configure
- make
 - ◆ builds "miniperl"
 - ◆ uses miniperl to continue
 - ◆ builds the extensions
 - ◆ builds "perl"

Build your own perl

- get source from CPAN
 - ◆ build in source directory
- ./Configure
- make
 - ◆ builds "miniperl"
 - ◆ uses miniperl to continue
 - ◆ builds the extensions
 - ◆ builds "perl"
- make test

Build your own perl

- get source from CPAN
 - ◆ build in source directory
- ./Configure
- make
 - ◆ builds "miniperl"
 - ◆ uses miniperl to continue
 - ◆ builds the extensions
 - ◆ builds "perl"
- make test
- make install

Build your own perl

- install to ~

```
./Configure -Dprefix=~
```

- use alternative compiler

```
./Configure -Dcc=/usr/local/bin/gcc-3.2
```

- use more 'optimizer';

```
-Doptimize='-O3 -march=i586 -fomit-frame-pointer'
```

- avoid slow things

- ◆ -g
- ◆ threads
- ◆ -DDEBUGGING

Compromises

- XS
- Inline
 - For scripts and for modules
 - Needs temporary directory
- Compile your own perl?
- Different perl version

Compromises

- XS
- Inline
 - For scripts and for modules
 - Needs temporary directory
- Compile your own perl?
- Different perl version
 - ◆ problems
 - gcc 3.2

```
perl -ni~ -we 'print unless /: </' x2p/makefile makefile
```

- FreeBSD

```
-Uusenm
```

```
-Dlddflags='-shared -L/usr/local/lib'
```

```
-Dldflags='-Wl,-E -L/usr/local/lib'
```

Tests

- Must not introduce new bugs
- Use your full regression tests :-)
- **Keep** a copy of original program
 - ◆ use source control

Where to start?

- What might be slow?
- Think of re-write
- Try it
- Note results

Profilers

- Unix time command

```
$ time perl5.8.0 enc2xs.orig -Q -O -o /dev/null ...  
...
```

```
real    1m16.643s  
user    1m7.214s  
sys     0m4.421s
```

Profilers

- Unix time command

```
$ time perl5.8.0 enc2xs.orig -Q -O -o /dev/null ...  
...
```

```
real    1m16.643s  
user    1m7.214s  
sys     0m4.421s
```

- Devel::DProf

```
$ perl5.8.0 -d:DProf enc2xs.orig -Q -O -o /dev/null ...
```

Profilers

- Unix time command

```
$ time perl5.8.0 enc2xs.orig -Q -O -o /dev/null ...  
...
```

```
real    1m16.643s  
user    1m7.214s  
sys     0m4.421s
```

- Devel::DProf

```
$ perl5.8.0 -d:DProf enc2xs.orig -Q -O -o /dev/null ...
```

- Devel::SmallProf

- ◆ :-) Should give line by line profiling
- ◆ :-(Fails tests on 5.8

- And others

Devel::DProf

- Run with `-d:DProf`

```
perl5.8.0 -d:DProf enc2xs.orig -Q -O -o /dev/null ...
```

- Run `dprofpp` to process the `tmon.out` file.

```
Total Elapsed Time = 66.85123 Seconds
```

```
User+System Time = 62.35543 Seconds
```

```
Exclusive Times
```

%Time	ExclSec	Cumuls	#Calls	sec/call	Csec/c	Name
106.	66.70	102.59	218881	0.0003	0.0005	main::enter
49.5	30.86	91.767	6	5.1443	15.294	main::compile_ucm
19.2	12.01	8.333	45242	0.0003	0.0002	main::encode_U
4.74	2.953	1.078	45242	0.0001	0.0000	utf8::unicode_to_na
4.16	2.595	0.718	45242	0.0001	0.0000	utf8::encode
0.09	0.055	0.054	5	0.0109	0.0108	main::BEGIN
0.01	0.008	0.008	1	0.0078	0.0078	Getopt::Std::getopt
0.00	0.000	-0.000	1	0.0000	-	Exporter::import
0.00	0.000	-0.000	3	0.0000	-	strict::bits
0.00	0.000	-0.000	1	0.0000	-	strict::import
0.00	0.000	-0.000	2	0.0000	-	strict::unimport

What causes slowness

- CPU

What causes slowness

- CPU
- RAM
 - ◆ perl trades RAM for speed
 - ◆ getting slower (relative to CPU)
 - ◆ *Random Access Memory* - pah!
 - ◆ memory like a pyramid

What causes slowness

- CPU

- RAM

```
if (exists $seen{$uch}) {  
    warn sprintf("U%04X is %02X%02X and %02X%02X\n",  
                $val, $page, $ch, @{$seen{$uch}});  
}  
else {  
    $seen{$uch} = [$page, $ch];  
}
```

What causes slowness

- CPU

- RAM

```
if (exists $seen{$uch}) {  
    warn sprintf("U%04X is %02X%02X and %02X%02X\n",  
                $val, $page, $ch, @{$seen{$uch}});  
}  
else {  
    $seen{$uch} = [$page, $ch];  
}
```

- ◆ faster as

```
$seen{$uch} = $page << 8 | $ch;
```

Benchmark

- Try out your ideas

```
use Benchmark ':all';
```

```
sub orig {  
    ...  
}
```

```
sub new {  
    ...  
}
```

```
cmpthese (10000, { orig => \&orig, new => \&new } );
```

Benchmark

- Try out your ideas

```
use Benchmark ':all';
```

```
sub orig {  
    ...  
}
```

```
sub new {  
    ...  
}
```

```
cmpthese (10000, { orig => \&orig, new => \&new } );
```

- And see what it says

```
Benchmark: timing 10000 iterations of new, orig...
```

```
    new:  1 wallclock secs ( 0.70 usr +  0.00 sys =  0.70 CPU)
```

```
    orig:  4 wallclock secs ( 3.94 usr +  0.00 sys =  3.94 CPU)
```

```
      Rate orig  new
```

```
orig  2540/s  -- -82%
```

```
new  14222/s 460%  --
```

What causes slowness in perl?

- Ops
- CPU
- RAM

What causes slowness in perl?

- Ops

- ◆ Start with line of hex (54726164696e67207374796c652f6d61)
- ◆ split into into groups of 4 digits (5472 6164 696e ...)
- ◆ convert each to a number.

```
sub orig {  
    map {hex $_} $line =~ /(\....)/gi  
}  
sub new {  
    unpack "n*", pack "H*", $line;  
}
```

Ops are bad, m'kay

```
$ perl -MO=Terse -e'map {hex $_} $line =~ /((...)/g;'
LISTOP (0x16d9c8) leave [1]
  OP (0x16d9f0) enter
  COP (0x16d988) nextstate
  LOGOP (0x16d940) mapwhile [2]
    LISTOP (0x16d8f8) mapstart
      OP (0x16d920) pushmark
      UNOP (0x16d968) null
      UNOP (0x16d7e0) null
      LISTOP (0x115370) scope
        OP (0x16bb40) null [174]
        UNOP (0x16d6e0) hex [1]
        UNOP (0x16d6c0) null [15]
        SVOP (0x10e6b8) gvsv GV (0xf4224) *_
  PMOP (0x114b28) match /((...)/
    UNOP (0x16d7b0) null [15]
    SVOP (0x16d700) gvsv GV (0x111f10) *line
```

Ops are bad, m'kay

```
$ perl -MO=Terse -e'unpack "n*", pack "H*", $line;'
LISTOP (0x16d818) leave [1]
  OP (0x16d840) enter
  COP (0x16bb40) nextstate
  LISTOP (0x16d7d0) unpack
    OP (0x16d7f8) null [3]
    SVOP (0x10e6b8) const PV (0x111f94) "n*"
  LISTOP (0x115370) pack [1]
    OP (0x16d7b0) pushmark
    SVOP (0x16d6c0) const PV (0x111f10) "H*"
    UNOP (0x16d790) null [15]
      SVOP (0x16d6e0) gvsv GV (0x111f34) *line
```

Benchmarking size

- Devel::Size

```
$ perl -lw  
use Devel::Size;  
$page = 1; $ch = 1;  
print Devel::Size::total_size ([$page, $ch]);  
__END__  
92
```

```
$ perl -lw  
use Devel::Size;  
$page = 1; $ch = 1;  
print Devel::Size::total_size ($page << 8 | $ch);  
__END__  
16
```

Banish the demons of Stupidity

- Arrays and Hashes

Banish the demons of Stupidity

- Arrays and Hashes
- Regexps

```
if ( /\G.../gc ) {  
} elsif ( /\G.../gc ) {  
} elsif ( /\G.../gc ) {
```

Banish the demons of Stupidity

- Arrays and Hashes

- Regexps

```
if ( /\G.../gc ) {  
} elsif ( /\G.../gc ) {  
} elsif ( /\G.../gc ) {
```

- pack and unpack

- ◆ 5.4: # () ! / D e E F g G j J k K m M o O r R t T U W y Y z Z
- ◆ 5.5: # () ! / D e E F g G j J k K m M o O r R t T U W y Y z
- ◆ 5.6: # () D e E F g G j J k K m M o O r R t T W y Y z
- ◆ 5.8: e E g G k K m M o O r R t T W y Y z

pack and unpack

- perl ops do the work

```
foreach my $bit (0..31) {  
    next unless vec ($_[0], $bit, 1);  
    # Do stuff.  
}
```

- regexp bytecode does the work

```
my $expanded = unpack "b*", $_[0];  
while ($expanded =~ /1/g) {  
    my $bit = pos ($expanded) - 1;  
    # Do stuff.  
}
```

- scale differently

- just bit 12
- bits 1, 24, 25, 26, 27, 28, 29, 30

speed or comfort?

- pack
- sprintf
- regexp
- perl

Banish the demons of Stupidity

- Arrays and Hashes
- Regexp
- pack and unpack
- undef
 - ◆ Are you throwing results away?
 - ◆ Make expensive calculations optional

AutoSplit and AutoLoader

- AutoLoader - only load the subroutines you use
- Can extend your existing &AUTOLOAD
- AutoSplit stubs your subroutines
- SelfLoader similar - but no auto stubs

AutoSplit tricks and gotchas

```
...  
1;  
# While debugging, disable AutoLoader like this:  
# __END__  
...
```

AutoSplit tricks and gotchas

- unexpected use strict;

```
use strict;
```

```
...
```

```
1;
```

```
# While debugging, disable AutoLoader like this:
```

```
# __END__
```

```
...
```

AutoSplit tricks and gotchas

- unexpected use strict;

```
use strict;
```

```
...
```

```
1;
```

```
# While debugging, disable AutoLoader like this:
```

```
# __END__
```

```
...
```

- evals subs in at run time - mod_perl?

=pod @ __END__

- Skipping over POD documentation not free

```
#!/perl -w  
use strict;
```

```
=head1 You don't want to do that
```

```
big block of pod
```

```
=cut
```

```
...  
1;  
__END__
```

```
=head1 You want to do this
```

- But maybe not your style
- Maybe not a huge gain
- **__END__** incompatible with Apache::Registry

Importing is slow

- Exporter is written in perl
- Exporter uses eval ""
- Many modules have huge @EXPORT

```
use POSIX;           # Exports all the defaults
use POSIX ();       # Exports nothing.
```

Memoize

- Caches function results
 - ◆ trades memory for speed
 - ◆ not for trivial functions
- For functions that only calculate
 - ◆ so no side effects (print, ...)
- Can tie cache to a disk file
 - ◆ and expire results
 - ◆ but lock the cache in a CGI

General optimising techniques

- Avoid string constructions

- ◆ String eval
- ◆ Symbolic references

- Pull things out of loops

```
my $type_func = $encode_types{$type};
```

- Tail recursion

- ◆ Rewrite as a loop
- ◆ Can replace any recursion with iteration

- Inline small functions

- Experiment with number of arguments

- ◆ Rather than having code that sets defaults, specify them explicitly

Regexps

- avoid `$&`
 - ◆ `$&` has dynamic scope (unlike everything else)

```
$text =~ /.*/;
$line = $&; # Now every match will copy $& - slow
$text =~ /(.* rules)/;
$line = $1; # Didn't mention $& - fast
```

Regexps

- avoid `$&`
 - ◆ `$&` has dynamic scope (unlike everything else)

```
$text =~ /.*/; # rules/
```

```
$line = $&; # Now every match will copy $& - slow
```

```
$text =~ /(.* rules)/;
```

```
$line = $1; # Didn't mention $& - fast
```

- avoid use English;
 - ◆ because it aliases `$&`

Regexps

- avoid `$&`
 - ◆ `$&` has dynamic scope (unlike everything else)

```
$text =~ /.*/; rules/;
```

```
$line = $&; # Now every match will copy $& - slow
```

```
$text =~ /(.* rules)/;
```

```
$line = $1; # Didn't mention $& - fast
```

- avoid use English;
 - ◆ because it aliases `$&`
- avoid needless captures
 - ◆ use `(?:)`

Regexps

- avoid `$&`
 - ◆ `$&` has dynamic scope (unlike everything else)

```
$text =~ /.*/; # rules/
```

```
$line = $&; # Now every match will copy $& - slow
```

```
$text =~ /(.* rules)/;
```

```
$line = $1; # Didn't mention $& - fast
```

- avoid use English;
 - ◆ because it aliases `$&`
- avoid needless captures
 - ◆ use `(?:)`
- `qr/.../o`;

Regexps

- avoid `$&`
 - ◆ `$&` has dynamic scope (unlike everything else)

```
$text =~ /.*/;
$line = $&; # Now every match will copy $& - slow
$text =~ /(.* rules)/;
$line = $1; # Didn't mention $& - fast
```

- avoid use English;
 - ◆ because it aliases `$&`
- avoid needless captures
 - ◆ use `(?:)`
- `qr/.../o`;

- `qr` once

```
sub foo {
    my $reg1 = qr/.../;
    my $reg2 = qr/... $reg1 .../;
# aaaaaaargh
```

tr

- `tr//!/` # fastest way to count chars
- `tr/q/Q/` faster than `s/q/Q/g`
- `tr/a-z//d` faster than `s/[a-z]//g`
 - ◆ but `tr/a//` slower than `s/a//g`

Dynamic templates

- `printf` templates don't have to be constant

```
#foreach my $c (split(//,$out_bytes)) {  
#   $s .= sprintf "\\x%02X",ord($c);  
#}  
# 9.5% faster changing that loop to this:  
$s .= sprintf +("\\x%02X" x length $out_bytes),  
              unpack "C*", $out_bytes;
```

Dynamic templates

- `printf` templates don't have to be constant

```
#foreach my $c (split(//,$out_bytes)) {  
#   $s .= sprintf "\\x%02X",ord($c);  
#}  
# 9.5% faster changing that loop to this:  
$s .= sprintf +("\\x%02X" x length $out_bytes),  
              unpack "C*", $out_bytes;
```

- neither do `pack/unpack` nor `regexps`

Hash keys

- Hash keys are strings
- Internally hashing "spreads" data around an array
 - ◆ scalar context gives hash quality

Passing data between processes

- Your parser written in perl
- perl's parser written in C

Passing data between processes

- Your parser written in perl
- perl's parser written in C
- Data::Dumper
 - ◆ Tweak it - `$Data::Dumper::Indent = 0;`
 - ◆ String eval

Passing data between processes

- Your parser written in perl
- perl's parser written in C
- Data::Dumper
 - ◆ Tweak it - `$Data::Dumper::Indent = 0;`
 - ◆ String eval
- Storable
 - ◆ Faster still
 - ◆ No string eval
 - ◆ But not indestructible

How to make perl fast enough

- use the language's fast features
- take advantage of perl's dynamic nature
- give the interpreter hints
- use less 'OPs'
- use less 'CPU'
- use less 'RAM'