The Perl-OpenMP Project

Part II:

State of the Art: Perl and Multithreading via OpenMP

Recap

- OpenMP is a way to "*easily*" make existing serial C, C++, and Fortran codes use mult-cores
- Industry standard, supported by GCC since v4.2 (2005)
- Declarative annotations in code behind code comments
- Has various constructs for sharing work
- Compiler provides a runtime
- Execution controlled via API and %ENV
- Perl-OpenMP Project was started to explore the space


OpenMP in OSS (updated)

- ImageMagick
- `rperl` uses it for some auto-parallelization (thanks to Will “the” Chill)

OpenMP in OSS (updated)

- ImageMagick
- `rperl` uses it for some auto-parallelization (thanks to Will “the” Chill)

Recap – Code Example of OpenMP

```
#include <stdio.h>
#include <omp.h>

int main (int argc, char*argv[]) {
  #pragma omp parallel
  {
    printf("hi, from thread %d!\n",
           omp_get_thread_num());
  }
  return 0;
}
```

```
prompt# gcc -fopenmp hi.c -o hi.x
prompt# OMP_NUM_THREADS=8 ./hi.x
hi, from thread 1!
hi, from thread 0!
hi, from thread 2!
hi, from thread 7!
hi, from thread 5!
hi, from thread 4!
hi, from thread 6!
hi, from thread 3!
prompt#
```

```
prompt# gcc -fopenmp hi.c -o hi.x
prompt# OMP_NUM_THREADS=4 ./hi.x
hi, from thread 1!
hi, from thread 0!
hi, from thread 2!
hi, from thread 3!
prompt#
```

```
prompt# gcc -fopenmp hi.c -o hi.x
prompt# OMP_NUM_THREADS=2 ./hi.x
hi, from thread 1!
hi, from thread 0!
prompt#
```

Objectives

- Demo 2 modules from the Project
- Discuss the future of Perl w/ OpenMP
  - In XS code
  - Shared libs via FFI
- Discuss the future of **`perl`** w/OpenMP
  - Other OSS projects use it
  - Can we?


Current CPAN Releases*

- **`Alien::OpenMP`**

  makes Inline::C with OpenMP easy

- **`OpenMP::Environment`**

  provides *perlish* way to manipulate environmental variables used by OpenMP at runtime.

** more on these in Part II!*


Alien::OpenMP

- Create after `OpenMP::Environment`, due to there being a *need* for it to use `O::E`.
- `A::O` is designed to *work* with `Inline::C`, which is meant to easily add C based custom libraries in Perl code
- `A::O` provides the compiler flags and `include` file injection to support OpenMP


Remainder of Talk Mostly ...

- Some slides
- A little demo
- Maybe live coding
- *Yay*!


OpenMP via `Inline::C`

- Works but is *gross and unnatural*
- Must specify compiler/linker flags
- Must manually add required "`#include`", e.g.,

```
#include <omp.h> // for gcc
```

- %ENV only read on library load (actually *expected*, but we want it possible to read on each function call)
- Example follows ...


OpenMP via Inline::C

```
#!/usr/bin/env perl
use strict;
use warnings;

# build and load subroutines
use Inline (
    C           => 'DATA',
    name        => q{Test},
    ccflagsex   => q{-fopenmp},
    lddlflags   => join( q{ }, $Config::Config{lddlflags}, q{-fopenmp} ),
);

# pass in number of threads, but this is not idiomatically OpenMP
for my $num_threads (qw/1 2 4 8 16 32 64 128 256/) {
    test($num_threads);
}

__DATA__
__C__
#include <omp.h>
#include <stdio.h>
void test(int num_threads) {
  omp_set_num_threads(num_threads);
  #pragma omp parallel
  {
    if (0 == omp_get_thread_num())
      printf("'%d' should be '%d'\n", omp_get_num_threads(), num_threads);
  }
}
```

`Inline::C` *with* `Alien::OpenMP`

- As clean as possible
- Handles compiler and linker flags
- Injects "`#include <omp.h>`"
- %ENV still only read on library load
  - Will address this, but not yet


Inline::C with Alien::OpenMP

```
#!/usr/bin/env perl
use strict;
use warnings;

# build and load subroutines
use Alien::OpenMP qw//;
use Inline (
    C           => 'DATA',
    with        => 'Alien::OpenMP',
);

# pass in number of threads, but this is not idiomatically OpenMP
for my $num_threads (qw/1 2 4 8 16 32 64 128 256/) {
    test($num_threads);
}

__DATA__
__C__
//NOTE: '#include <omp.h>' is not required bc Alien::OpenMP injects it
#include <stdio.h>
void test(int num_threads) {
  omp_set_num_threads(num_threads);
  #pragma omp parallel
  {
    if (0 == omp_get_thread_num())
      printf("'%d' should be '%d'\n", omp_get_num_threads(), num_threads);
  }
}
```

`OpenMP::Environment` + `system`

- Works as expected wrt `%ENV` because `system` operates under updated environment
- Perlish "*launcher*" script for OpenMP-enabled executables


OpenMP::Environment + system

```
#!/usr/bin/env perl
use strict;
use warnings;

use FindBin qw/$Bin/;
use lib qq{$Bin/../lib};
use Getopt::Long qw/GetOptionsFromArray/;
use Util::H2O::More qw/opt2h2o/;
use OpenMP::Environment ();

# init options
my @opts = (qw/threads=i/);
my $o = h2o {threads => 4};
my $ret = GetOptionsFromArray( \@ARGV, $o, @opts );

my $oenv = OpenMP::Environment->new;
$oenv->omp_num_threads($o->threads);
# run executable after setting OMP_NUM_THREADS

my $bin = $ARGV[-1];
my $exit_code = system($bin);
if ( $exit_code == 0 ) {
    print qq{OK - now do stuff after a successful execution\n};
}
else {
    print qq{Oof - something went wrong.\n};
    exit $exit_code;
}
```

Inline::C + Alien::OpenMP + OpenMP::Environment

- Also as clean as possible
- Provides *Perlish* %ENV controls
- Still has issue with stale %ENV, but can be solved by explicitly re-reading %ENV; e.g:

```
void _ENV_set_num_threads() {
  char *num;
  num = getenv("OMP_NUM_THREADS");
  omp_set_num_threads(atoi(num));
}
```


```
#!/usr/bin/env
use strict;
use warnings;
use OpenMP::Environment qw//;
use Alien::OpenMP qw//;
use Inline (
    C           => 'DATA',
    with        => 'Alien::OpenMP',
    BUILD_NOISY => 1,
);
my $oenv = OpenMP::Environment->new;
for my $num_threads (qw/1 2 4 8 16 24/) {
    $oenv->omp_num_threads($num_threads);
    test(); # in C, made to be sensitive to $ENV{OMP_NUM_THREADS}
}
__DATA__
__C__
#include <stdio.h>
#include <stdlib.h>
void test() {
  _ENV_set_num_threads(); // update via OMP_NUM_THREADS in %ENV
  #pragma omp parallel
  {
    if (0 == omp_get_thread_num())
      printf("%-2d threads\n", omp_get_num_threads());
  }
}
//  may provide this and other similar update methods via Inline::C injection
void _ENV_set_num_threads() {
  char *num;
  num = getenv("OMP_NUM_THREADS");
  omp_set_num_threads(atoi(num));
}
```

A::0 should provide some provided 'helper" C funcs

Currently "State of the Art"

```
#!/usr/bin/env
use strict;
use warnings;
use OpenMP::Environment qw//;
use Alien::OpenMP qw//;
use Inline (
    C           => 'DATA',
    with        => 'Alien::OpenMP',
    BUILD_NOISY => 1,
);
my $oenv = OpenMP::Environment->new;
for my $num_threads (qw/1 2 4 8 16 24/) {
    $oenv->omp_num_threads($num_threads);
    test(); # in C, made to be sensitive to $ENV{OMP_NUM_THREADS}
}
__DATA__
__C__
#include <stdio.h>
#include <stdlib.h>
void test() {
  _ENV_set_num_threads(); // update via OMP_NUM_THREADS in %ENV
  #pragma omp parallel
  {
    if (0 == omp_get_thread_num())
      printf("%-2d threads\n", omp_get_num_threads());
  }
}
```

A::O **should** provide some provided 'helper" C funcs

Possible improvement 1

```
#!/usr/bin/env
use strict;
use warnings;
use OpenMP::Environment qw//;
use Alien::OpenMP qw//;
use Inline (
    C           => 'DATA',
    with        => 'Alien::OpenMP::WithENV', # doesn't exist yet!
    BUILD_NOISY => 1,
);
my $oenv = OpenMP::Environment->new;
for my $num_threads (qw/1 2 4 8 16 24/) {
    $oenv->omp_num_threads($num_threads);
    test(); # in C, made to be sensitive to $ENV{OMP_NUM_THREADS}
}
__DATA__
__C__
#include <stdio.h>
#include <stdlib.h>
void test() {
  // with injected call to re-read %ENV
  #pragma omp parallel
  {
    if (0 == omp_get_thread_num())
      printf("%-2d threads\n", omp_get_num_threads());
  }
}
```

A::O should provide some provided 'helper" C funcs; and maybe insert them into the

Possible improvement 2

Tie::'d Example

```
#!/usr/bin/perl

use strict;
use warnings;
use FindBin qw/$Bin/;
use lib qq{$Bin/lib};

use Tie::Array::OmpSum;
use OpenMP::Environment qw//;

my @giant_list = (1 .. 100_000_000);
tie my $array_sum, 'Tie::Array::OmpSum', \@giant_list;

my $oe = OpenMP::Environment->new;
foreach my $num_threads (qw/1 2 4 8 16 20/) {
  $oe->omp_num_threads($num_threads);
  printf qq{%02d threads -> sum = $array_sum ...\n}, $num_threads;
}
```


Tie::'d Example

```
package Tie::Array::OmpSum;
use strict;
use warnings;
use Alien::OpenMP qw//;
use Inline (
    C           => 'DATA',
    with        => 'Alien::OpenMP',
);
sub TIESCALAR {
  my $pkg  = shift;
  return bless { array_ref => shift }, $pkg;
}
sub STORE {
  my ($self, $value) = @_;
  return;
}
sub FETCH {
  my ($self) = shift;
  my $sum = sum($self->{array_ref});
  return $sum;
}
__PACKAGE__
__DATA__
__C__
#include <stdio.h>
#include <stdlib.h>
SV *sum(SV *array) {  // stolen from Inline::C::Cookbook
    int numelts, i;
    if ((!SvROK(array))
        || (SvTYPE(SvRV(array)) != SVt_PVAV)
        || ((numelts = av_len((AV *)SvRV(array))) < 0)
    ) {
        return &PL_sv_undef;
    }

    /* read %ENV and update according to OMP_NUM_THREADS */
    _ENV_set_num_threads();
    int total = 0;
    #pragma omp parallel shared(total)
    {
      #pragma omp for reduction(+:total)
      for (i = 0; i <= numelts; i++) {
        total += SvIV(*av_fetch((AV *)SvRV(array), i, 0));
      }
    }
    return newSViv(total);
}
//  may provide this and other similar update methods via Inline::C injection
void _ENV_set_num_threads() {
  char *num;
  num = getenv("OMP_NUM_THREADS");
  omp_set_num_threads(atoi(num));
}
```

What's `next;`?

- OpenMP has been available in gcc since 2005!!!!1!111!!!!1
- It was designed to easily (and *incrementally*!) annotate existing sequential code (usually tightly coupled loops)
- Declarations hid behind code comments, so are invisible when OpenMP isn't available


Ways OpenMP May Benefit Perl

- `Tie::*` modules that provide SMP overlay for `ARRAY`s and `HASH`es (RO/reductions based at first); gain more experience
- `overload.pm` things?
- Gain more experience with using OpenMP in XS codes that deal more intimately with Perl API/internals data structures
- Via CPAN, provide SMP aware modules for speeding up *uniprocess* `perl`


Ways OpenMP May Benefit `perl`

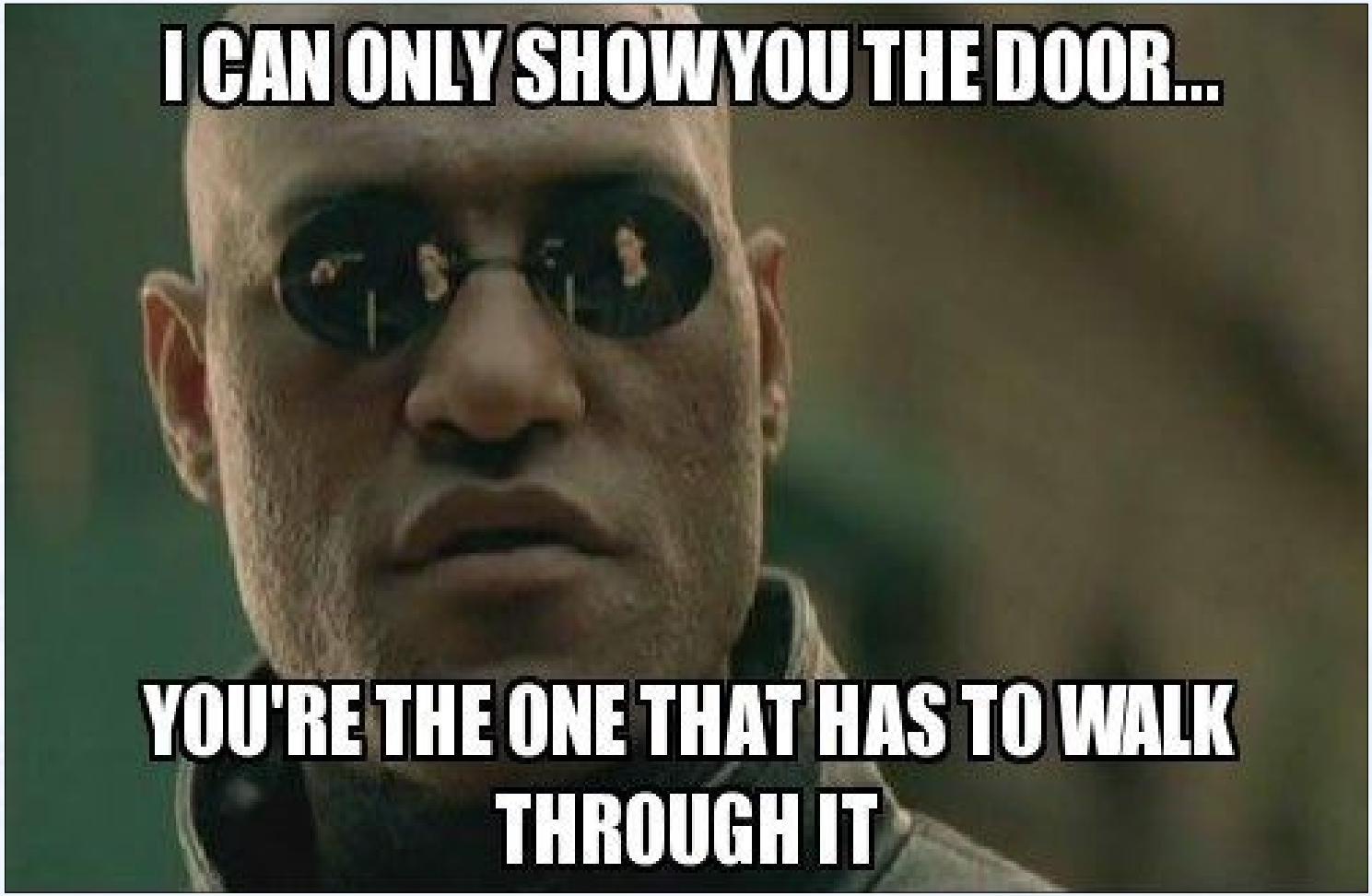
Move `perl` internals to a "*sequential except when it's not*" model … meaning:

- Identify and experiment with `perl` internals, for things that may safely be done in parallel (e.g., RO things, add `-fopenmp` to `perl` source build)
- Introduce a few more Perl %ENVs, e.g., `PERL_OMP_NUM_THREADS`
- Incrementally "*parallelize*" more fundamental operations on data structures
- Fast *and* "not fake" types (see `rperl`)


Role of Perl-OpenMP Project

- Make it as easy as possible to use OpenMP with `Inline::C`
- Break into XS + OpenMP, FFI + OpenMP
- Create *more* POC `Tie::*` modules
- Create POC XS codes with OpenMP
- Build awareness of OpenMP in Perl Comunity
- Maybe one day explore OpenMP in `perl`


Demo Time



And trust us on the Sun screen ...

Stay tuned!