Inline::C

Let's Go Native

Miguel Duarte - malduarte@gmail.com

O que é

- Permite ligar Perl com C
- Faz parte de uma família de módulos que integra Perl com outras linguagens (AWK, Java, Python, ...)

Porquê Perl + C?

- Solução 100% Perl demasiado lenta
- Reutilização de bibliotecas nativas
- Quando é mais fácil implementar em C

Porquê Inline::C

- Integrar Perl com C era demasiado complexo e trabalhoso para o comum dos mortais (perldoc perlxs, perlxstut, perlapi, perlguts, ...)
- Em 1996 o SWIG http://swig.org veio simplificar a vida, mas continuava a ser um pincel
- Em 2000 surge o Inline::C e a vida ficou mais simples

Olá Mundo!

```
use Inline C;
escreve("Ola Mundo!");
__END__
__C__

void escreve(char *mensagem) {
    puts(mensagem);
}
```

```
> perl exemplo_01.pl
Ola Mundo!
>
```

Sim! É mesmo só isto!

Como funciona?

1. Analisa as declarações de funções
2. Gera o código cola XS automaticamente
3. Calcula hash do código evitando recompilações
4. Gera uma biblioteca dinâmica (.so, .dll, ...)
5. Coloca a função no namespace Perl
6. Invoca a função

Caso da Vida Real

- Análise de Gbytes de tráfego num operador móvel
- Ficheiros de texto com campos delimitados por “;”
- Versão inicial em AWK com performance aceitável
- AWK já não era apropriado para o tipo de processamento que se pretendia

Conversão para Perl

AWK

```
BEGIN { FS=";" }
{
    # Processamento (...)
}
```

Perl

```
while(<>)
{
    @fields = split ';';
    # Processamento (...)
}
```

Versão em Perl 6x mais lenta!

Solução

- Reutilizar uma biblioteca que tinha desenvolvido três anos antes em C
- Não sabia do Inline!
- Perdi-me na documentação do XS
- Usei inicialmente o SWIG, mas lá acabei por descobrir o Inline::C que tratou da papinha toda

Duas alternativas

- Colocar o código fonte numa secção __C__e deixar o Inline::C compilá-lo
- Usar uma biblioteca pré compilada

```
use Inline C => Config => ENABLE => AUTOWRAP
             => MYEXTLIB => $ENV{'HOME'} . '/lib/registo.so';
use Inline C => q{
int parse_record(void* vp_dr, char* line, char delimiter);
void* create_record(int max_fields);
char* get_field(void* vp_dr, int index);
};
```

Resultados

- Ficou 2x mais rápido que a versão original em AWK com ~ 0 esforço (a biblioteca já existia)

```
cmpthese( -3, {
    'Split' => sub { split /;/, $str },
    'Record' => sub { parse_record($rec, $str, ";") }
});
```

```
perl registo.pl
            Rate  Split Record
Split   2111/s     --   -95%
Record 39938/s  1792%     --
```

O que optimizar?

- Benchmarks, Benchmaks, Benchmarks...
- Analisar o desempenho e perceber onde se está a gastar mais tempo
- As características de cada SO/Hardware podem ter impacto

E agora os “Contras”

- Performance vs. complexidade acrescida
- Pode ficar mais lento!
- Dependência de um compilador de C
- Portabilidade entre SO’s
- Problemas de algoritmia: um bom algoritmo em Perl é melhor que um mau algoritmo em C

Sequência de Fibonacci

$$F_n := F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

Implementação “ingénua” em Perl

```perl
sub fib_ingenuo_em_perl {
    my $n = pop;
    if($n == 0) { return 0; }
    if($n == 1 || $n == 2) { return 1; }
    return fib_ingenuo_em_perl($n-1) + fib_ingenuo_em_perl($n-2);
}
```

Ui! $O(\phi^N)$

Tunning I

- Agora que sei do Inline::C vou poder resolver os mistérios do "Código da Vinci" mais depressa!

```c
int fib_ingenuo_em_c(int n) {
    if(n == 0) return 0;
    if(n == 1 || n == 2) return 1;
    return fib_ingenuo_em_c(n - 1) + fib_ingenuo_em_c(n - 2);
}
```

```
perl fib_naive.pl 20
                 Rate Ingenuo Perl    Ingenuo C
Ingenuo Perl 23.7/s             --         -100%
Ingenuo C    6393/s         26878%           --
```

Rescaldo do Tunning I

- Inline::C tornou o código mais rápido
- A partir de certos valores de N, ambas as implementações são inaceitáveis!
- Um mau algoritmo em Perl continuará a ser um mau algoritmo em C (é o caso)
- Caramba, vamos ter de pensar!!

Tunning II

- Estou a calcular os mesmos valores vezes sem conta!
- Toca de implementar uma cache

```
@cache = ();
$cache[0] = 0;
$cache[1] = 1;
$cache[2] = 1;
```

```
sub fib_em_perl {
    my $n = pop;
    $cache[$n] = fib_em_perl($n - 1) + fib_em_perl($n - 2)
            unless defined $cache[$n];
    return $cache[$n];
}
```

```
perl fib_naive.pl 20
                Rate Ingenuo Perl    Ingenuo C       Em Perl
Ingenuo Perl  23.7/s             --       -100%         -100%
Ingenuo C     6392/s         26873%          --           -9%
Em Perl       7013/s         29496%         10%           --
```

Rescaldo do Tunning II

- Um bom algoritmo em Perl bate um mau algoritmo em C (ou qualquer outra linguagem)
- Um bom algoritmo em Perl pode oferecer performances tais que não justifiquem o uso do Inline::C

Tunning III

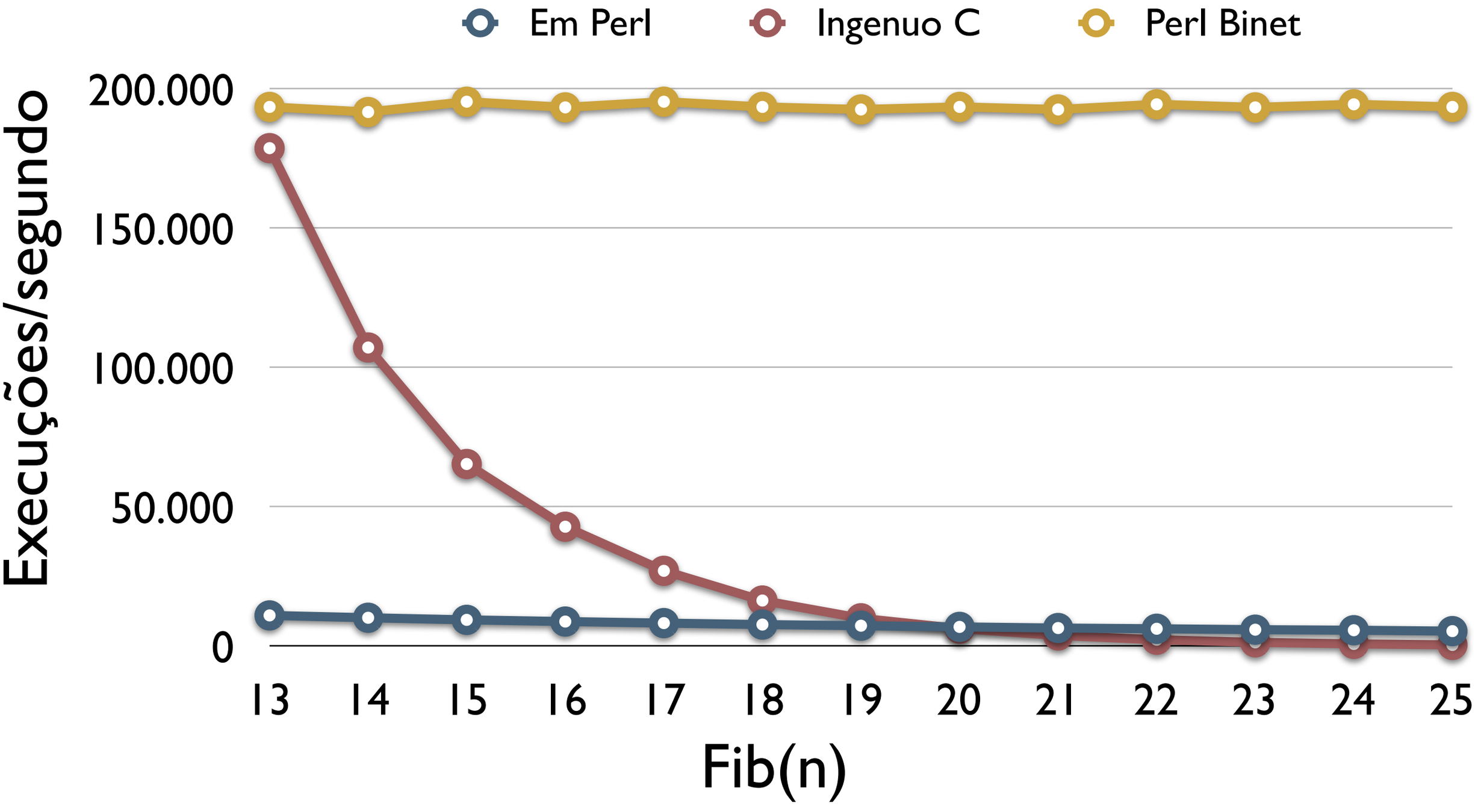
- A Velocidade vicia. Queremos sempre mais!

$$\varphi = \frac{1+\sqrt{5}}{2}. \qquad F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$$

```
my $phi = (1 + 5 ** 0.5) / 2;
sub fib_binet {
    my ($n) = @_;
    return int(round(($phi**$n - (1-$phi)**$n) / 5 ** 0.5));
}
```

```
 perl fib_naive.pl 20
                Rate Ingenuo C   Em Perl PerlBinet
Ingenuo C     6392/s         --      -9%      -97%
Em Perl       7013/s        10%       --      -96%
PerlBinet   192752/s      2916%    2648%        --
```

Evolução Performance



Rescaldo do Tunning III

- Umas horas a vasculhar artigos pode valer bem a pena
- Passou-se de um algoritmo de complexidade exponencial, para um linear O(N) para um cujo tempo de calculo é constante!
- Inline::C perfeitamente desnecessário!

Conclusões

- O Inline::C pode trazer ganhos de performance interessantes
- É muito fácil de usar - desde que saibam C :-)
- É necessário investigar bem a causa da performance deficiente.
- Desenvolver em C irá dar muito mais trabalho e trazer ganhos pouco significativos com o tempo gasto em elaborar um melhor algoritmo em Perl

Só mais uma coisa...

- Vejam o Capitulo 9 do "*Advanced Perl Programming 2nd Edition*" - Inline Extensions
- Vejam o perldoc do Inline::C e o Cookbook associado.
- Cobrem usos avançados do Inline que vos podem ser uteis:
  - Arrays, Hashes