Optimización de programas en Perl usando el profiler NYTProf para analizar su código fuente

Índice

- Introducción
  - ¿Qué es Perl?
  - Algunos profiler para Perl
- NYTProf
  - Descripción del supuesto
  - Analizando el código con NYTProf
- Optimización del código
- Resultados de la optimización

Introducción

1. ¿Qué es Perl?

Perl es un lenguaje de programación creado por Larry Wall, el 18 de diciembre de 1987.

En Programming Perl [1] se dice que “Perl is designed to make the easy jobs easy, without making the hard jobs impossible.”.

Por último, comentar que dispone de una comunidad muy activa de desarrolladores.

Introducción

2. Algunos profiler para Perl

   - Devel::DProf
   - Devel::SmallProf
   - Cache::Profile
   - Devel::NYTProf

NYTProf

NYTProf es el profiler elegido para el experimento, ya que dispone de una gran cantidad de funcionalidades distintas.

NYTProf son dos profiler en uno. Uno es un profiler de sentencias, y el otro es un profiler de subrutinas.

1. Descripción del supuesto

   El supuesto realizara una encriptación de un número, realizando una serie operaciones con unos números pasados enfichero.

NYTProf

1. Descripción del supuesto

- Primer paso.

```
sub primera_parte_encriptacion{
        my($num, @vec) = @_;
        for(my $i=0; $i<@vec; $i++){
                for(my $j=$i; $j<@vec; $j++){
                        $num = suma($num, $vec[$j]);
                }
        }
        return $num;
}
```

- Segundo paso.

```
sub segunda_parte_encriptacion{
        my($num, @vec) = @_;
        for(my $i=0;$i<@vec;$i++){
                if($i == 0){
                        $num = div_truncada($num, $vec[$i]);
                }
                elsif($i != 0 && ($i%7)==0){
                        $num = suma($num, $vec[$i]);
                }
                elsif($i != 0 && ($i%3)==0){
                        $num = resta($num, $vec[$i]);
                }
                elsif($i != 0 && ($i%2)==0){
                        $num = suma($num, $vec[$i]);
                }
                else{
                        $num = resta($num, $vec[$i]);
                }
        }
        return $num;
}
```

NYTProf

1. Descripción del supuesto

   - Tercer paso.

```
sub ordenacion_burbuja{
    my(@vec) = @_;
        for(my $i=0; $i<@vec; $i++){
            for(my $j=$i; $j<@vec; $j++){
                if($vec[$i] > $vec[$j]){
                    my $temp = $vec[$j];
                    $vec[$j] = $vec[$i];
                    $vec[$i] = $temp;
                }
            }
        }
        return @vec;
}
sub tercera_parte_encriptacion{
        my($num, @vec) = @_;
        @vec = ordenacion_burbuja(@vec);
        my $n = scalar(@vec);
        return (mull($num, $vec[resta(div_truncada($n, 2), 1)]));
}
```

NYTProf

2. Analizando el código con NYTProf

NYTProf nos da diversas herramientas con las cuales analizar los datos, como:

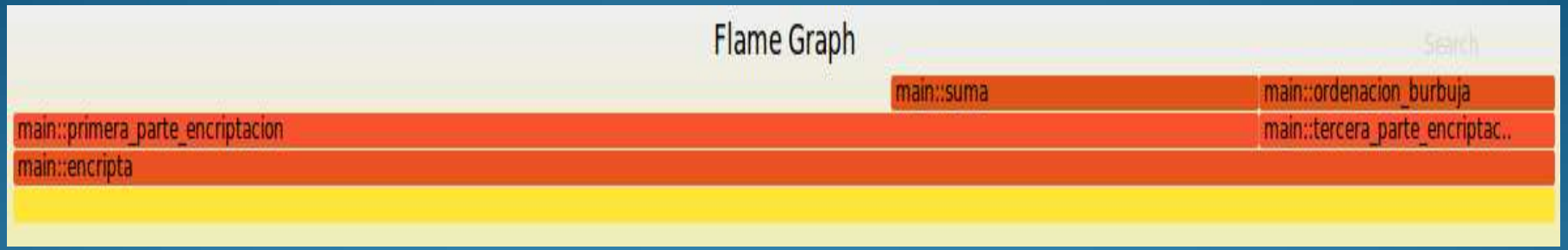
- Flame Graph
- Top 15 subrutinas
- Tree map del tiempo exclusivo de subrutinas
- Grafo de llamadas entre subrutinas
- Grafo de llamada entre paquetes
- Ficheros de código fuente ordenados por tiempo exclusivo

```
MemoriaISE $ time perl -d:NYTProf programa.pl 20 ./numeros.txt
Numero encriptado : -570376278976677


real    24m36.927s
user    24m34.936s
sys     0m1.296s
MemoriaISE $
```

```
MemoriaISE $ time perl programa.pl 20 ./numeros.txt
Numero encriptado : -570376278976677


real    4m1.944s
user    4m1.840s
sys     0m0.008s
MemoriaISE $
```

NYTProf

Flame Graph



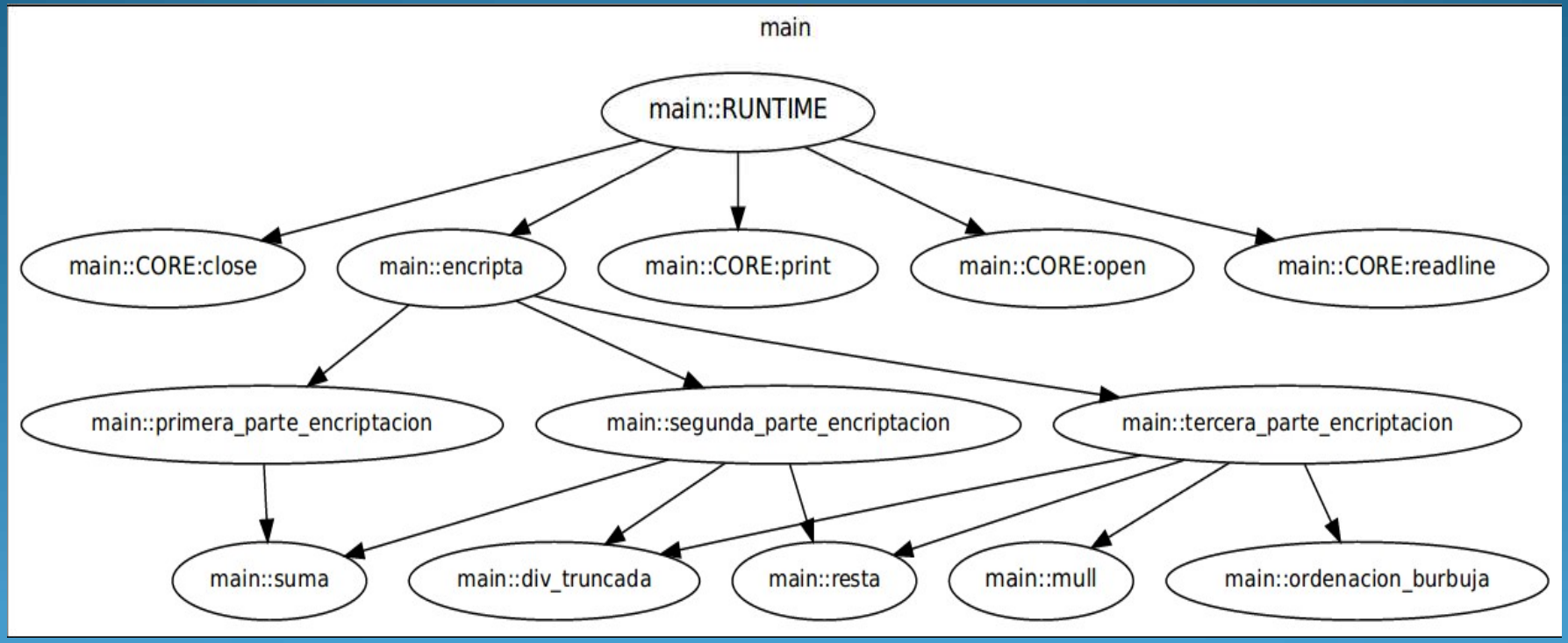
Top 15 Subrutine

Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
1	1	1	564s	801s	main::primera_parte_encriptacion
200018571	2	1	236s	236s	main::suma
1	1	1	190s	190s	main::ordenacion_burbuja
1	1	1	81.7ms	108ms	main::segunda_parte_encriptacion
1	1	1	24.0ms	24.0ms	main::CORE:readline (opcode)
11429	3	1	14.6ms	14.6ms	main::resta
1	1	1	9.67ms	190s	main::tercera_parte_encriptacion
1	1	1	3.80ms	991s	main::encripta
1	1	1	722µs	722µs	main::CORE:print (opcode)
2	2	1	36µs	36µs	main::div_truncada
1	1	1	32µs	32µs	main::CORE:open (opcode)
1	1	1	29µs	29µs	main::CORE:close (opcode)
1	1	1	5µs	5µs	main::mull

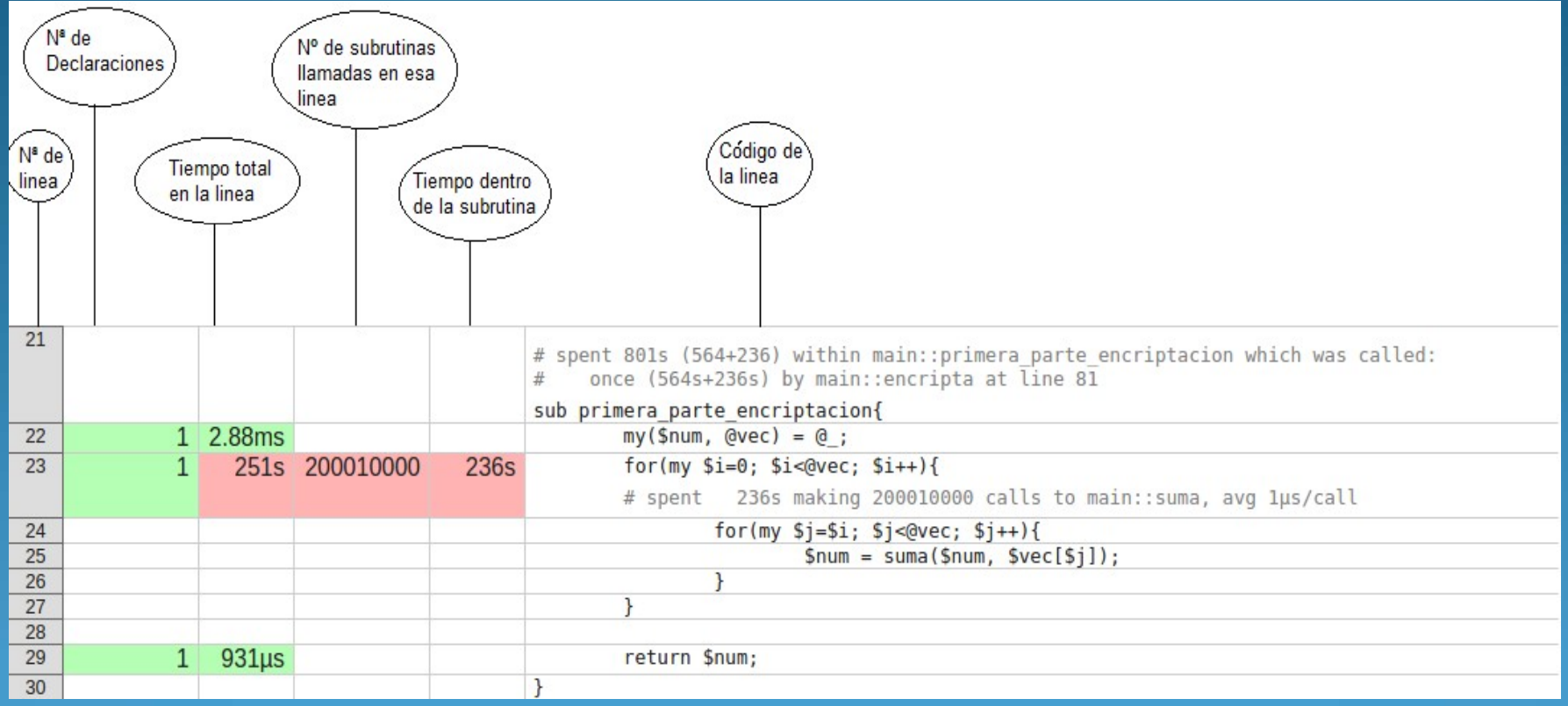
NYTProf

Grafo de llamadas entre subrutinas



NYTProf

2. Analizando el código con NYTProf



21					`# spent 801s (564+236) within main::primera_parte_encriptacion which was called:`
					`#  once (564s+236s) by main::encripta at line 81`
					`sub primera_parte_encriptacion{`

22	1	2.88ms			`  my($num, @vec) = @_;`
23	1	251s	200010000	236s	`  for(my $i=0; $i<@vec; $i++){` `  # spent  236s making 200010000 calls to main::suma, avg 1µs/call`
24					`  for(my $j=$i; $j<@vec; $j++){`
25					`  $num = suma($num, $vec[$j]);`
26					`  }`
27					`  }`
28					
29	1	931µs			`  return $num;`
30					`}`

NYTProf

2. Analizando el código con NYTProf

55					`# spent 190s within main::ordenacion_burbuja which was called:`
					`#  once (190s+0s) by main::tercera_parte_encriptacion at line 73`
					`sub ordenacion_burbuja{`

56	1	2.86ms			`  my(@vec) = @_;`
57					
58	1	14.2ms			`  for(my $i=0; $i<@vec; $i++){`
59	20000	72.6s			`  for(my $j=$i; $j<@vec; $j++){`
60	200010000	57.7s			`  if($vec[$i] > $vec[$j]){`
61	99421415	17.2s			`  my $temp = $vec[$j];`
62	99421415	21.3s			`  $vec[$j] = $vec[$i];`
63	99421415	21.4s			`  $vec[$i] = $temp;`
64					`  }`
65					`  }`
66					`  }`
67					
68	1	3.78ms			`  return @vec;`
69					`}`
70					

71					`# spent 190s (9.67ms+190) within main::tercera_parte_encriptacion which was called:`
					`#  once (9.67ms+190s) by main::encripta at line 83`
					`sub tercera_parte_encriptacion{`

72	1	2.27ms			`  my($num, @vec) = @_;`
73	1	6.12ms	1	190s	`  @vec = ordenacion_burbuja(@vec);` `  # spent  190s making 1 call to main::ordenacion_burbuja`
74	1	4µs			`  my $n = scalar(@vec);`

75	1	1.01ms	3	30µs	`  return (mull($num, $vec[resta(div_truncada($n, 2), 1)]));`
					`  # spent  20µs making 1 call to main::div_truncada`
					`  # spent  5µs making 1 call to main::mull`
					`  # spent  5µs making 1 call to main::resta`

76					`}`

Optimización del código

```
sub primera_parte_encriptacion{
            my($num, @vec) = @_;
            for(my $i=0; $i<@vec; $i++){
                $num = suma($num, mull($vec[$i],suma($i,1)));
            }
            return $num;
}
```

```
use sort '_mergesort';
```

```
sub tercera_parte_encriptacion{
    my($num, @vec) = @_;
       @vec = sort { $a <=> $b } @vec;
       my $n = scalar(@vec);
       return (mull($num, $vec[resta(div_truncada($n, 2), 1)]));
}
```

Resultados de la optimizacion