Programmation Parallèle en Perl

David Couvin

Institut Pasteur de la Guadeloupe

Institut Pasteur de la Guadeloupe

Université des Antilles

Sommaire

- Définition du langage Perl
- Pour qui?
- Pourquoi?
- Programmation Parallèle
- Exemples

Programmation Perl

- Perl est un langage de programmation créé par Larry Wall en 1987 pour traiter facilement de l'information de type textuel. Ce langage, interprété, s'inspire des structures de contrôle et d'impression du langage C, mais aussi de langages de scripts sed, awk et shell.

  Source : https://fr.wikipedia.org/wiki/Perl_(langage)

Comprehensive Perl Archive Network

- Le Comprehensive Perl Archive Network, ou CPAN, est un site Web consacré au langage de programmation Perl. CPAN désigne également un module Perl servant à accéder à ce site. Son nom vient du Comprehensive TeX Archive Network, ou CTAN, son homologue consacré à TeX.

https://fr.wikipedia.org/wiki/Comprehensive_Perl_Archive_Network

Syntaxe et éléments de base

```
# Je suis un commentaire classique

my $variable = 1; # Je suis un commentaire après une ligne de code

=for comment

    Je suis un commentaire
    sur plusieurs lignes.

=cut
```

Syntaxe et éléments de base

Perl est typé statiquement de façon simple : le premier caractère d'un identificateur de variable est un caractère non alphanumérique appelé sigil:

- le sigil dollar **$** dénote une variable scalaire (comme pour les shells Unix)

```
$a = 42;              # Affectation de '42' à la variable 'a'
$b = 'réponse:';      # Affectation d'une chaîne de caractère à 'b'
print "$b", $a * $a; # Affiche 'réponse:1764'
```

- le sigil arobase **@** désigne une variable tableau et est indexé par **[]** (ils peuvent être utilisés comme des piles ou des files, point commun avec Javascript)

```
@a = ('lun', 'mar', 'mer', 'jeu', 'ven', 'sam', 'dim'); # Affecte une liste a la variable tableau 'a'
print $a[2]; # Affiche 'mer' (l'indexation commence a zéro)

# récupération de plusieurs éléments ou d'une tranche d'un tableau
my @b = @a[2, 4]; # @b = ( 'mer', 'ven' )
my @c = @a[2..4]; # @c = ( 'mer', 'jeu', 'ven' )
```

Syntaxe et éléments de base

- le sigil pourcent **%** un tableau associatif, aussi appelé *hashage* ou *hash et est indexé par* **{}**

```
%a = ( # Création d'une table de hachage clef => valeur
  John  => 'Sheridan',
  Londo => 'Mollari',
  Kosh  => 'Naranek'
);
print $a{Londo}; # Affiche 'Mollari'
```

Contrairement à Perl 6 le sigil $ est utilisé lors de l'appel d'un élément d'un tableau ou d'un élément de hash.

```
$a[2];    # Appel d'un élément de tableau/liste
$a{John}; # Appel d'un élément de table de hash
```

Les trois types de variables du même nom peuvent coexister:

```
$a = 'rien'; # scalaire contenant la chaine 'rien'
@a = 1..5;   # tableau contenant les chiffres '1, 2, 3, 4, 5'
%a = (       # table de hash contenant 3 paires de clef => valeur
  John  => 'Sheridan',
  Londo => 'Mollari',
  Kosh  => 'Naranek'
);
```

Le module Parallel::ForkManager

- Ce module est destiné à être utilisé dans des opérations qui peuvent être effectuées en parallèle où le nombre de processus à répartir devrait être limité. Une utilisation typique est un téléchargeur qui récupérera des centaines / milliers de fichiers.
- https://metacpan.org/pod/Parallel::ForkManager

Exemple de code sans fork

```
use strict;
use warnings;
use Data::Dumper qw(Dumper);

my @numbers = map { $_ * 2000000 } reverse 1 .. 10;
my %results;

foreach my $q (@numbers) {
    $results{$q} = calc($q);
}

print Dumper \%results;

sub calc {
    my ($n) = @_;
    my $sum = 0;
    for (1 .. $n) {
        $sum += 1 + rand()/100;
    }
    return $sum;
}
```

Exemple de code parallélisé

```
use strict;
use warnings;
use Parallel::ForkManager;
use Data::Dumper qw(Dumper);

my $forks = shift or die "Usage: $0 N\n";

my @numbers = map { $_ * 2000000 } reverse 1 .. 10;
my %results;

print "Forking up to $forks at a time\n";
my $pm = Parallel::ForkManager->new($forks);

$pm->run_on_finish( sub {
    my ($pid, $exit_code, $ident, $exit_signal, $core_dump, $data_structure_reference) = @_;
    my $q = $data_structure_reference->{input};
    $results{$q} = $data_structure_reference->{result};
});

foreach my $q (@numbers) {
    my $pid = $pm->start and next;
    my $res = calc($q);
    $pm->finish(0, { result => $res, input => $q });
}
$pm->wait_all_children;

print Dumper \%results;
```

Exemple de code parallélisé (suite)

```
sub calc {
    my ($n) = @_;
    my $sum = 0;
    for (1 .. $n) {
        $sum += 1 + rand()/100;
    }
    return $sum;
}
```

https://perlmaven.com/speed-up-calculation-by-running-in-parallel

Autres exemples

- https://blog-en.openalfa.com/how-to-perform-parallel-processing-in-perl
- https://bioinformaticsonline.com/pages/view/37590/parallel-processing-with-perl