Fichier `polynomes.c`

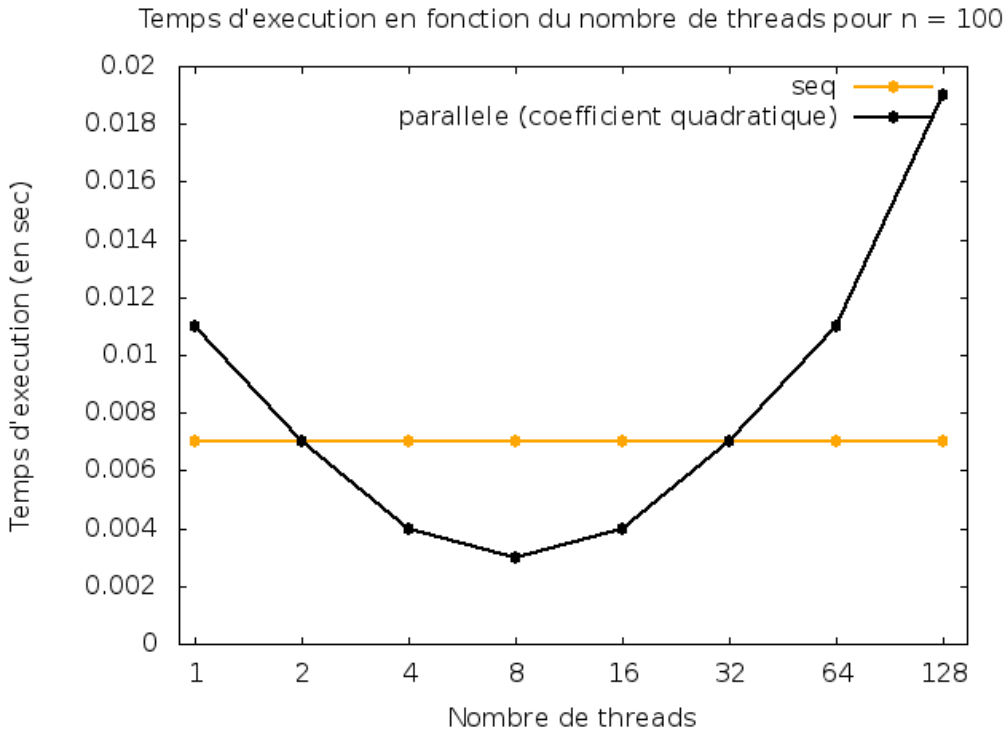
1 Fonction `fois`

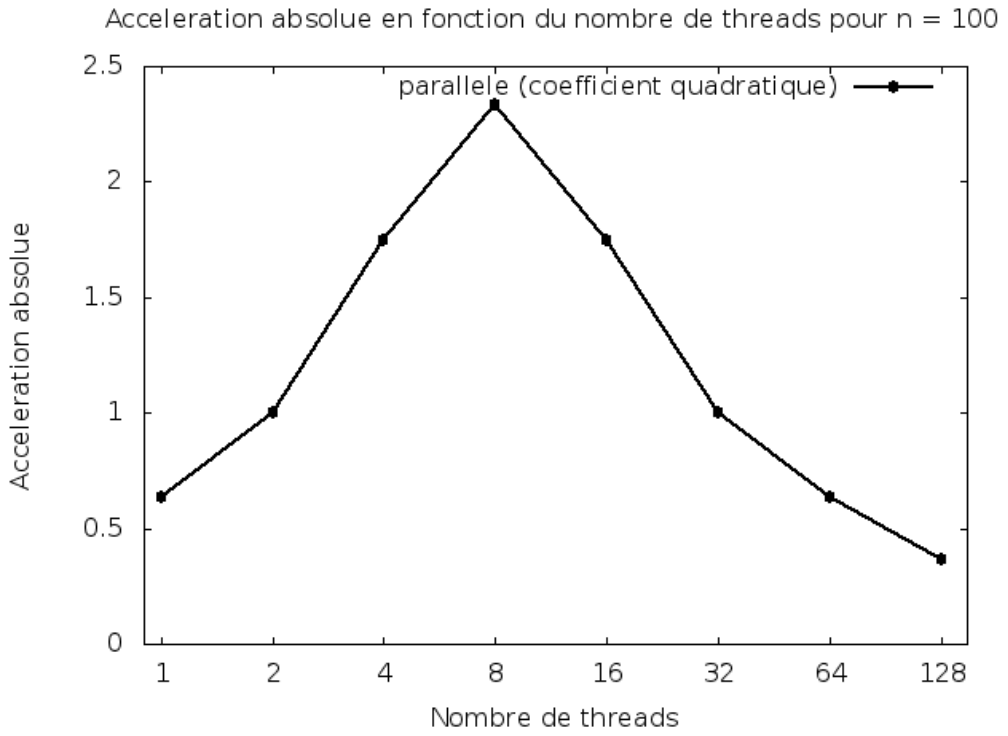
```
Polynome fois( Polynome p1, Polynome p2 )
{
  Polynome p = allouer( p1.degre + p2.degre );

  if ( sequentiel() ) {
    for( int i = 0; i <= p.degre; i++ ) {
      p.coefficients[i] = coefficient( i, p1, p2 );
    }
  } else {
    parallel_for( blocked_range<size_t>(0, p.degre+1),
                  [=]( blocked_range<size_t> r ) {
                    for( int i = r.begin(); i < r.end(); i++ ) {
                      p.coefficients[i] = coefficient( i, p1, p2 )
                    }
                  }
                  );
  }
  return p;
}
```

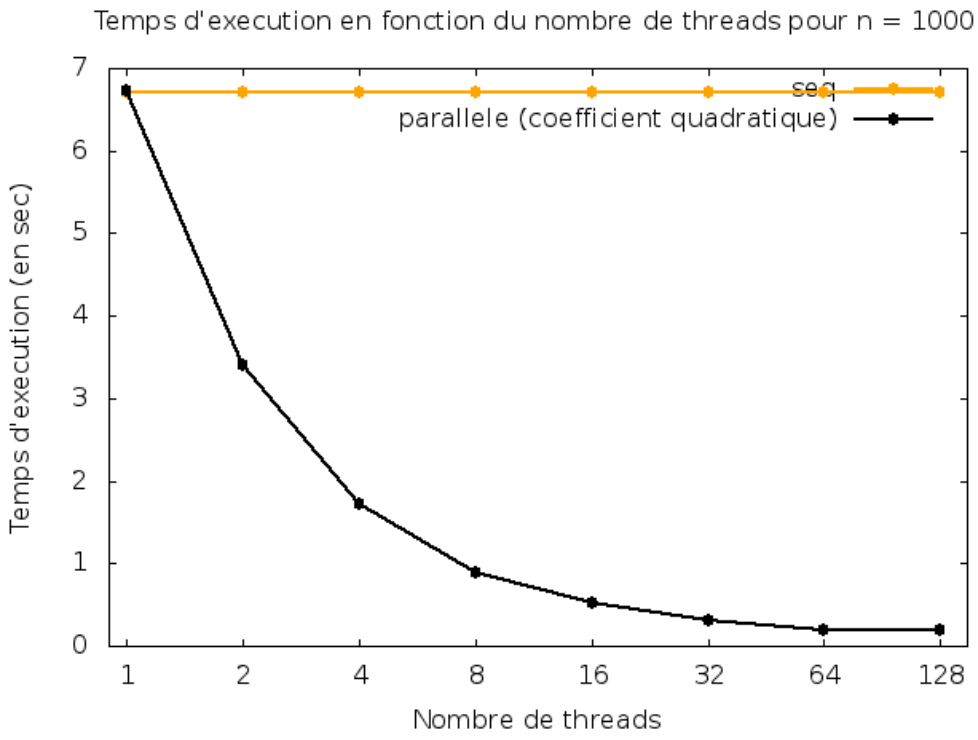
2 Fonction `coefficient` de complexité quadratique

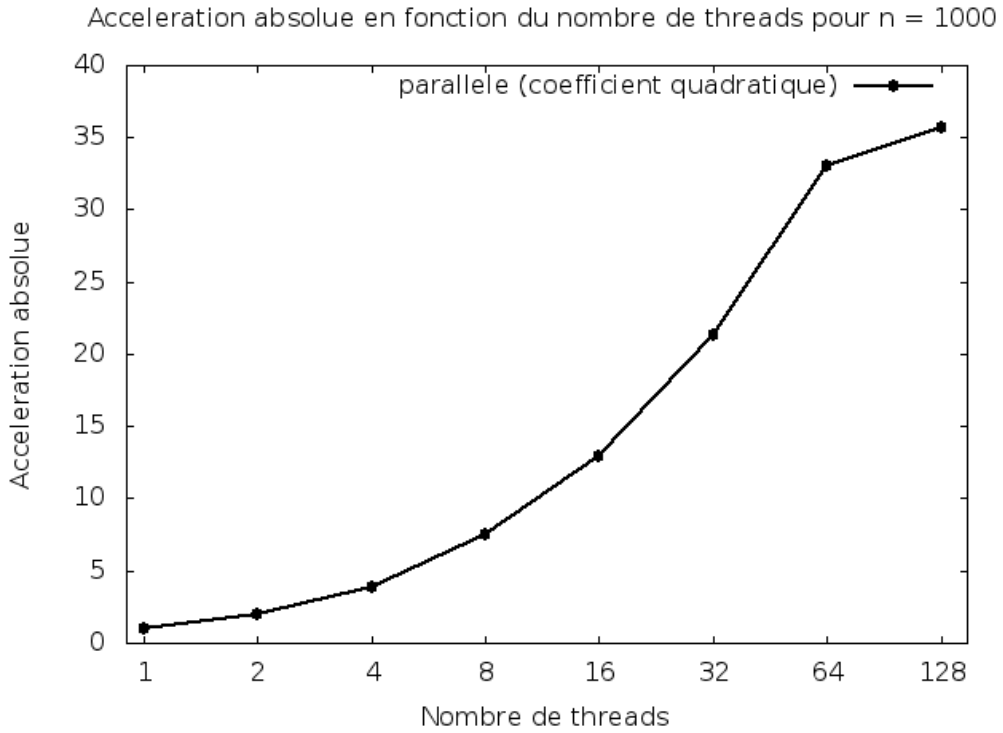
```
static double coefficient( int k, Polynome p1, Polynome p2 )
{
  if ( sequentiel() ) {

    double c = 0.0;
    for( int i = 0; i <= p1.degre; i++ ) {
      for( int j = 0; j <= p2.degre; j++ ) {
        if( i+j == k ) {
          c += p1.coefficients[i] * p2.coefficients[j];
        }
      }
    }
    return c;

  } else {

    return
        parallel_reduce( blocked_range<size_t>(0, p1.degre+1),
                         0.d,
                         [=]( blocked_range<size_t> r, double in ) {
                           double c = in;
                           for( int i = r.begin(); i < r.end(); i++ ) {
                             for( int j = 0; j <= p2.degre; j++ ) {
                               if( i+j == k ) {
                                 c += p1.coefficients[i] * p2.coefficients[j];
                               }
                             }
                           }
                           return c;
                         },
                         std::plus<double>()
                         );

    }
}
```

Temps d'exécution et accélérations de la version quadratique



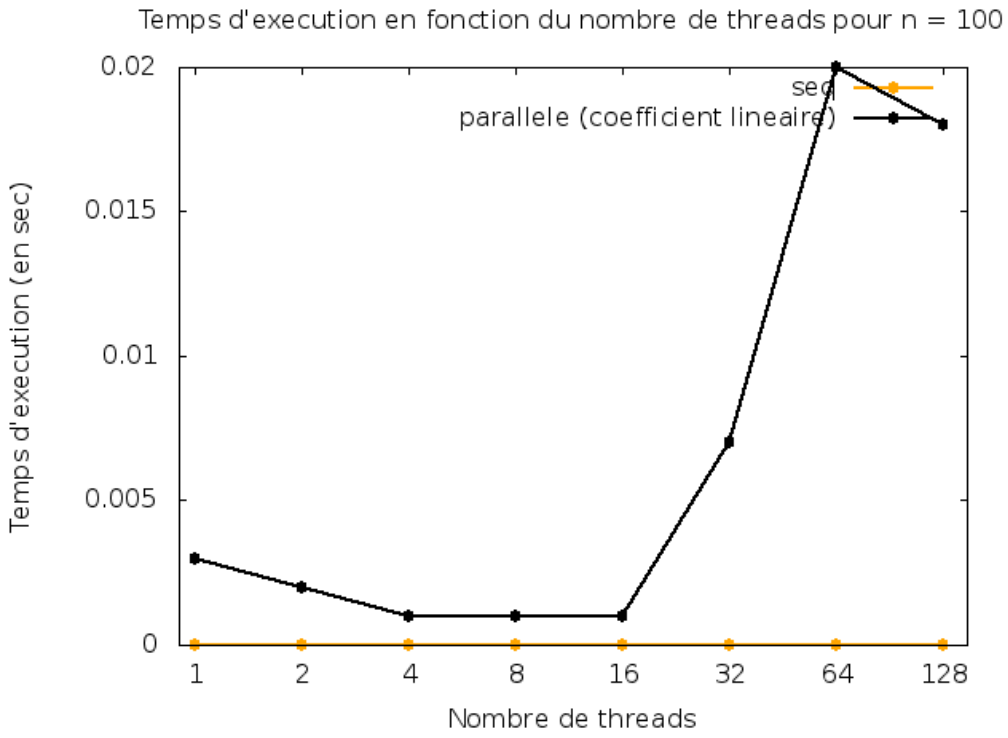


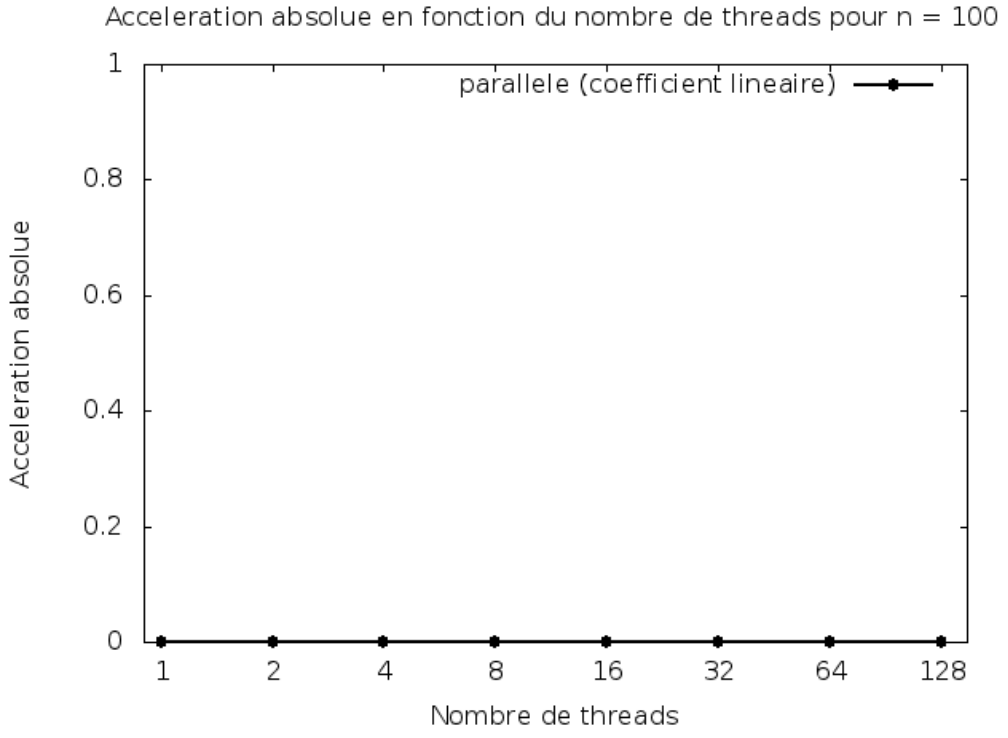

Temps d'execution en fonction du nombre de threads pour n = 1000
Temps d'execution (en sec)
Nombre de threads
seq
parallele (coefficient quadratique)
0
1
2
3
4
5
6
7
1
2
4
8
16
32
64
128

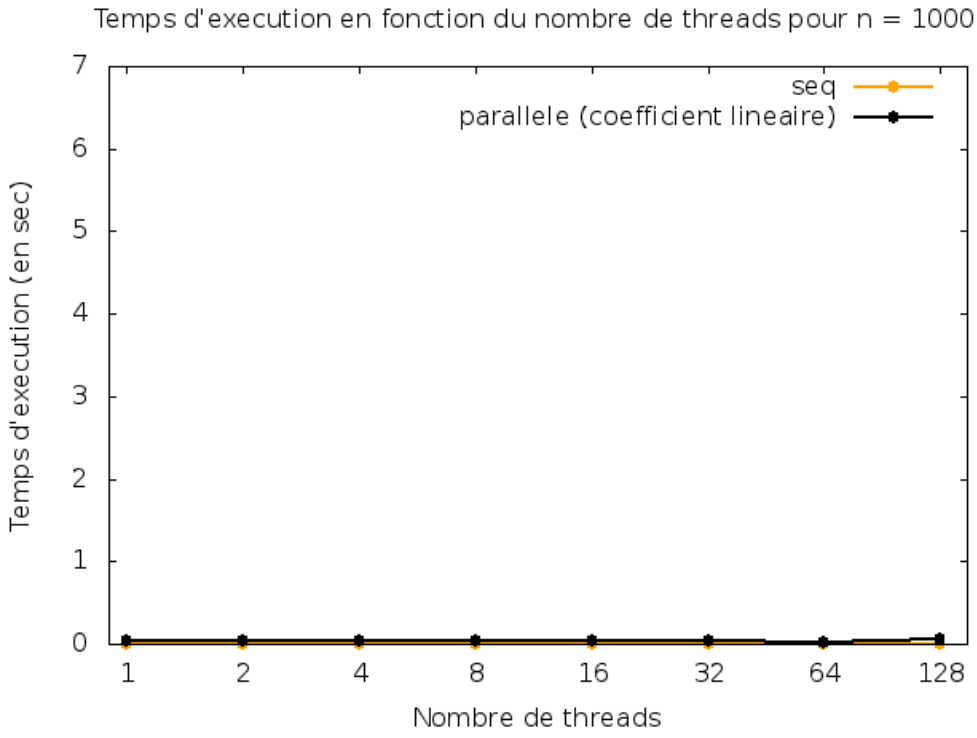

Acceleration absolue en fonction du nombre de threads pour n = 1000
Acceleration absolue
Nombre de threads
parallele (coefficient quadratique)
0
5
10
15
20
25
30
35
40
1
2
4
8
16
32
64
128

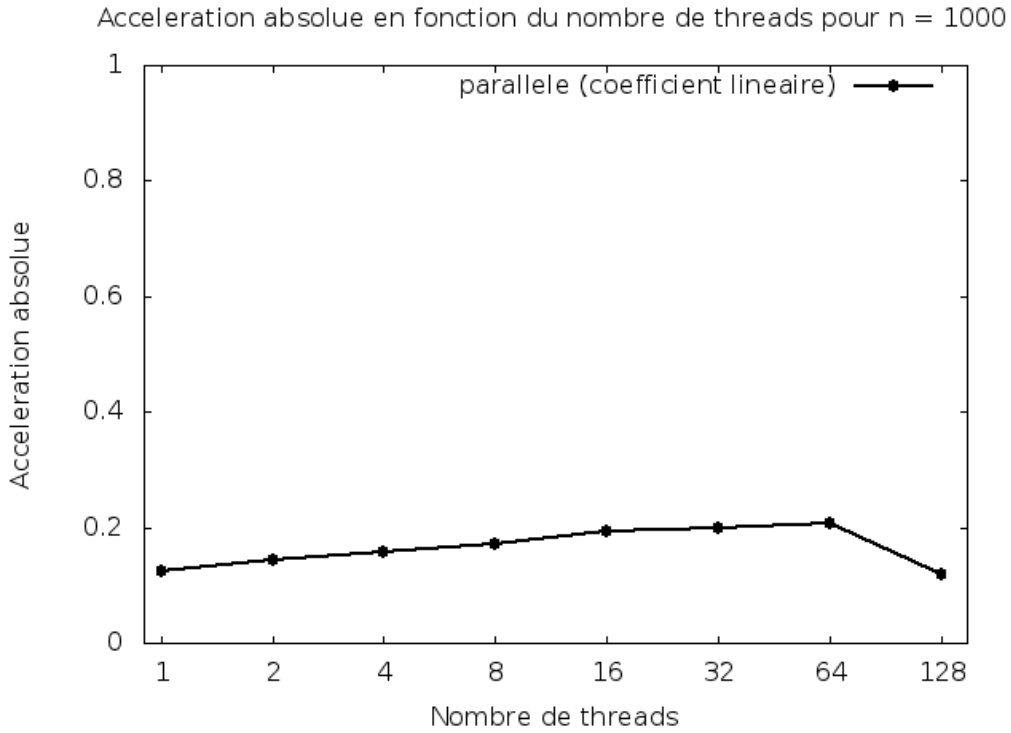
3 Fonction `coefficient` de complexité linéaire

```
static double coefficient( int k, Polynome p1, Polynome p2 )
{
  int expMinR1 = MAX( 0, k-p2.degre );
  int expMaxR1 = MIN( k, p1.degre );

  assert( expMinR1 <= expMaxR1 && "expMinR1 doit etre <= expMaxR1??" );

  if ( sequentiel() ) {

    double c = 0.0;
    for( int i = expMinR1; i <= expMaxR1; i++ ) {
      c += p1.coefficients[i] * p2.coefficients[k-i];
    }
    return c;

  } else {

    return
      parallel_reduce( blocked_range<size_t>(expMinR1, expMaxR1+1),
                       0.d,
                       [=]( blocked_range<size_t> r, double in ) {
                         double c = in;
                         for( int i = r.begin(); i < r.end(); i++ ) {
                           c += p1.coefficients[i] * p2.coefficients[k-i];
                         }
                         return c;
                       },
                       std::plus<double>()
                       );

  }
}
```

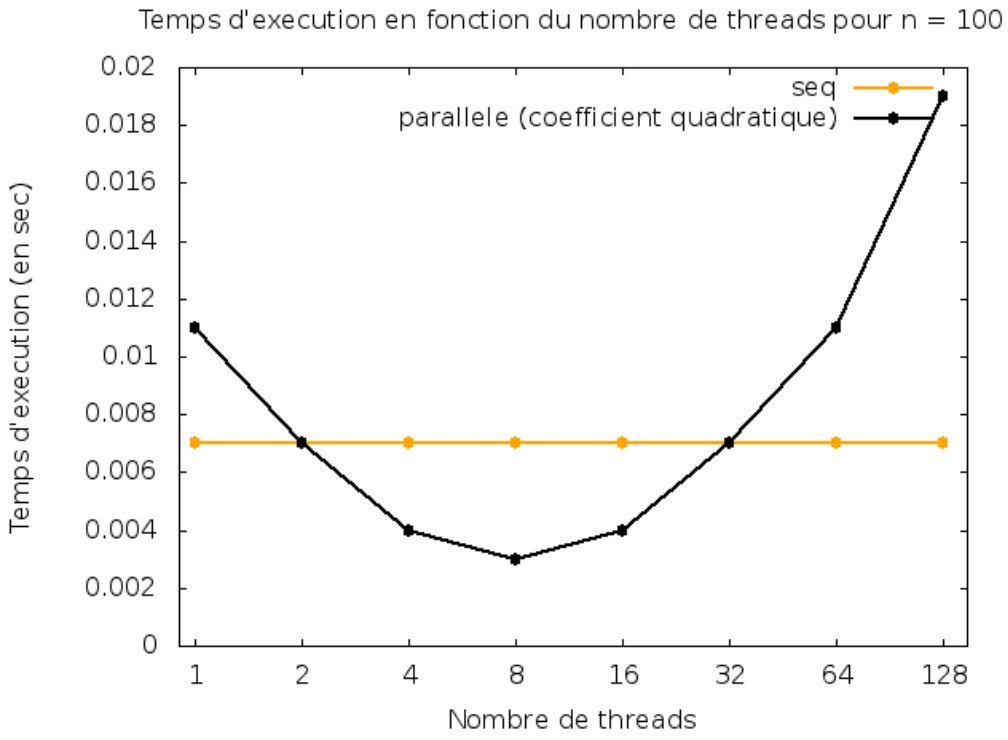

Temps d'execution en fonction du nombre de threads pour n = 100
Temps d'execution (en sec)
0.02
0.015
0.01
0.005
0
seq
parallele (coefficient lineaire)
1
2
4
8
16
32
64
128
Nombre de threads

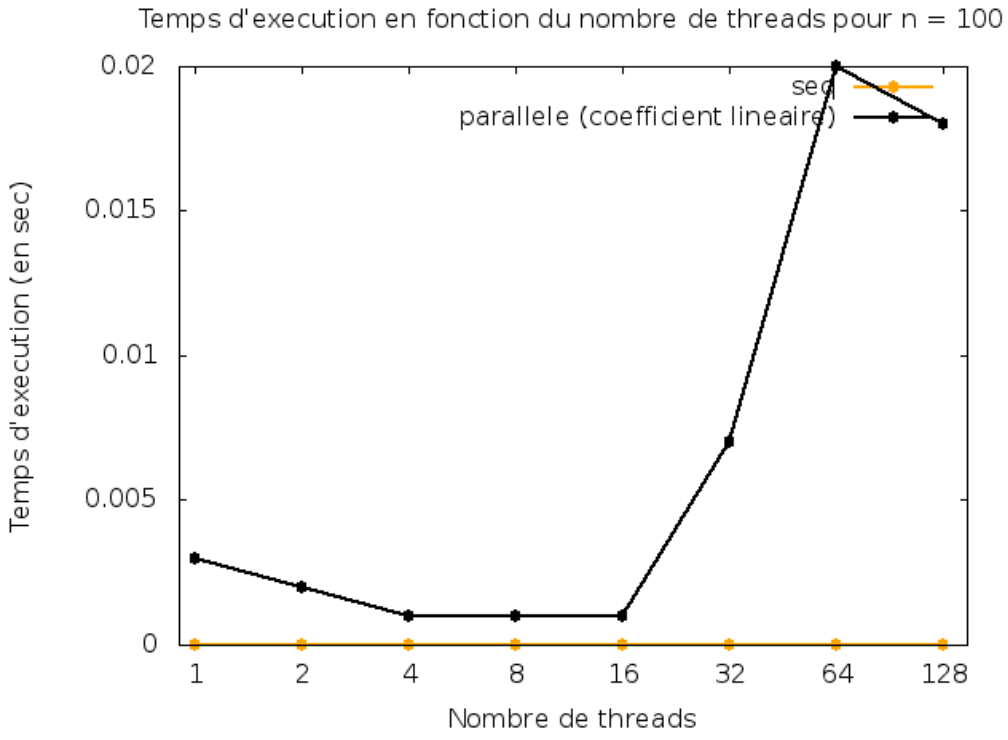

Acceleration absolue en fonction du nombre de threads pour n = 100
Acceleration absolue
1
0.8
0.6
0.4
0.2
0
parallele (coefficient lineaire)
1
2
4
8
16
32
64
128
Nombre de threads


Temps d'execution en fonction du nombre de threads pour n = 1000
Temps d'execution (en sec)
Nombre de threads
seq
parallele (coefficient lineaire)
0
1
2
3
4
5
6
7
1
2
4
8
16
32
64
128

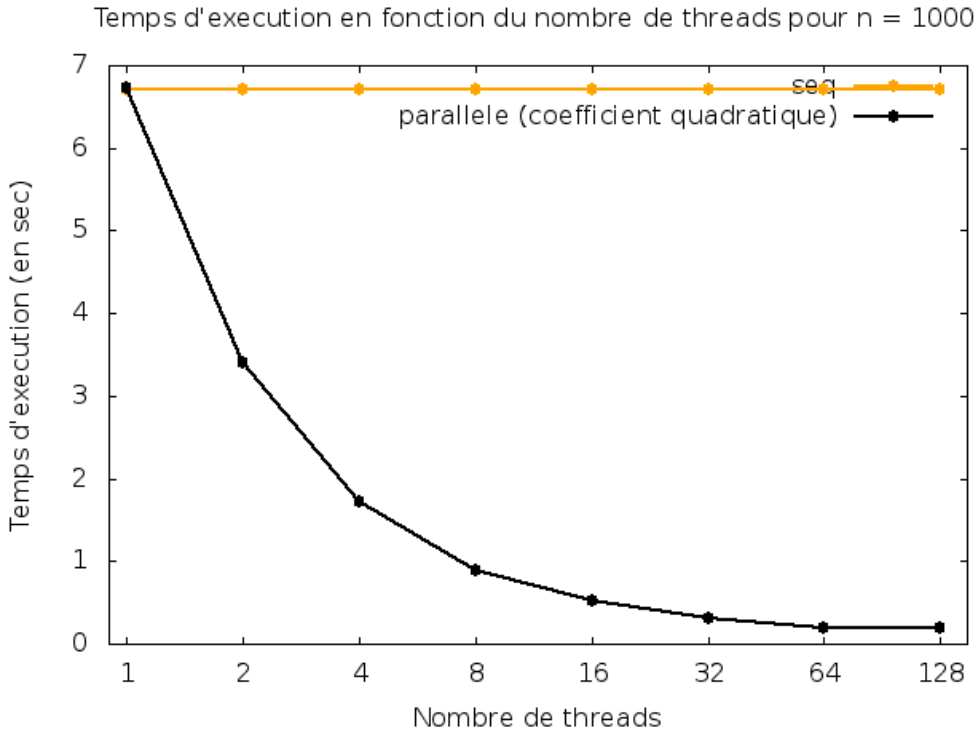

Acceleration absolue en fonction du nombre de threads pour n = 1000
Acceleration absolue
Nombre de threads
parallele (coefficient lineaire)
0
0.2
0.4
0.6
0.8
1
1
2
4
8
16
32
64
128

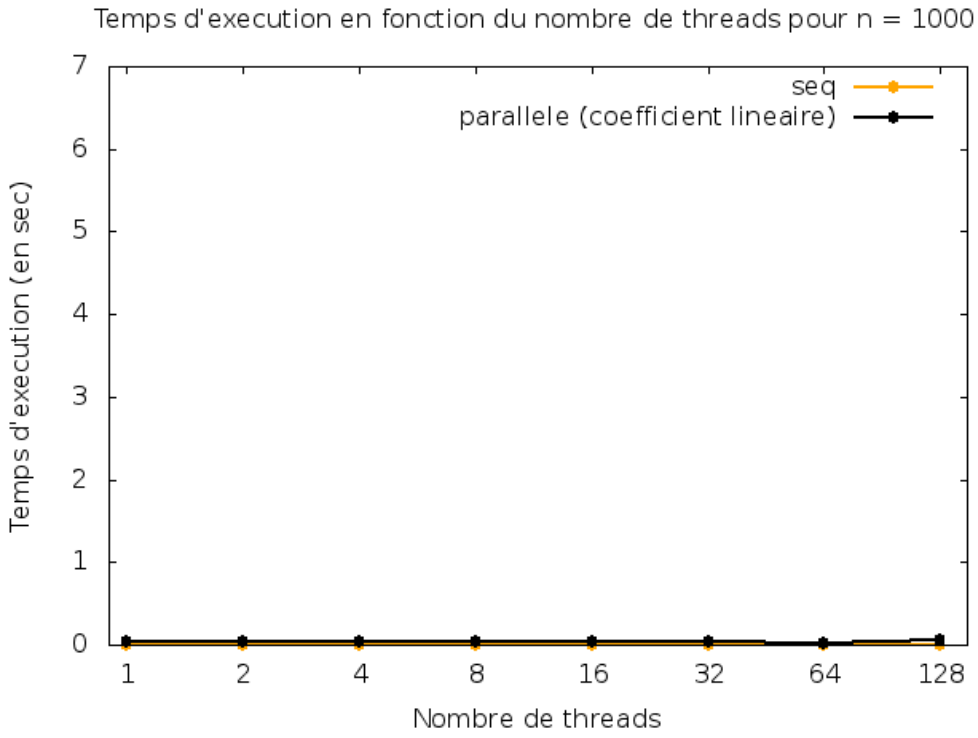
Comparaisons temps quadratique vs. linéaire pour `n = 100`



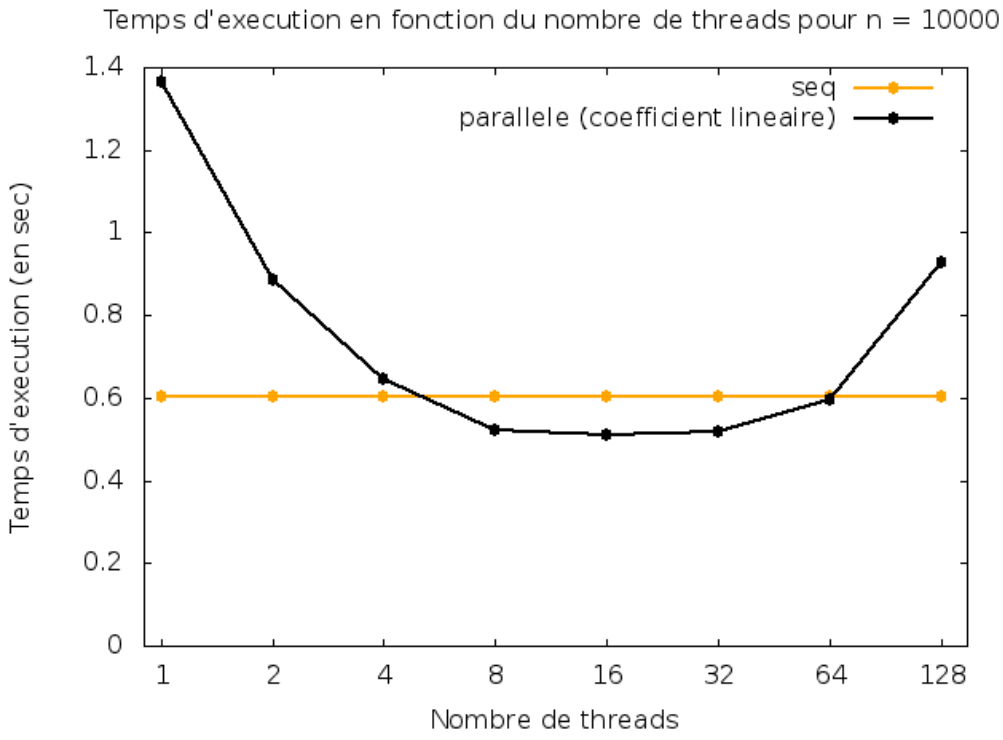


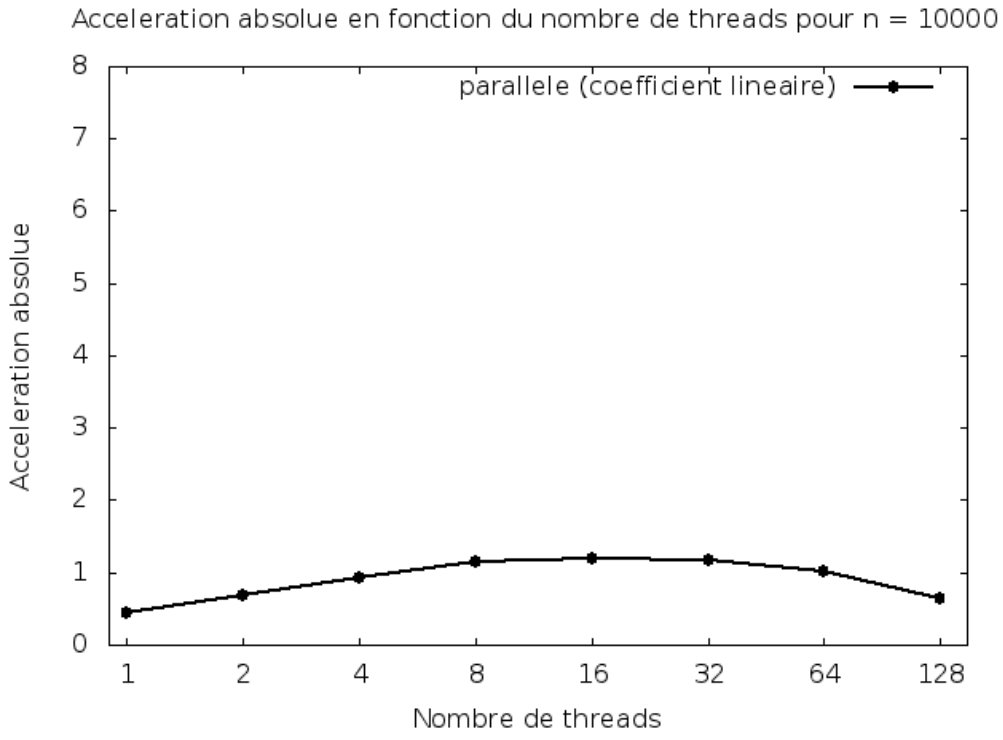
Comparaisons temps quadratique vs. linéaire pour `n = 1000`)

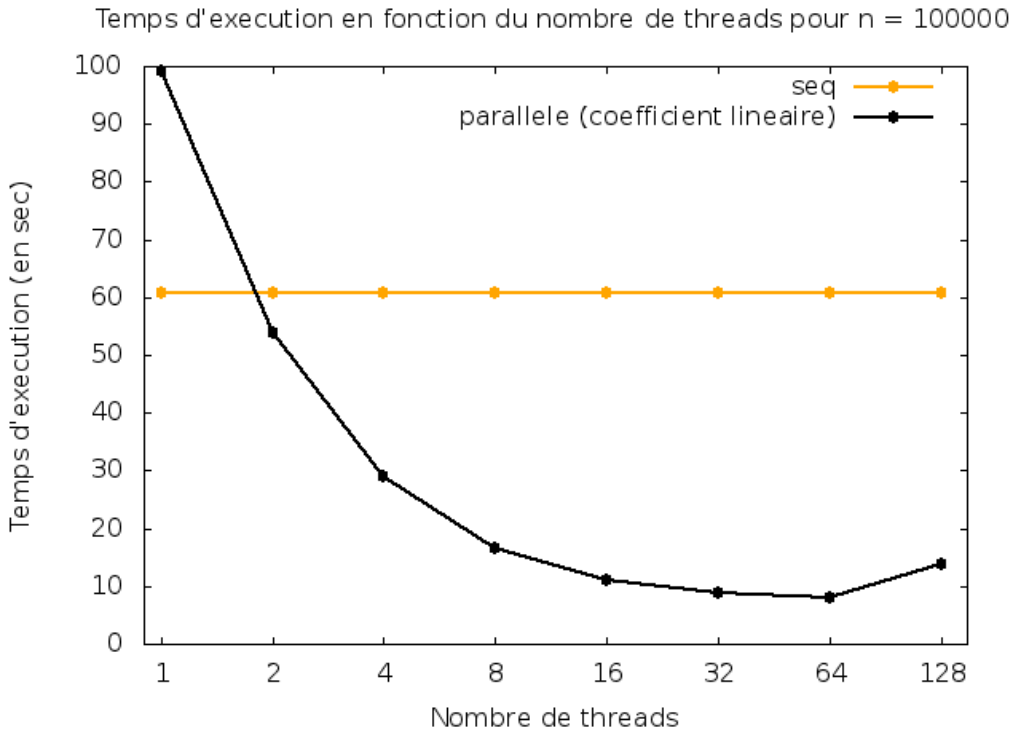


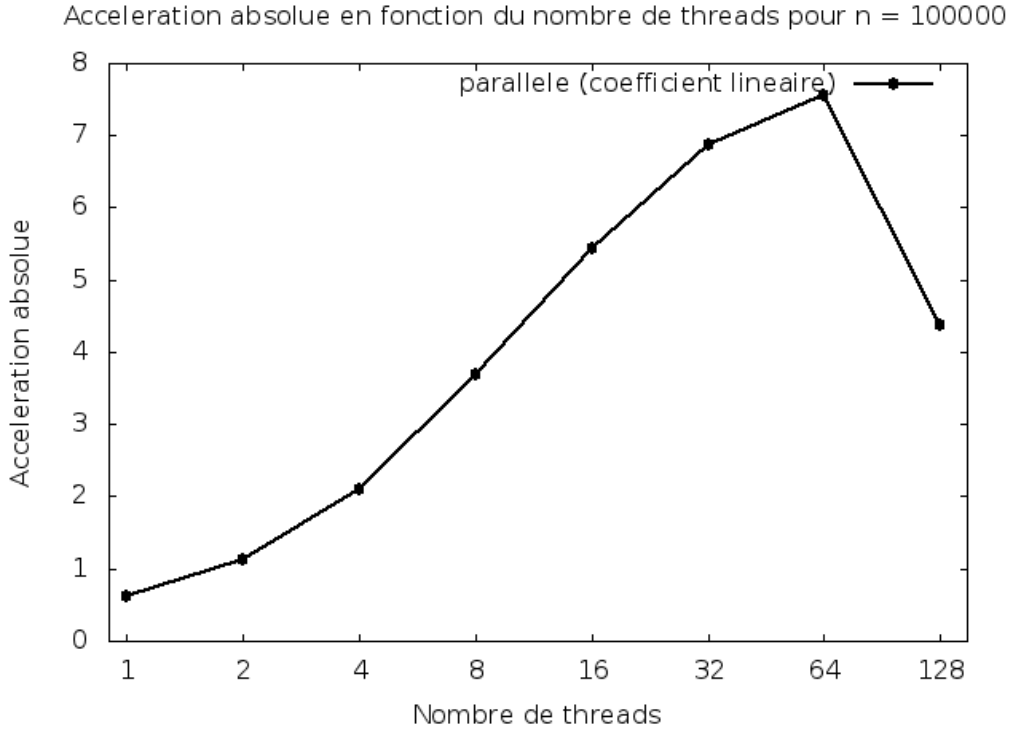


Temps et accélération pour `n = 10000` et `n = 100000`






Temps d'execution en fonction du nombre de threads pour n = 100000
Temps d'execution (en sec)
Nombre de threads
seq
parallele (coefficient lineaire)
0
10
20
30
40
50
60
70
80
90
100
1
2
4
8
16
32
64
128


Acceleration absolue en fonction du nombre de threads pour n = 100000
Acceleration absolue
Nombre de threads
parallele (coefficient lineaire)
0
1
2
3
4
5
6
7
8
1
2
4
8
16
32
64
128

4 Conclusion

Il est relativement facile d'obtenir de bonnes accélération... quand le programme qu'on veutre paralléliser est très peu performant (sic)!

Le temps pour la version quadratique pour `n = 10000` a été omis... parce qu'après une demi-heure, j'étais tanné d'attendre le résultat et j'ai tué le programme!