

Rapport projet laboratoire M1
Programmation parallèle pour réseau de neurones

ABDELMOUMENE Djahid

October 7, 2019

1 Introduction

Dans ce projet nous essayons de tester plusieurs techniques d'optimisation parallèle, et éventuellement les mettre en œuvre sur une implémentation d'une bibliothèque des réseaux neurones.

2 Les techniques d'optimisation utilisées

2.1 Algorithme naïf

Nous prenons comme point de référence l'algorithme naïf avec trois boucles imbriquées:

```
1 for(i=0; i<N; i++) {
2     for(j=0; j<N; j++) {
3         acc = 0.0;
4         for(x=0; x<N; x++) {
5             acc += a[i][x] * b[x][j];
6         }
7         res[i][j] = acc;
8     }
9 }
```

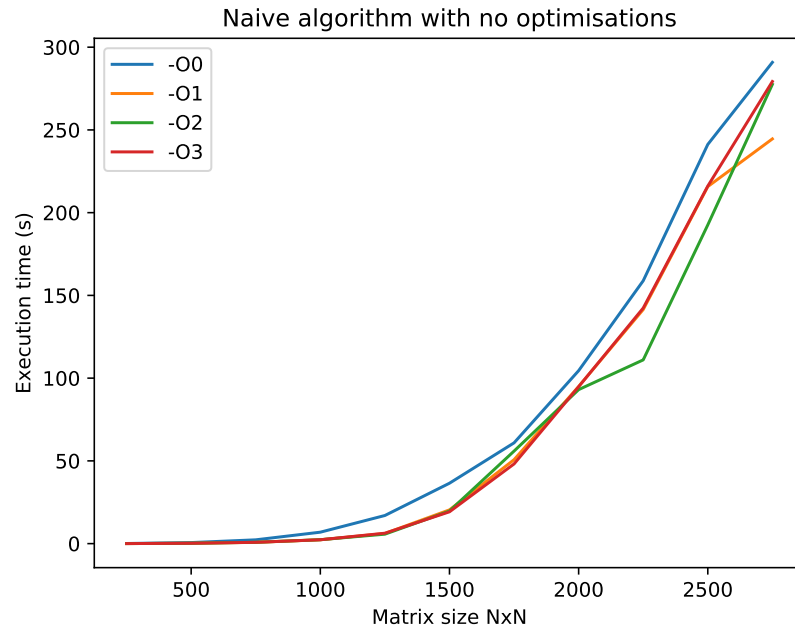


Figure 1: Pas d'optimisations

Ce que nous pouvons remarquer ici, c'est que l'effet des niveaux d'optimisation de GCC commence à faiblir quand N devient de plus en plus grand.

2.2 Optimisations de boucles imbriquées

Pour les optimisations

```

1  int ib = 10, kb = 10;
2  for (ii = 0; ii < N; ii += ib) {
3      for (kk = 0; kk < N; kk += kb) {
4          for (j=0; j < N; j += 2) {
5              for(i = ii; i < ii + ib; i += 2 ) {
6                  if (kk == 0)
7                      acc00 = acc01 = acc10 = acc11 = 0;
8                  else {
9                      acc00 = res[i + 0][j + 0];
10                     acc01 = res[i + 0][j + 1];
11                     acc10 = res[i + 1][j + 0];
12                     acc11 = res[i + 1][j + 1];
13                 }
14                 for (k = kk; k < kk + kb; k++) {
15                     acc00 += b[k][j + 0] * a[i + 0][k];
16                     acc01 += b[k][j + 1] * a[i + 0][k];

```

```

17         acc10 += b[k][j + 0] * a[i + 1][k];
18         acc11 += b[k][j + 1] * a[i + 1][k];
19     }
20     res[i + 0][j + 0] = acc00;
21     res[i + 0][j + 1] = acc01;
22     res[i + 1][j + 0] = acc10;
23     res[i + 1][j + 1] = acc11;
24 }
25 }
26 }
27 }

```

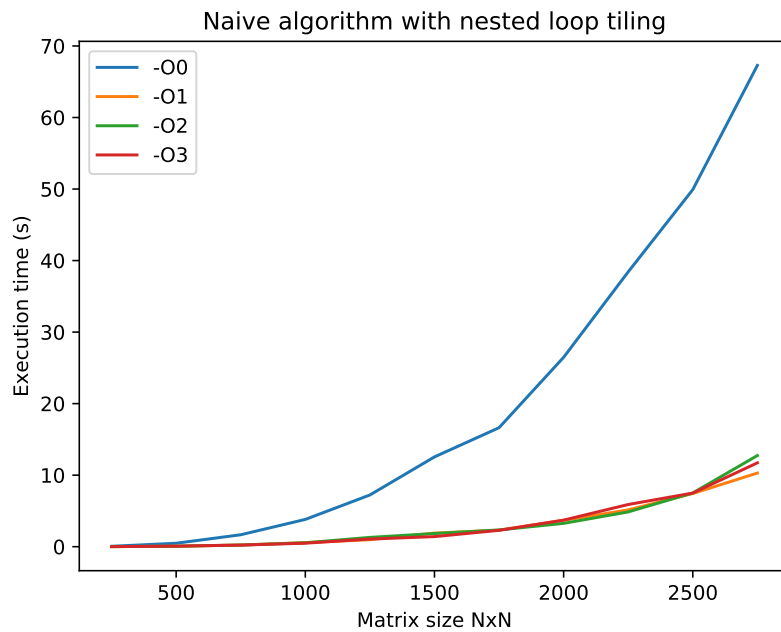


Figure 2: Avec les optimisations

2.3 Parallélisation

```

1 #pragma omp parallel for shared(a, b, res) \\  
2     private(i, j, x, acc) \\  
3     schedule(static)  
4 for(i=0; i<N; i++) {  
5     for(j=0; j<N; j++) {  
6         acc = 0.0;  
7         for(x=0; x<N; x++) {

```

```

8     acc += a[i][x] * b[x][j];
9 }
10 res[i][j] = acc;
11 }
12 }

```

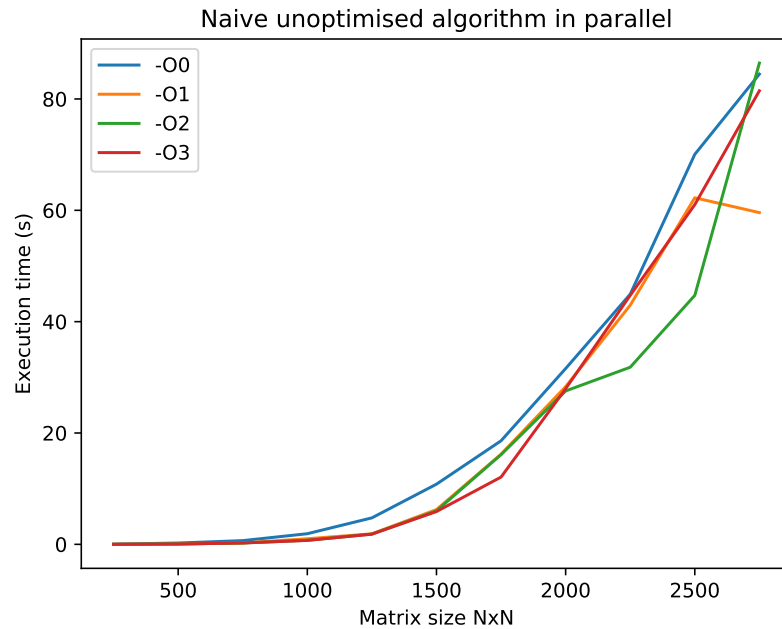


Figure 3: Exécuté en parallèle

2.4 Optimisations et parallélisation

```

1  int ib = 10, kb = 10;
2  #pragma omp parallel for shared(a, b, res, ib, kb) \
3      private(i, ii, j, k, kk, acc00, acc01, acc10, acc11) \
4      schedule(static)
5  for (ii = 0; ii < N; ii += ib) {
6      for (kk = 0; kk < N; kk += kb) {
7          for (j=0; j < N; j += 2) {
8              for(i = ii; i < ii + ib; i += 2 ) {
9                  if (kk == 0)
10                     acc00 = acc01 = acc10 = acc11 = 0;
11                  else {
12                     acc00 = res[i + 0][j + 0];
13                     acc01 = res[i + 0][j + 1];

```

```

14         acc10 = res[i + 1][j + 0];
15         acc11 = res[i + 1][j + 1];
16     }
17     for (k = kk; k < kk + kb; k++) {
18         acc00 += b[k][j + 0] * a[i + 0][k];
19         acc01 += b[k][j + 1] * a[i + 0][k];
20         acc10 += b[k][j + 0] * a[i + 1][k];
21         acc11 += b[k][j + 1] * a[i + 1][k];
22     }
23     res[i + 0][j + 0] = acc00;
24     res[i + 0][j + 1] = acc01;
25     res[i + 1][j + 0] = acc10;
26     res[i + 1][j + 1] = acc11;
27 }
28 }
29 }
30 }

```

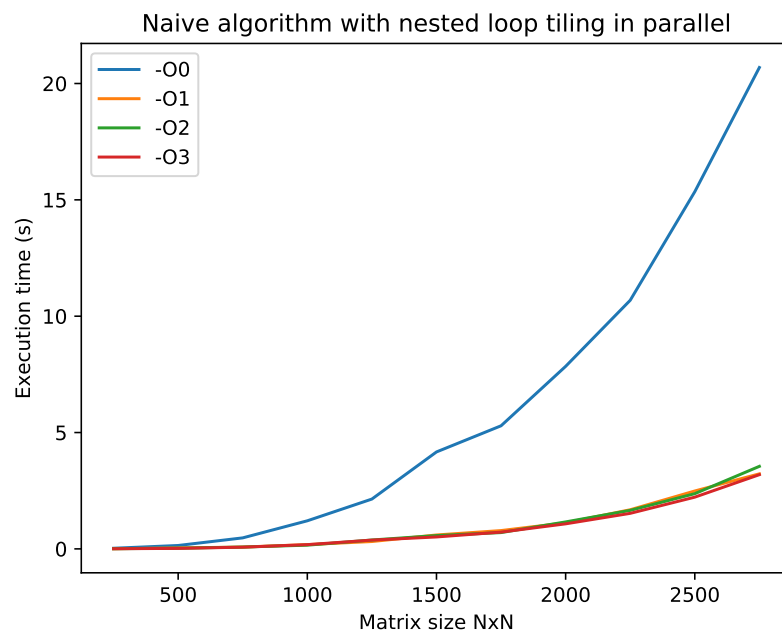


Figure 4: Avec les optimisations et en parallèle

3 Comparaison entre les algorithmes

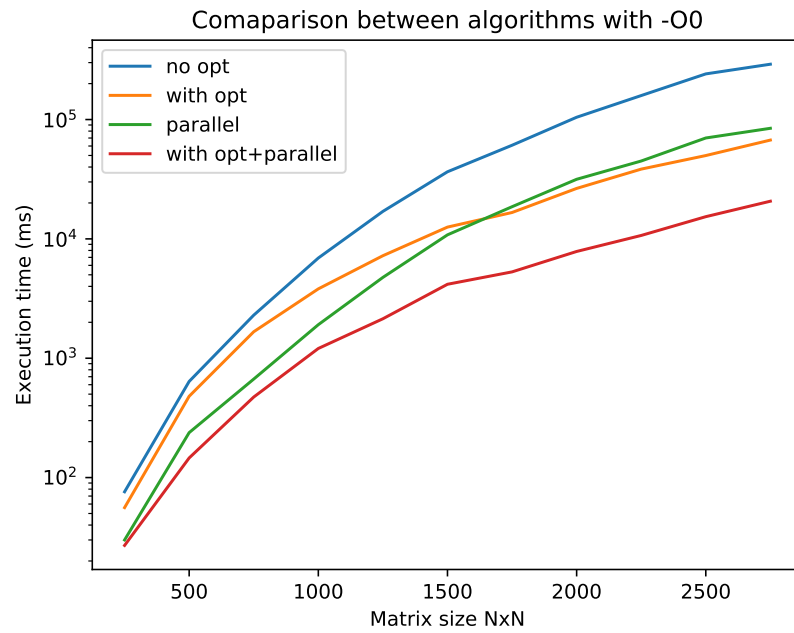


Figure 5: Avec niveau d'optimisation 0 (échelle logarithmique)

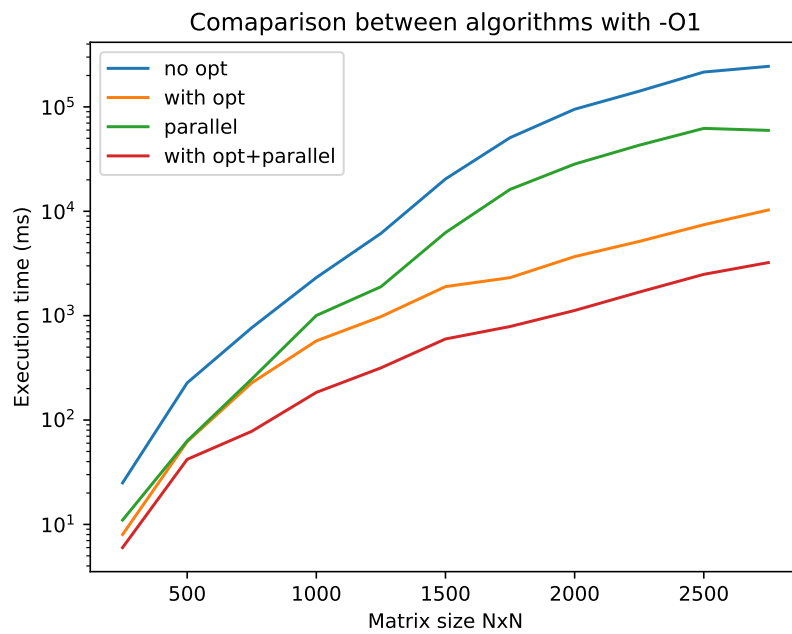


Figure 6: Avec niveau d'optimisation 1 (échelle logarithmique)

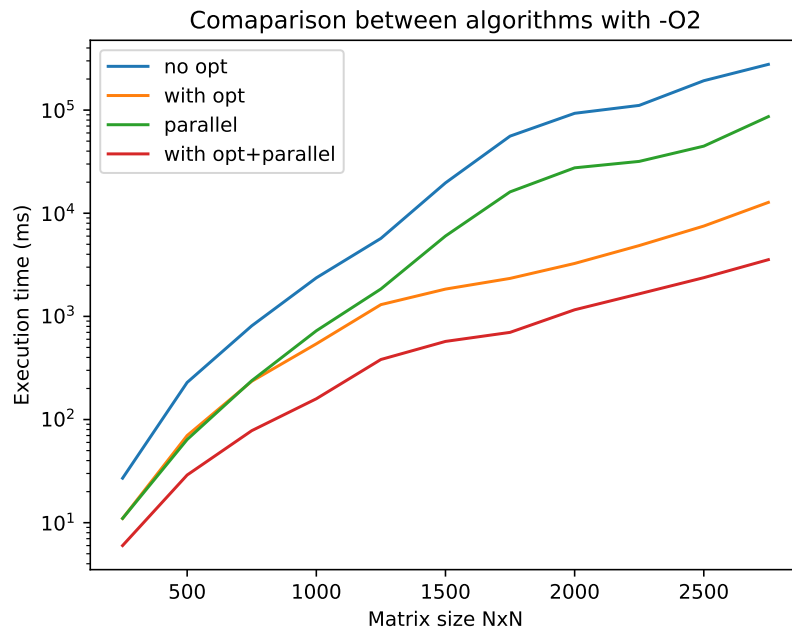


Figure 7: Avec niveau d'optimisation 2 (échelle logarithmique)

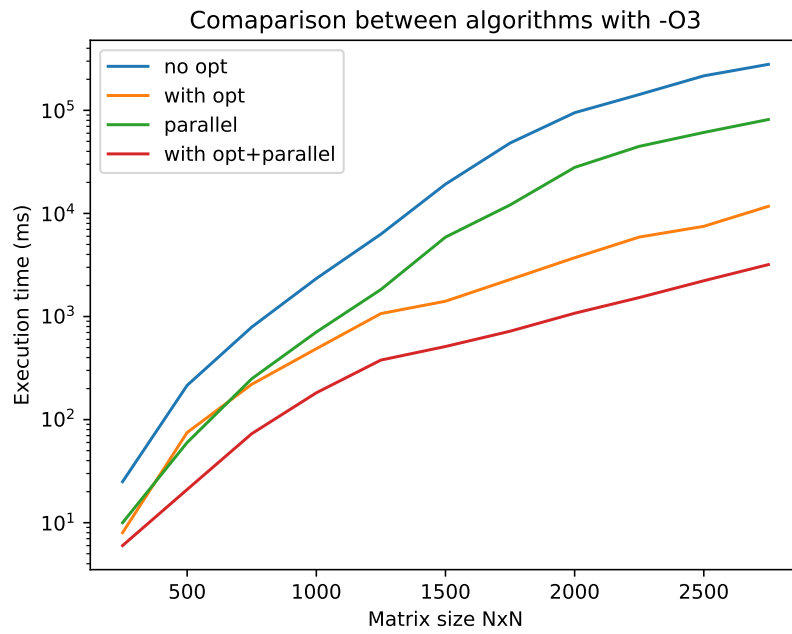


Figure 8: Avec niveau d'optimisation 3 (échelle logarithmique)

4 Observations