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 un implementation d'un bibliothéque des réseaux neurones.

2 Les techniques d'optimisation utilisé

2.1 Algorithme naive

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

```
for(i=0; i<N; i++) {
    for(j=0; j<N; j++) {
        acc = 0.0;
        for(x=0; x<N; x++) {
            acc += a[i][x] * b[x][j];
        }
        res[i][j] = acc;
    }
}
```

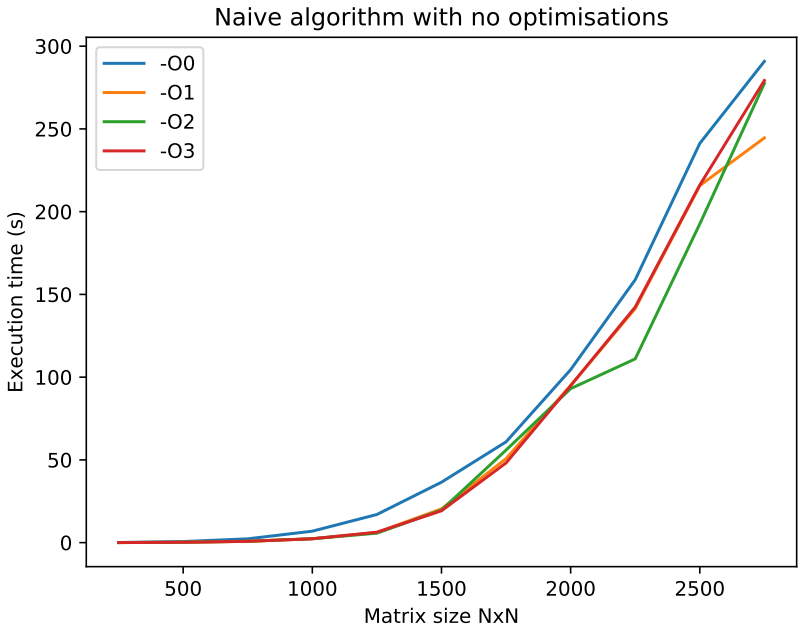

Figure 1: Pas d'optimisations

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

```
int ib = 10, kb = 10;
for (ii = 0; ii < N; ii += ib) {
    for (kk = 0; kk < N; kk += kb) {
        for (j=0; j < N; j += 2) {
            for(i = ii; i < ii + ib; i += 2 ) {
                if (kk == 0)
                acc00 = acc01 = acc10 = acc11 = 0;
                else {
                    acc00 = res[i + 0][j + 0];
                    acc01 = res[i + 0][j + 1];
                    acc10 = res[i + 1][j + 0];
                    acc11 = res[i + 1][j + 1];
                }
                for (k = kk; k < kk + kb; k++) {
                    acc00 += b[k][j + 0] * a[i + 0][k];
                    acc01 += b[k][j + 1] * a[i + 0][k];
```

```
                acc10 += b[k][j + 0] * a[i + 1][k];
                acc11 += b[k][j + 1] * a[i + 1][k];
            }
            res[i + 0][j + 0] = acc00;
            res[i + 0][j + 1] = acc01;
            res[i + 1][j + 0] = acc10;
            res[i + 1][j + 1] = acc11;
        }
    }
  }
}
```

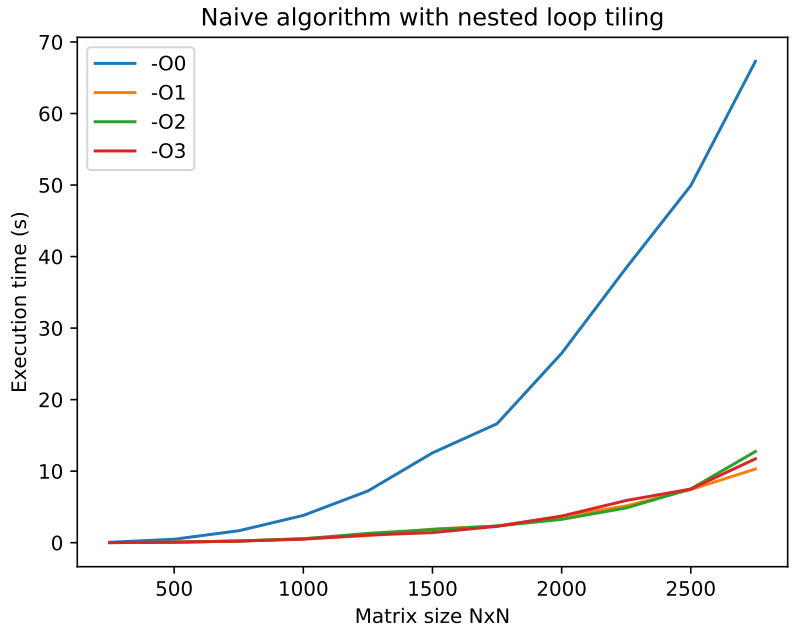

Figure 2: Avec les optimisations

2.3 Parallélisation

```
#pragma omp parallel for shared(a, b, res) \\
                         private(i, j, x, acc) \\
                         schedule(static)
for(i=0; i<N; i++) {
    for(j=0; j<N; j++) {
    acc = 0.0;
    for(x=0; x<N; x++) {
```

```
        acc += a[i][x] * b[x][j];
    }
    res[i][j] = acc;
    }
}
```

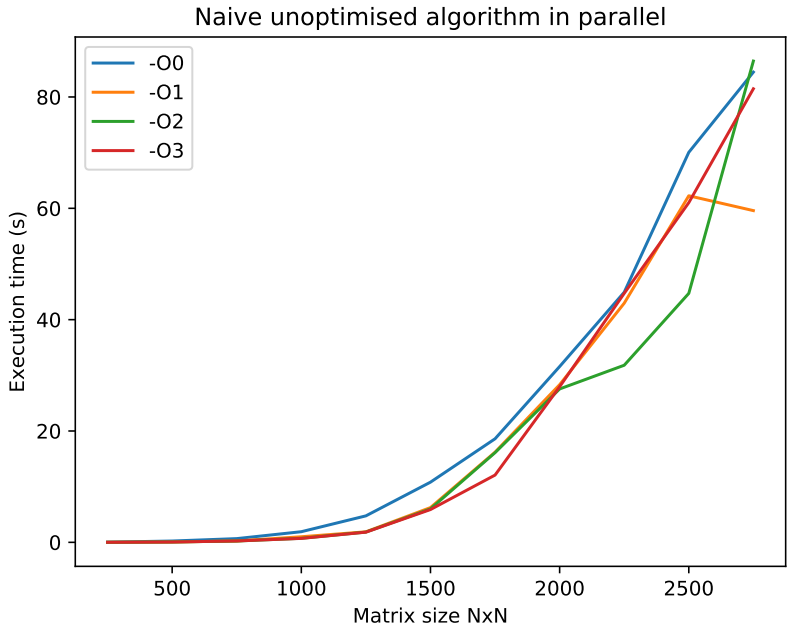

Figure 3: Exécuté en paralléle

2.4 Optimisations et parallélisation

```
int ib = 10, kb = 10;
#pragma omp parallel for shared(a, b, res, ib, kb) \\
            private(i, ii, j, k, kk, acc00, acc01, acc10, acc11) \\
            schedule(static)
for (ii = 0; ii < N; ii += ib) {
    for (kk = 0; kk < N; kk += kb) {
        for (j=0; j < N; j += 2) {
            for(i = ii; i < ii + ib; i += 2 ) {
                if (kk == 0)
                    acc00 = acc01 = acc10 = acc11 = 0;
                else {
                    acc00 = res[i + 0][j + 0];
                    acc01 = res[i + 0][j + 1];
```

```
                    acc10 = res[i + 1][j + 0];
                    acc11 = res[i + 1][j + 1];
                }
                for (k = kk; k < kk + kb; k++) {
                    acc00 += b[k][j + 0] * a[i + 0][k];
                    acc01 += b[k][j + 1] * a[i + 0][k];
                    acc10 += b[k][j + 0] * a[i + 1][k];
                    acc11 += b[k][j + 1] * a[i + 1][k];
                }
                res[i + 0][j + 0] = acc00;
                res[i + 0][j + 1] = acc01;
                res[i + 1][j + 0] = acc10;
                res[i + 1][j + 1] = acc11;
            }
        }
    }
}
```

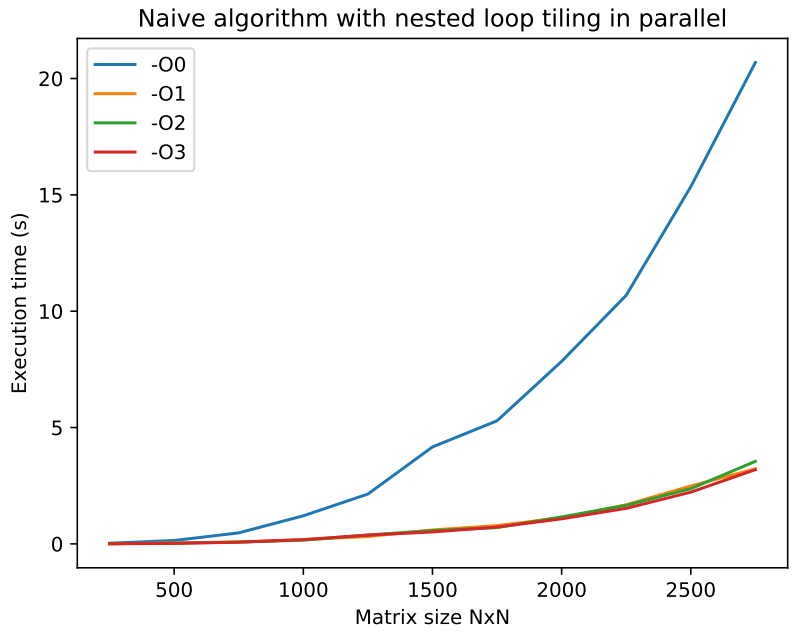

Figure 4: Avec les optimisations et en paralléle

3 Comparaison entre les algorithmes

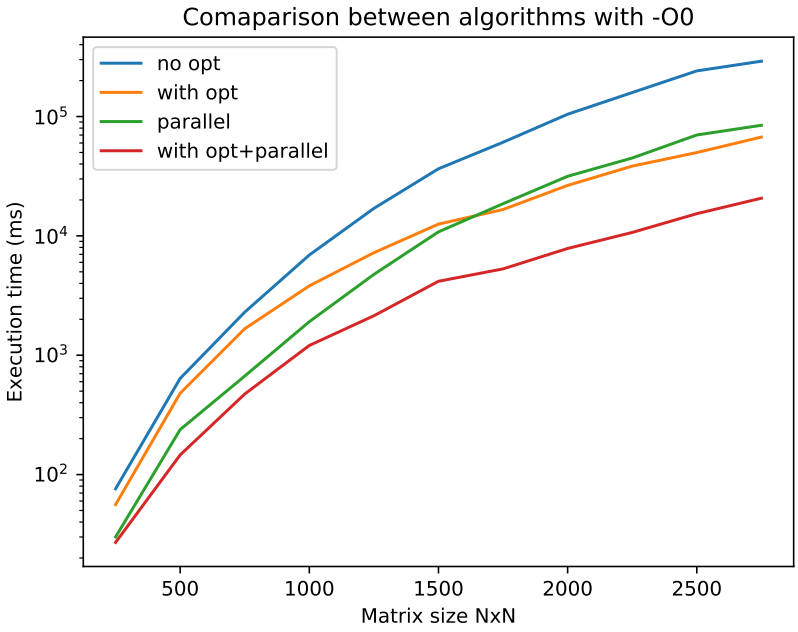

Figure 5: Avec niveau d'optimisation 0 (échelle logarithmique)

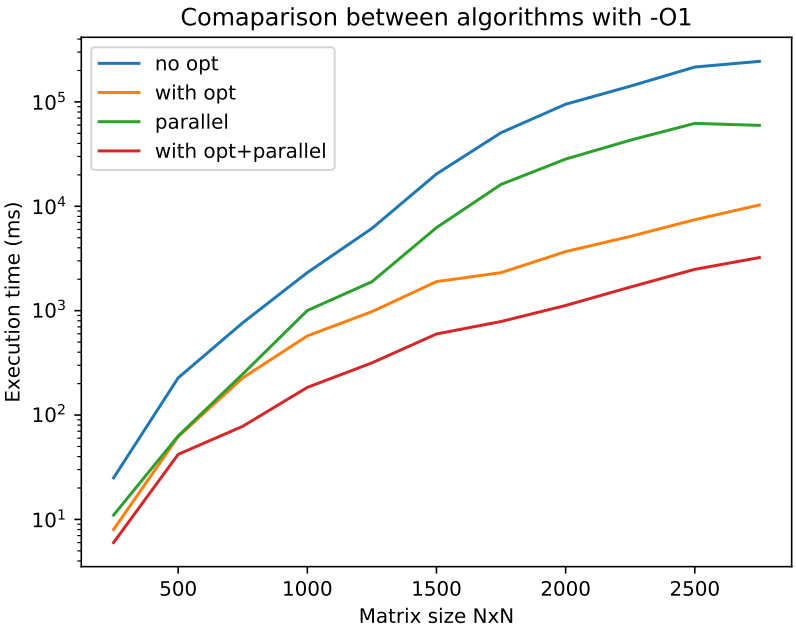

Figure 6: Avec niveau d'optimisation 1 (échelle logarithmique)

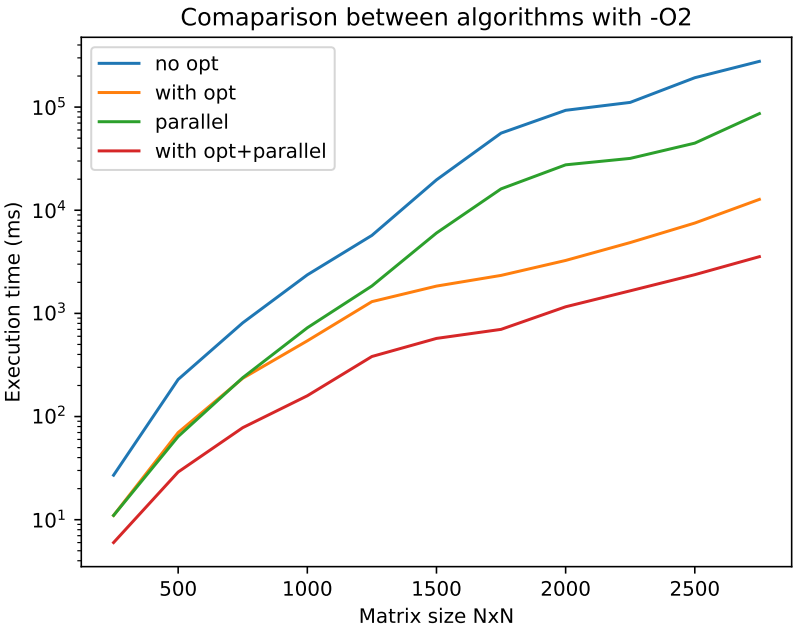

Figure 7: Avec niveau d'optimisation 2 (échelle logarithmique)

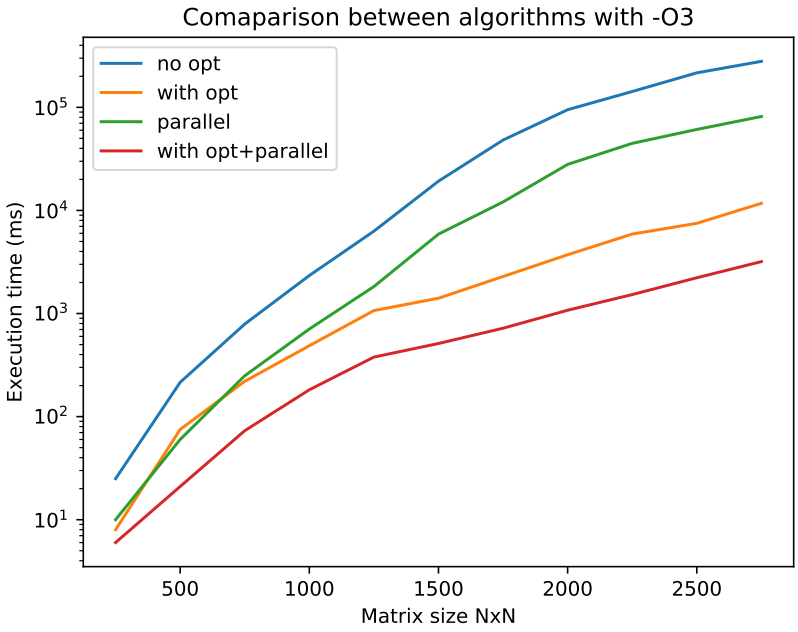

Figure 8: Avec niveau d'optimisation 3 (échelle logarithmique)

4 Observations