Université Mohamed V – Rabat
Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes
Master de recherche CLOUD AND HIGH PERFORMANCE COMPUTING

Mémoire de projet de fin d'études

La descente du gradient stochastique parallélisé avec OpenMP dans un contexte de l'intelligence artificielle

Soutenu le 02/09/2021 par :
Houda Assassi

Encadré par :
Pr. Mostapha Zbakh [ENSIAS]
Pr. Claude Tadonki [Mines ParisTech]

Membre de jury :
Pr. Abderrahmane Ez-zaout [FSR], Président
Pr. Ali Ouacha [FSR], Examinateur

Année universitaire : 2020/2021

Remerciements

En préambule à ce mémoire, j'adresse tout d'abord mes sincères remerciements à mes encadrants le Professeur Mustapha Zbakh de l'ENSIAS de Rabat et le Professeur Claude Tadonki de l'école des mines ParisTech de Paris qui m'ont encadré avec beaucoup de cœur et de patience et à qui j'exprime toute ma reconnaissance de m'avoir donné la chance de bénéficier de leurs connaissances ainsi que leur aide tout au long de ce projet.

Je remercie tant mes professeurs de toute l'équipe pédagogique du master qui ont contribué dans notre formation enrichissante et instructive et spécialement les Professeurs Mohamed Lazaar, Azize Kour, Hanane Bekkali et Mohamed Essaaidi qui m'ont marqué le plus, je remercie encore les membres du jury les Professeurs Abderrahmane Ez-zaout et Ali Ouacha de la faculté des sciences de Rabat qui ont pris le soin d'examiner ce travail et de l'enrichir avec leurs observations.

Je remercie intensément ma petite famille qui a toujours été présente, qui veille et qui prie toujours pour moi.

''Merci Dieu pour la force et le courage que tu m'as donné dans les moments les plus difficiles durant ces années d'étude.''

Résumé

Ce présent document est une synthétisation d'un travail de mémoire de recherche réalisé dans le cadre d'un projet de fin d'étude pour l'obtention du Master Cloud and High Performance Computing (CLOUDHPC).

Sur le plan mathématique, les algorithmes utilisés dans le Machine Learning existent déjà depuis des dizaines d'années, et à l'ère de la révolution digitale ces algorithmes reprennent toute leur importance et tirent profit de la puissance de calcul disponible aujourd'hui, mémoire, processeurs, cartes graphiques etc.

Mon travail consistait tout d'abord à étudier les algorithmes de classifications les plus utilisés dans le domaine du Machine Learning, ensuite, se focaliser sur un de ces algorithmes, l'étudier et voir la possibilité de sa parallélisation. L'algorithme choisi dans ce projet est la descente du gradient stochastique, il a été implémenté et en utilisant le Multithreading en OpenMP.

Mots-clés : algorithmes de classification – descente du gradient – descente du gradient stochastique – Machine Learning – OpenMP

Abstract

This document is a final year thesis for the Cloud and High Performance Computing (CLOUDHPC) master degree.

Mathematically, the algorithms used in Machine Learning have been around for decades, and in the era of the digital revolution, these algorithms are regaining their importance and taking advantage of the computing power available today, memory, processors, graphics cards etc.

My work consisted first in studying the most used classification algorithms in Machine Learning, then, to focus on one of these algorithms, to study it and to see the possibility of its parallelization. The algorithm chosen in this project is the stochastic gradient descent, it has been implemented and tested using multithreading in OpenMP

Keywords : classification algorithms - gradient descent - stochastic gradient descent - Machine Learning - OpenMP

Liste des figures

Liste des abréviations

Abréviation	Désignation
SGD	Stochastic Gradient Descent
IA	Intelligence artificielle
DLL	Dynamic Link Library
MPI	Message Passing Interface
OpenMP	Open Multi-Processing
GHz	Gigahertz
MHz	Megahertz
Go	Gigaoctet
µs	Microseconde

Table des matières

INTRODUCTION GENERALE

Au cours de la dernière décennie, la quantité de données disponibles n'a cessé d'augmenter et étant donné que la bande passante du stockage et du réseau par ordinateur n'a pas pu suivre l'augmentation des données, l'idée de concevoir des algorithmes d'analyse de données capables d'effectuer la plupart des étapes d'une manière distribuée et sans contraintes strictes sur la communication est devenue de plus en plus pertinente.

Le passage des algorithmes d'apprentissage par lots aux algorithmes d'apprentissage en ligne a permis de faire face à l'augmentation de la taille des ensembles de données, puisqu'il a permis de réduire le temps d'exécution des algorithmes , qui est passé d'un comportement cubique ou quadratique à un comportement linéaire en fonction de la taille de l'échantillon. Cependant, lorsque nous disposons de plus d'un seul disque de données, il devient infaisable de traiter toutes les données par descente de gradient stochastique, qui est une méthode d'apprentissage en ligne et séquentiel par nature, du moins si nous voulons obtenir le résultat en quelques heures plutôt qu'en quelques jours.

Le présent travail est réparti comme suit : dans le premier chapitre, on va présenter un état d'art sur les algorithmes de classification, dans le deuxième chapitre on va faire une synthétisation de deux articles scientifiques traitant la parallélisation de l'algorithme du gradient stochastique avec MPI et OpenMP et dans le troisième chapitre on va faire une analyse expérimentale et discuter les résultats obtenus.

CHAPITRE 1 ETAT DE L'ART

Introduction

Dans ce premier chapitre, on va mener une étude comparative de sept algorithmes de classification les pus courants et présenter leur avantages et leur limitations. O va aborder l'algorithme de a descente du gradient puis l'algorithme de la descente du gradient stochastique et ses différentes variantes.

1 Etude descriptive et comparative des algorithmes de classification

1.1 La répartition des algorithmes de classification

Les algorithmes de classification peuvent généralement être classés de la manière suivante :

Les classifieurs linéaires

- Le discriminant linéaire de Ficher ou l'analyse discriminante linéaire.
- La classification naïve bayésienne
- La régression logistique

Les machines à vecteur de support

- Les machines à vecteur de support à moindre carrés

Les classifieurs quadratiques

Estimation Kernel

- La méthode des k plus proches voisins

Les arbres de décision

- Les forêts d'arbres décisionnels

Les réseaux de neurones

La quantification vectorielle d'apprentissage

1.2 Application des algorithmes de classification

- ✓ Classification des pourriels
- ✓ Prédiction de la volonté de remboursement des prêts des clients des banques
- ✓ Identification des cellules tumorales du cancer
- ✓ Analyse des sentiments
- ✓ Classification des médicaments
- ✓ Détection des points clés du visage
- ✓ Détection des piétons dans une conduite automobile

1.3 Les sept algorithmes de classification les plus courants

1. La régression logistique
   La régression logistique est un algorithme d'apprentissage automatique pour la classification. Dans cet algorithme, les probabilités décrivant les résultats possibles d'un seul essai sont modélisées à l'aide d'une fonction logistique.

2. La classification naïve bayésienne
   L'algorithme de Bayes est basé sur le théorème de Bayes avec l'hypothèse de l'indépendance entre chaque paire de caractéristiques. Ils fonctionnent bien sur des applications du mon réel telles que la classification de documents et le filtrage de spam.

3. L'algorithme du gradient stochastique
   La descente du gradient stochastique est une approche simple et très efficace pour ajuster des modèles linéaires. Elle est particulièrement utile que lorsque le nombre d'échantillons est très important. Elle prend en charge différentes fonctions de perte et pénalités pour classifications.

4. La méthode des k plus proches voisins

La classification basée sur les voisins est un type d'apprentissage paresseux car elle ne tente pas de construire un modèle interne général, mais stocke simplement les instances de données d'apprentissage. La classification est calculée à partir d'un simple vote majoritaire des k voisins les plus proches de chaque point.

5. L'arbre de décision
   Etant donné une donnée d'attributs et ses classes, un arbre de décision produit une séquence de règles qui peuvent être utilisés pour classer les données.

6. Les forêts d'arbres décisionnels
   Méta-estimateur qui ajuste un certain nombre d'arbres de décision sur divers sous-échantillons d'ensemble de données et utilise la moyenne pour améliorer la précision prédictive du modèle et contrôler le 'Over-fitting'. La taille du sous-échantillon est toujours la même que celle de l'échantillon d'entrée original. Mais les échantillons sont tirés avec remplacement.

7. Machine à vecteur de support
   La machine à vecteur de support est une représentation des données d'entrainement sous forme de points dans un espace séparé en catégories par un écart clair aussi large que possible. Les nouveaux exemples sont ensuite mis en correspondance dans ce même espace et leur appartenance à une catégorie est prédite en fonction du coté de l'écart qu'ils occupent.

1.4 Définitions et terminologie

La classification est une technique qui permet de classer les données dans un nombre donné de classes, elle peut être effectuée sur des données structurées ou non structurées.
L'objectif principal d'un problème de classification est d'identifier la catégorie/classe à laquelle une nouvelle donnée appartient.
Classifieur (Classifier) : un algorithme qui fait correspondre les données d'entrée à une catégorie spécifique.

Modèle de classification (Classification Model) : tente de tirer une conclusion à partir des valeurs d'entrée données pour l'entrainement. Il prédit les étiquettes/catégories de classe pour les nouvelles données.

Caractéristique (Feature) : une propriété individuelle mesurable d'un phénomène observé.

La classification binaire (Binary Classification) : une tâche de classification avec deux résultats possibles. Par exemple : classification par sexe (Homme /Femme)

La classification multi-classes (Multi-class Classification) : une classification avec plus de deux classes. Dans la classification multi-classes, chaque échantillon est assigné à une et une seule étiquette cible. Par exemple : un animal peut être un chat ou chien, mais pas les deux en même temps.

La classification multi-labels (Multi-label Classification) : une tâche de classification où chaque échantillon est associé à un ensemble d'étiquettes (plus d'une classe). Par exemple : un article peut porter à la fois sur le sport, une personne et un lieu.

1.5 Les étapes de construction d'un modèle de classification

1. **Initialiser** le classifieur à utiliser.
2. **Entraîner le classifieur :** tous les classifieurs de scikit-learn utilisent une méthode fit(x,y) pour ajuster le modèle pour les données d'entrainement x et l'étiquette d'entraînement y données.
3. **Prévoir la cible :** étant donnée une observation non étiqueté x, predict(x) retourne l'étiquette prédite y.
4. **Évaluer** le modèle classifieur

1.6 Etude comparative

1.6.1 Description et analyse exploratoire des données

Les données représentent des salaires, elles ont été extraites depuis la base de données du bureau du recensement des états unis (https://data.census.gov/cedsci/).

Le fichier de données d'extension csv et de4,23 Mo contient 48 842 instances (lignes) et 11 attributs (colonnes).

On a deux classes : '>50 000' et '<50 000'

On trouve dans l'ensemble de données 7 variables explicatives :

1. **age_bin** avec cinq valeurs possibles (a. 0-25 , b. 26-30 & 71-100 , c. 31-35 & 61-70 , d. 36-40 & 56-60 , e. 40,55)
2. **capital_gl_bin**avec trois valeurs possibles (a. =0 , b. <0 , c. >0)
3. **education_bin**avec cinq valeurs possibles (a. Low , b. Mid , c. Bachelors , d. Masters , e. High)
4. **hours_per_week_bin**avec cinq valeurs possibles (a. 0-30 , b. 31-40 , c. 71-100 , d. 41-50 & 61-70 , e. 51-60)
5. **msr_bin**avec trois valeurs possibles (a. Low , b. Mid , c. High)
6. **occupation_bin**avec cinqvaleurspossibles(a. Low , b. Mid-Low , c. Mid-Mid , d. Mid-High , e. High)
7. **race_sex_bin**avec trois valeurs possibles(a. Low , b. Mid , c. High)

1.6.2 La matrice de comparaison des algorithmes de classification

La précision est un rapport entre le nombre d'observations correctement prédites et le nombre total d'observations. La précision est la mesure de performance la plus intuitive.

Le score F1 est la moyenne pondérée de la précision et le rappel utilisée dans tous les types des algorithmes de classification. Il est généralement plus utile que la précision. Surtout si la distribution de classe est inégale.

L'algorithme de classification	La précision	Le score F1
La régression logistique	84,60%	0,6337
La classification naïve bayésienne	80,11%	0,6005
L'algorithme du gradient stochastique	82,20%	0,5780
La méthode des k plus proches voisins	83,56%	0,5924

L'arbre de décision	84,23%	0,6308
Les forêts d'arbres décisionnels	84,33%	0,6275
Machine à vecteur de support	84,09%	0,6145

Figure1.1 : La matrice de comparaison des 7 algorithmes de classification

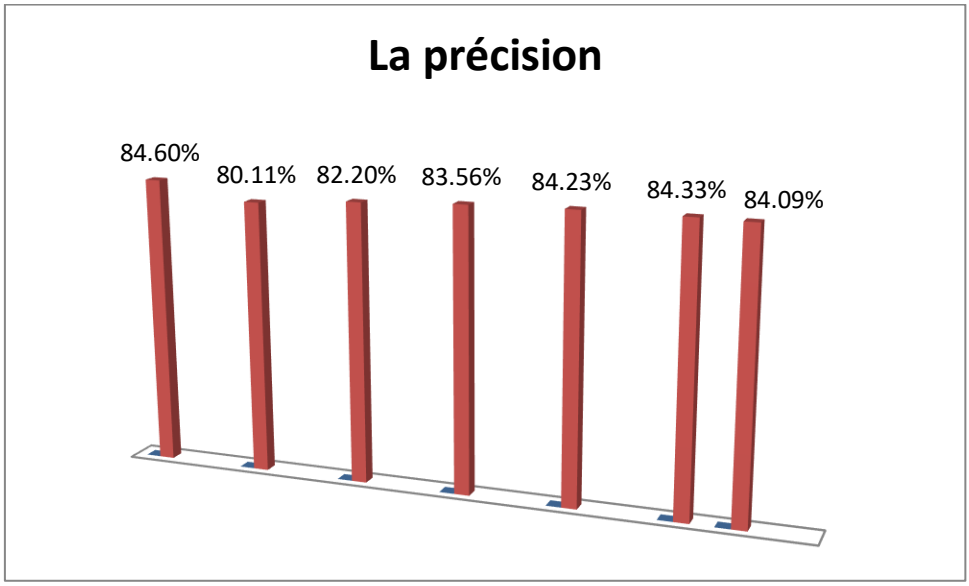

Figure1.2 : Graphique représentant la précision

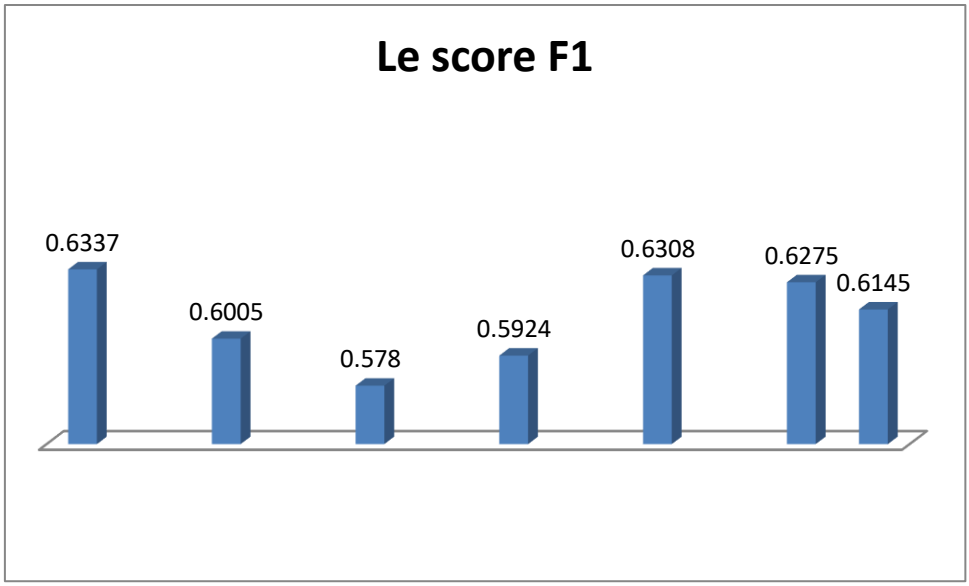

Figure1.3 : Graphique représentant le score F1

Remarque : la représentation graphique de gauche à droite suit l'ordre des algorithmes de classification dressés dans le tableau.

1.6.3 Avantages et limitations des algorithmes de classification

- **La régression logistique**

  Avantages :

  - Conçu pour la classification
  - Plus utile pour comprendre l'influence de plusieurs variables indépendantes sur une seule variable de résultat

  Limitations :

  - Ne fonctionne que lorsque la variable prédite est binaire
  - Suppose que tous les prédicteurs sont indépendants les uns des autres
  - Suppose que les données sont exemptes des valeurs manquantes

- **La classification naïve bayésienne**

  Avantages :

  - Nécessite une petite quantité de données d'entrainement pour estimer les paramètres nécessaires

- Extrêmement rapides par rapport aux méthodes plus sophistiqués

Limitations :

- Connu pour être un mauvais estimateur

➢ **L'algorithme du gradient stochastique**

Avantages :

- Efficacité et facilité d'implémentation

Limitations :

- Nécessite un certain nombre d'hyperparamètres
- Sensible à l'échelle des caractéristiques

➢ **La méthode des k plus proches voisins**

Avantages :

- Simple à implémenter
- Robuste aux données d'entraînement bruyantes
- Efficace si les données d'entrainement sont nombreuses

Limitations :

- Il faut déterminer la valeur de k
- Le coût de calcul est élevé car il faut calculer la distance de chaque instance à tous les échantillons d'entrainement

➢ **L'arbre de décision**

Avantages :

- Simple à comprendre et à visualiser
- Nécessite un peu de préparation de données
- Peut traiter des données numériques et catégorielles

Limitations :

- Peuvent créer des arbres complexes qui ne se généralisent pas bien
- Peuvent être instables car de petites variations dans les données peuvent entraîner la génération d'un arbre complètement différent

- **Les forêts d'arbres décisionnels**

  Avantages :

  - Réduction du 'Over-fitting'
  - Plus précis que les arbres de décision dans la plupart des cas

  Limitations :

  - Prévision lente en temps réel
  - Difficile à implémenter
  - Algorithme complexe

- **Machine à vecteur de support**

  Avantages :

  - Efficace dans les espaces de grande dimension
  - Utilise un sous –ensemble de points d'entraînement dans la fonction de décision, ce qui lui permet d'utiliser efficacement la mémoire.

  Limitations :

  - Ne fournit pas directement des estimations de probabilités, celle-ci est calculée à l'aide d'une Cross-validation à cinq reprises ce qui est coûteux.

2 L'algorithme de la descente du gradient

2.1 Contexte de l'intelligence artificielle

La descente du gradient est parmi les algorithmes les plus importants dans le Machine Learning et le Deep Learning, c'est un algorithme d'optimisation **utilisé dans l'entraînement des modèles** de la régression linéaire, la régression logistique ou encore les réseaux de neurones.

Grâce à l'entraînement, de nombreux algorithmes obtiennent de bonnes performances, et une grande majorité des entraînements repose sur l'optimisation d'une fonction de perte, plus la valeur de cette dernière est petite, meilleurs sont les résultats de l'algorithme.

Ainsi, l'algorithme de la descente du gradient est parmi les meilleurs outils pour des raisons de complexité qui permettent de trouver le minimum

d’une fonction. L’algorithme de la descente du gradient est particulièrement utile lorsque le nombre d'échantillons est important.

2.2 Algorithme du gradient: démarche itérative pour la minimisation d’une fonction

L’optimisation d’une fonction est un problème très familier dans de nombreux domaines, il joue un rôle important en recherche opérationnelle, dans les mathématiques appliquées, en analyse et en analyse numérique, en statistique pour l’estimation du maximum de vraisemblance d’une distribution, pour la recherche de stratégies dans le cadre de la théorie des jeux, ou encore en théorie du contrôle et de la commande.

2.3 Exemple d’optimisation d’une fonction différentiable et convexe

$f(x)=x^2 - x +1$ une fonction à minimiser par rapport à x

- f() est la fonction à minimiser
- x est le paramètre, on cherche la valeur de x qui minimise f()

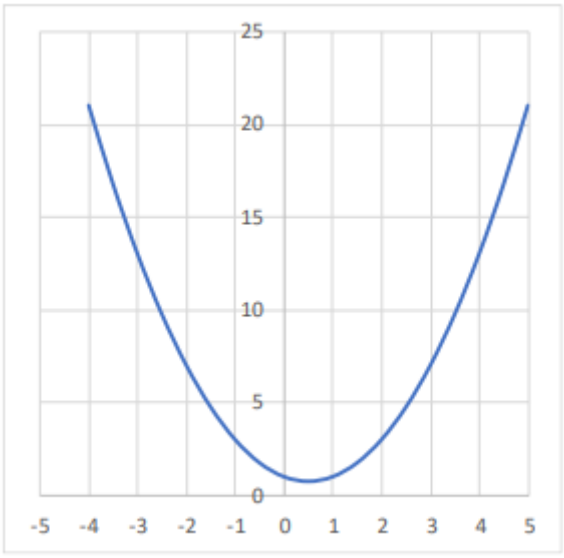

Figure1.4: la courbe représentative de la fonction $f(x)=x^2 - x +1$

La solution analytique passe par f’(x) = 0 en s’assurant que f’’(x) = 0
f’(x)= 2x - 1 = 0 ⇒ x* = ½

2.4 L’algorithme de la descente du gradient

La résolution analytique n'est parfois pas possible à cause du nombre élevé de paramètres, on procède par l’approximation avec une approche itérative, l’algorithme du gradient est un algorithme itératif.

L'algorithme est décrit par 3 étapes essentielles:

Première étape: **Initialisation au hasard avec x0**

- Il est quasiment impossible de suggérer des valeurs intelligentes.

Deuxième étape: **Répéter** $\boldsymbol{xt + 1 = xt - \eta \times \nabla f(xt)}$

- Le gradient est une généralisation multidimensionnelle de la dérivée si on a un seul paramètre au point xt. Il indique la direction et l'importance de la pente au voisinage de xt.
- - parce qu'on cherche à minimiser f(), sinon on peut mettre +.
- □ est un paramètre qui permet de moduler la correction, si □ est trop faible on a une lenteur de convergence et si □ trop est trop élevé on a une oscillation.

Troisième étape: **jusqu'à convergence**

- Le nombre d'itérations fixé ou différence entre valeurs successives xt ou ||∇(□□)|| très petit.

2.5 Algorithme du gradient – Exemple

$f(x)=x^2 - x +1$

Il n'y a qu'un seul paramètre, la dérivée partielle est égale à la dérivée. Donc

$\nabla f(x) = \partial f(x)/ \partial x = f'(x)= 2x - 1$

On prend $\eta = 0,3$ et x0 = 5 avec f(x0) = 21

Avec la formule $xt+1 = xt - \eta \times \nabla f(xt)$ on obtient le tableau suivant:

xt+1	f'(xt)	f(xt+1)
2,3000	9,0000	21,0000
1,2200	3,6000	3,9900
0,7880	1,4400	1,2684
0,6152	0,5760	0,8329
0,5461	0,2304	0,7633
0,5184	0,0922	0,7521
0,5074	0,0369	0,7503
0,5029	0,0147	0,7501
0,5012	0,0059	0,7500
0,5005	0,0024	0,7500
0,5002	0,0009	0,7500
0,5001	0,0004	0,7500
0,5000	0,0002	0,7500

Figure1.5: Minimisation de la fonction $f(x)=x^2 - x +1$ avec l'algorithme du gradient, x0 = 5

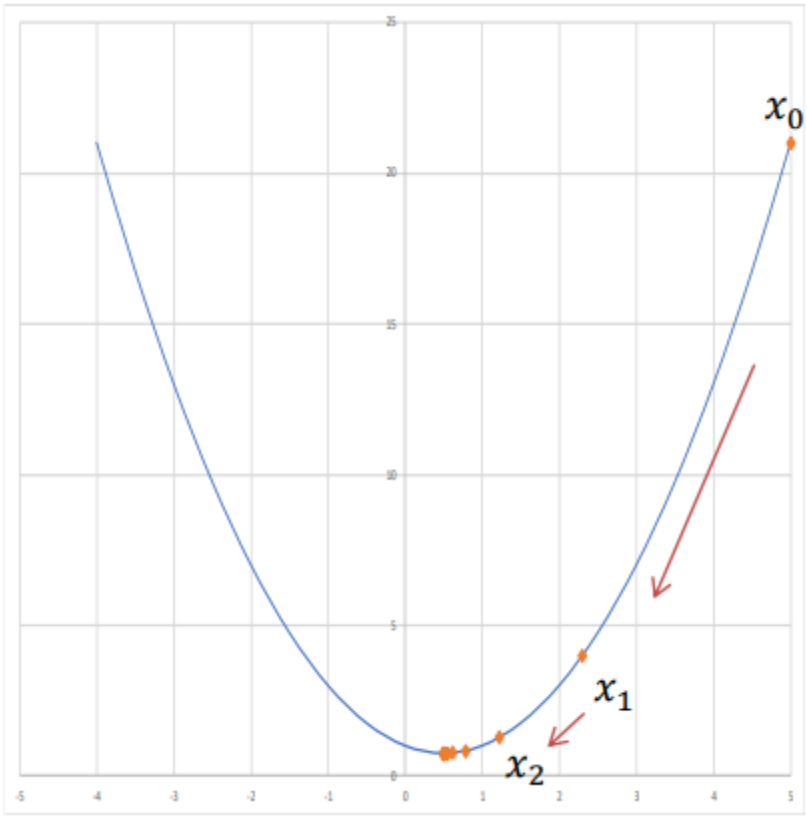

Figure1.6: Minimisation de la fonction $f(x)=x^2 - x +1$ avec l'algorithme du gradient, x0 = 5

Comme on peut aussi partir de l'autre côté en prenant cette fois-ci x0 = -4 avec f(x0) = 21 et gardant η = 0,3.

xt+1	f'(xt)	f(xt+1)
-1,3000	-9,0000	3,9900
-0,2200	-3,6000	1,2684
0,2120	-1,4400	0,8329
0,3848	-0,5760	0,7633
0,4539	-0,2304	0,7521
0,4816	-0,0922	0,7503
0,4926	-0,0369	0,7501
0,4971	-0,0147	0,7500
0,4988	-0,0059	0,7500
0,4995	-0,0024	0,7500

0,4998	-0,0009	0,7500
0,4999	-0,0004	0,7500
0,5000	-0,0002	0,7500

Figure1.7 : Minimisation de la fonction $f(x)=x^2 - x +1$ avec l'algorithme du gradient, x0 = -4

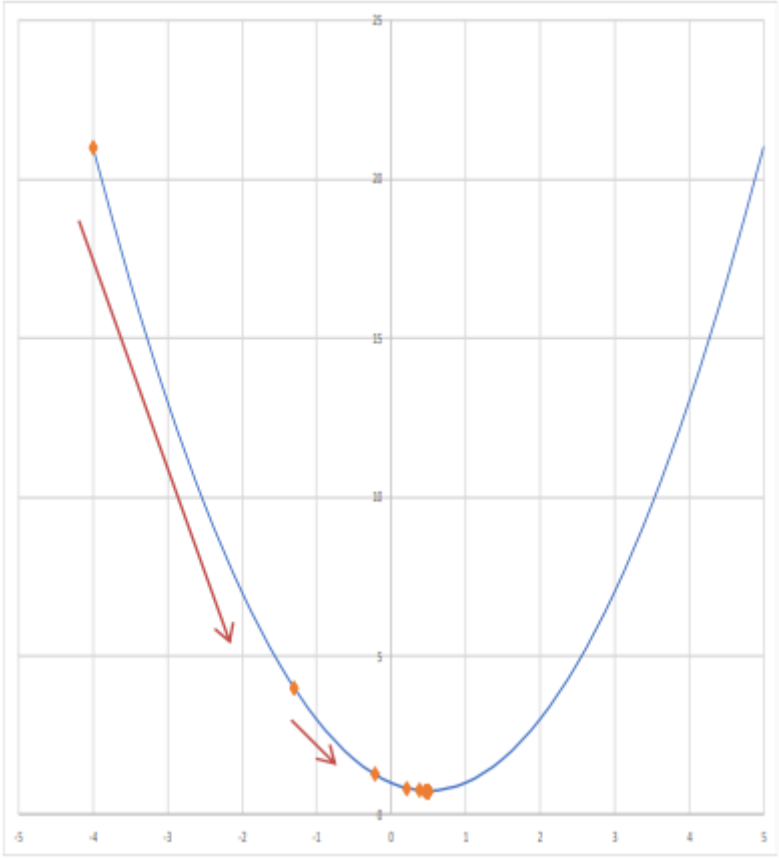

Figure1.8: Minimisation de la fonction $f(x)=x^2 - x +1$ avec l'algorithme du gradient, x0 = -4

3 L'algorithme de la descente du gradient stochastique

3.1 Problématique

La descente de gradient est un algorithme itératif qui commence à partir d'un point aléatoire sur une fonction et voyage dans sa pente en quelques étapes jusqu'à ce qu'il atteigne le point le plus bas de cette fonction.

Supposant que nous avons 10 000 points de données et 10 caractéristiques. La somme des résiduels carrés comprend autant de termes qu'il existe des points de données, donc 10000 termes dans notre cas. Nous devons calculer la dérivée de cette fonction en ce qui concerne chacune des caractéristiques,

donc en effet, nous allons faire 10000 * 10 = 100 000 calculs par itération. Il est courant de prendre 1000 itérations, en effet, nous avons 100 000 * 1000 = 100000000 calculs pour compléter l'algorithme. C'est à peu près une surcharge et donc une descente gradient est lente sur des données énormes. D'où vient l'itérer de l'utilisation de la descente de gradient stochastique. La descente de gradient stochastique SGD choisit au hasard un point de données de l'ensemble des données définies à chaque itération afin de réduire énormément les calculs.

La descente de gradient stochastique est un algorithme d'optimisation souvent utilisé dans les applications d'apprentissage de la machine afin de trouver les paramètres de modèle correspondant au meilleur ajustement entre les sorties prédites et réelles. C'est une technique inexacte mais puissante.

La Vanilla SGD met à jour le poids actuel en utilisant le gradient en cours ∂L / ∂W multiplié par un facteur appelé taux d'apprentissage, α.$\boldsymbol{\omega_{t+1} = \omega_t - \alpha * \frac{\partial L}{\partial \omega_t}}$

3.2 Les algorithmes d'optimisation de la descente du gradient stochastique

Les variations de cette équation sont communément appelées optimisateurs de descente du gradient stochastique. Il de y a trois manières principales comment elles diffèrent:

1 - Adapter le "composant gradient" (∂l / ∂W)

Au lieu d'utiliser un seul gradient comme dans une descente de gradient de vanille stochastique pour mettre à jour le poids, on prend un agrégat de plusieurs gradients.

2 - Adapter le "composant de taux d'apprentissage" (α)

Au lieu de garder un taux d'apprentissage constant, on adapte le taux d'apprentissage en fonction de la magnitude du gradient.

4- Adapter à la fois le composant gradient et la composante de taux d'apprentissage.

Optimiseur	Année de création	Taux d'apprentissage	Gradient
Momentum	1964		•
AdaGrad	2011	•	
RMSprop	2012	•	
AdaDelta	2012	•	
Nesterov	2013		•
Adam	2014	•	•
AdaMax	2015	•	•
Nadam	2015	•	•
AMSGrad	2018	•	•

Figure1.9 : Les algorithmes d'optimisation de la descente du gradient stochastique

Le tableau suivant représente une vue d'ensemble.

Optimiseur	Équation	Paramètre
SGD lassique	$\omega_{t+1} = \omega_t - \alpha * \frac{\partial \mathrm{L}}{\partial \omega_t}$	α : $taux\ d'apprentissag$
Momentu	$\omega_{t+1} = \omega_t - \alpha * m_t$ $m_t = \beta * m_{t-1} + (1-\beta) * \frac{\partial \mathrm{L}}{\partial \omega_t}$	m_t: Momentum initialiser a 0 .

m		$\beta: valeurcommun = 0.9$
AdaGrad	$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\nu_t + \epsilon}} * \frac{\partial \mathrm{L}}{\partial \omega_\mathrm{t}}$ $\nu_t = \nu_{t-1} + \left[\frac{\partial \mathrm{L}}{\partial \omega_\mathrm{t}}\right]^2$	ν_t:taux d'apprentissage par racine carrée initialisée à 0. $\varepsilon = 10^{-7}$ - $\alpha = 0.01$
RMSprop	$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\nu_t + \epsilon}} * \frac{\partial \mathrm{L}}{\partial \omega_\mathrm{t}}$ $\nu_t = \beta * \nu_{t-1} + (1-\beta) * \left[\frac{\partial \mathrm{L}}{\partial \omega_\mathrm{t}}\right]^2$	ν_t:taux d'apprentissage par racine carrée initialisée à 0. $\varepsilon = 10^{-6}$ - $\alpha = 0.001$ $\beta: valeurcommun = 0.9$
AdaDelta	$\omega_{t+1} = \omega_t - \frac{\sqrt{\mathcal{D}_t + \epsilon}}{\sqrt{\nu_t + \epsilon}} * \frac{\partial \mathrm{L}}{\partial \omega_\mathrm{t}}$ $\mathcal{D}_t = \beta * \mathcal{D}_{t-1} + (1-\beta) * [\Delta\omega_t]^2$ $\nu_t = \beta * \nu_{t-1} + (1-\beta) * \left[\frac{\partial \mathrm{L}}{\partial \omega_\mathrm{t}}\right]^2$	$\beta = 0.95 - \varepsilon = 10^{-6}$ $\Delta\omega_t = \omega_t - \omega_{t-1}$ $\mathcal{D}_t$:*Delta Adaptive*
Nesterov	$\omega_{t+1} = \omega_t - \alpha * m_t$ $m_t = \beta * m_{t-1} + (1-\beta) * \frac{\partial \mathrm{L}}{\partial \omega^*}$	$\omega^* = \omega_t - \alpha * m_{t-1}$ $\beta = 0.9$
Adam	$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\widehat{\nu_t}} + \epsilon} * \widehat{m}$ $\widehat{m} = \frac{m_t}{1-\beta_1^t}\widehat{\nu_t}$ $= \frac{\nu_t}{1-\beta_2^t}\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \omega} \\ \nu_t = \beta_2 * \nu_{t-1} + (1-\beta_2) * \left[\frac{\partial}{\partial \omega}\right. \end{cases}$	$\alpha = 0.001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\varepsilon = 10^{-8}$
AdaMax	$\omega_{t+1} = \omega_t - \frac{\alpha}{\nu_t} * \widehat{m}$	$\alpha = 0.001$

	$\widehat{m} = \frac{m_t}{1-\beta_1^t}$ $\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \omega} \\ v_t = \max(\beta_2 v_{t-1}, \left\|\frac{\partial L}{\partial \omega}\right\|) \end{cases}$	$\beta_1 = 0.9$ $\beta_2 = 0.999$
Nadam	$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\widehat{v_t}} + \epsilon} * \beta_1 \widehat{m} + \frac{1-\beta_1}{1-\beta_1^t}\frac{\partial L}{\partial \omega_t}$ $\widehat{m} = \frac{m_t}{1-\beta_1^t} \widehat{v_t} = \frac{v_t}{1-\beta_2^t}$ $\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \omega} \\ v_t = \beta_2 * v_{t-1} + (1-\beta_2) * \left[\frac{\partial L}{\partial \omega_t}\right]^2 \end{cases}$	$\alpha = 0.002$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\varepsilon = 10^{-7}$
AMSGrad	$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\widehat{v_t}} + \epsilon} * m_t$ $\widehat{v_t} = \max(\widehat{v_{t-1}}, v_t)$ $\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \omega} \\ v_t = \beta_2 * v_{t-1} + (1-\beta_2) * \left[\frac{\partial L}{\partial \omega_t}\right]^2 \end{cases}$	$\alpha = 0.001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\varepsilon = 10^{-7}$

Figure1.10 : Vue d'ensemble des algorithmes d'optimisation de la descente du gradient stochastique

4 L'algorithme de la descente du gradient parallélisé dans le contexte de l'IA

4.1 La descente du gradient stochastique et l'IA

La descente du gradient stochastique est un algorithme qui minimise une fonction objectif, le Machine Learning et le Deep Learning sont parmi plusieurs domaines qui se servent du principe de la minimisation d'une

fonction objectif dans l'entraînement des modèles. La grande majorité des entraînements reposent sur la minimisation d'une fonction de perte.

La descente du gradient stochastique est utilisé dans l'entraînement de nombreuses familles de modèles du Machine Learning, notamment les machines à vecteurs support, la régression logistique et les modèles graphiques. Et utilisé aussi dans l'entraînement des réseaux de neurones profond.

4.2 Pourquoi paralléliser ?

Avec le développement rapide du Machine Learning, du Deep Learning et du big data, il est devenu courant d'utiliser des données d'entraînements massives pour améliorer la qualité de la précision de la classification. La taille et la complexité du modèle, combinées à la taille de l'ensemble de données d'apprentissage, rendent le processus d'apprentissage très coûteux en termes de calcul et de temps. La descente de gradient stochastique SGD n'est plus le choix le plus approprié en raison de son comportement naturel d'optimisation séquentielle de la fonction de perte, cela a conduit au développement d'algorithmes (SGD) parallèles, tel que le SGD synchrone.

Conclusion

Ce chapitre constitue le point de départ du projet dans la mesure où il exposait le contexte général en explicitant la problématique et en éclaircissant l'axe de travail à suivre.

CHAPITRE 2 ANALYSE DES ARTICLES SUR LA PARALLELISATION DU SGD AVEC MPI et OpenMP

Introduction

Dans ce deuxième chapitre, on va analyser et synthétiser deux articles scientifiques, le premier propose un algorithme parallèle de descente de gradient stochastique multi-classes pour classer en milliers de classes des millions d'images avec des signatures de haute dimension .Et le deuxième article propose aussi une parallélisation de la méthode de la descente du gradient pour optimiser l'analyse de régression.

2.1 ARTICLE1: Parallel multiclass stochastic gradient descent algorithms for classifying million images with very-high-dimensional signatures into thousands classes

Cet article développe un nouvel algorithme de la descente du gradient stochastique pour les machines à vecteur de support nommé SVM-SGD multi classe pour la classification de grands ensembles de données d'images en plusieurs classes. Cet article propose donc un algorithme d'entraînement équilibré pour l'apprentissage de classificateurs SVM-SGD binaires (voir figure

11), et un processus d'entraînement parallèle des classificateurs avec plusieurs ordinateurs multi-cœurs/grilles (voir figure12).

L'article vise à accélérer les tâches d'apprentissage de SVM-SGD multi-classes avec plusieurs ordinateurs multiprocesseurs. L'idée est d'apprendre k classificateurs binaires en parallèle.

La programmation parallèle dans cet article est basée sur deux grands modèles, MPI et OpenMP, il présente une approche hybride qui combine les avantages des modèles OpenMP et MPI.

input :
training data of the positive class D_+
training data of the negative class D_-
positive constant $\lambda > 0$
number of epochs T

output:
SVM-SGD model w

init $w_1 = 0$
for $t \leftarrow 1$ **to** T **do**
  creating the reduced dataset D' from the full set of positive class D_+ and sampling without replacement D'_- from dataset D_- (with $|D'_-| = \sqrt{|D_-| \times |D_+|}$)
  setting $\eta_t = \frac{1}{\lambda t}$
  for $i \leftarrow 1$ **to** $|D_+|$ **do**
    randomly pick a datapoint $[x_i, y_i]$ from reduced set D'
    if $(y_i(w_t.x_i) \leq 0)$ **then**
      $w_{t+1} = w_t - \eta_t(\lambda w_t - \frac{1}{|D_{y_i}|} y_i x_i)$
    else if $(y_i(w_t.x_i) < 1$ **then**
      $w_{t+1} = w_t - \eta_t[\lambda w_t - \frac{1}{|D_{y_i}|} y_i x_i[1 - y_i(w_t.x_i)]]$
    else
      $w_{t+1} = w_t - \eta_t \lambda w_t$
    end
  end
end
return w_{t+1}

Figure2.1 :L'algorithme d'entraînement équilibré SVM-SGD pour les tâches de classification binaire

On peut voir que la marge peut être considérée comme la distance minimale entre deux coques convexes, H+ de la classe positive et H- de la classe négative (la distance la plus éloignée entre les deux classes). Le sous-échantillonnage de la classe négative (D' -) effectué par un entraînement équilibré SVM-SGD fournit la coque convexe réduite de H-, appelée H'-. Et ensuite, la distance minimale entre H+ et H' - est plus grande que celle entre H+ et H- (ensemble de données complet).

L’article confirme qu’il est plus facile d'atteindre la frontière de séparation que d'apprendre sur l'ensemble des données, par conséquent, la tâche d’entraînement de SVM-SGD équilibré est rapide à converger vers la solution.

Pour les problèmes à k classes, l'algorithme SVM-SGD multi-classe entraîne indépendamment k classificateurs binaires. Bien que l'équilibre d'entraînement équilibré, SVM-SGD traite les tâches de classification binaire avec rapidité, l'algorithme SVM-SGD multi-classes ne profite pas des avantages du calcul haute performance.

L'entraînement parallèle pour le SVM-SGD multi classe est décrit dans la figure 12. Le nombre de processus MPI dépend de la capacité mémoire du système HPC utilisé.

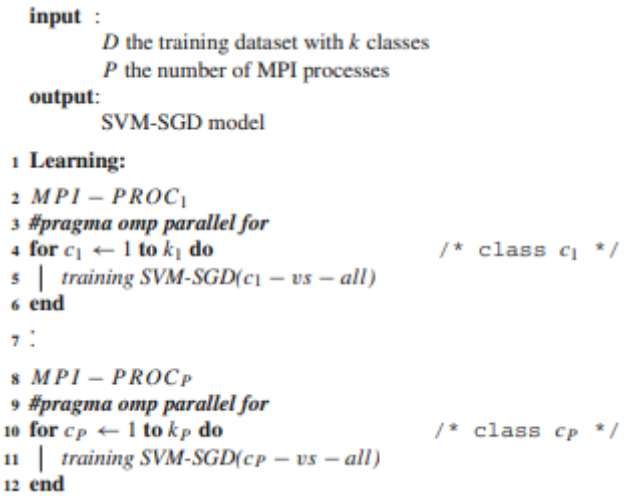

Figure2.2 : L’algorithme SVM-SGD multi classe parallèle hybride avec MPI et OpenMP

L'évaluation sur 1000 classes d'ImageNet, ILSVRC 2010 montre que cette algorithme est 270 fois plus rapide que le classificateur linéaire de pointe LIBLINEAR.

2.2 ARTICLE 2: Optimization of Regression Analysis by Conducting Parallel Calculations

Cet article étudie l’analyse de régression de grands ensembles de données par parallélisation. Il propose une parallélisation de la méthode de descente du gradient et de la descente de gradient stochastique en utilisant les technologies OpenMP et MPI, ainsi que leur hybride, pour optimiser l'analyse de régression.

Les étapes de l'article de la parallélisation de l'algorithme de descente du gradient avec la technologie OpenMP sont les suivant :

Dans la **première étape**, seul le fragment de code de l'ensemble de l'algorithme, qui est chargé de la multiplication de la matrice X par le vecteur θ dans la fonction d'hypothèse est parallélisé à l'aide de la directive directive OpenMP.

Remarque : L'hypothèse est une fonction qui décrit le mieux l'objectif du machie Learning

```
double* hypothesisP(double **X, double *Q, int m, int n) {
    double* result = new double[m];
    #pragma omp parallel for num_threads(2)
    for (int i = 0; i < m; i++) {
          result[i] = 0;
          for (int j = 0; j < n; j++) {
                result[i] += X[i][j] * Q[j];
          }
    }
    return result; }
```

Figure2.3 : Extrait du code de la première étape

A cette **deuxième étape**, la parallélisation des calculs nécessaires pour trouver le résultat de la fonction de coût. C'est-à-dire, la parallélisation de la fonction d'hypothèse, qui est décrite dans la première étape, et la parallélisation de l'opération de sommation des erreurs pour chaque élément.

```
double costP(double** X, double* Y, double* Q, int m, int n, int k) {
    double* hQ = new double[m];
    hQ = hypothesisP(X, Q, m, n);
    double* temp = new double[m];
    double sum = 0;
    #pragma omp parallel for num_threads(2)
    for (int i = 0; i < m; i++) {
          sum += (hQ[i] - Y[i]) * X[i][k];
    }
    return sum / m;
}
double* hypothesisP(double **X, double *Q, int m, int n) {
    double* result = new double[m];
    #pragma omp parallel for num_threads(2)
    for (int i = 0; i < m; i++) {
          result[i] = 0;
          for (int j = 0; j < n; j++) {
                result[i] += X[i][j] * Q[j];
}}
    return result;}
```

Figure2.4 : Extrait du code de la deuxième étape

Dans cette **troisième étape**, la parallélisation décrite à la deuxième étape est effectuée, mais avec une différente distribution des itérations (type, chunk_size)

```
double costP(double** X, double* Y, double* Q, int m, int n, int k) {
    double* hQ = new double[m];
    hQ = hypothesisP(X, Q, m, n);
    double sum = 0;
    #pragma omp parallel for num_threads(2) schedule(static, 250000)
    for (int i = 0; i < m; i++) {
          sum += (hQ[i] - Y[i]) * X[i][k];
    }
    return sum / m;}
double* hypothesisP(double **X, double *Q, int m, int n) {
    double* result = new double[m];
    #pragma omp parallel for num_threads(2) schedule(static, 250000)
    for (int i = 0; i < m; i++) {
          result[i] = 0;
          for (int j = 0; j < n; j++) {
                result[i] += X[i][j] * Q[j];}}
    return result;}
```

Figure2.5 : Extrait de la troisième étape

La parallélisation de l'algorithme de la descente du gradient stochastique peut se faire en bloquant le changement des coefficients par chaque processus, mais cette approche ne permet évidemment pas d'obtenir des avantages significatifs.

Cependant, il existe un algorithme intéressant qui permet de modifier les coefficients sans synchronisation. Cette méthode fonctionne sans aucun impact négatif sur l'efficacité mathématique de l'algorithme. La description de l'algorithme est la suivante :

Chaque Thread tire un exemple aléatoire i à partir des données d'entraînement.

- Le Thread lit l'état actuel de θ. $\theta \leftarrow (\theta - \alpha \nabla L(f_\theta(x_i), y_i))$
- Mises à jour du Thread

```
#pragma omp parallel for num_threads(2)
    for (int i = 0; i < m; i++) {
          for (int j = 0; j < n; j++) {
                tempQ[j] = Q[j] - alpha * (h(X[i], Q, n) - Y[i]) * X[i][j];

          }
          for (int k = 0; k < n; k++) {
                Q[k] = tempQ[k];
          }
    }
```

Figure2.6 : Extrait du code pour deux Threads

L'article présente aussi la parallélisation de l'algorithme de descente de gradient et de la descente de gradient stochastique en utilisant la technologie MPI.

La parallélisation de la descente du gradient se produit par le fait que les lignes de la matrice X[i] sont distribuées entre les Threads. Chaque thread calcule le résultat de la fonction de coût sur une quantité égale de données qui lui sont allouées, puis envoie un calcul à partir de chaque sélection de données et le thread principal, basé sur ces données, effectue des modifications au modèle.

```
DWORD Start = GetTickCount();
    if (ProcRank == 0) {
          double* tempQ = new double[n];
          for (int iter = 0; iter < 20; iter++) {
                for (int qiter = 0; qiter < n; qiter++) {
                      for (int p = 1; p < ProcNum; p++) {
                            MPI_Send(Q, n, MPI_DOUBLE, p, 1, MPI_COMM_WORLD);
                            MPI_Send(&qiter, 1, MPI_INT, p, 1, MPI_COMM_WORLD);
                      }
                      double procSum = costMpi(X, Y, Q, m, n, qiter, ProcNum, ProcRank);
                      double allSum = procSum;
                      for (int pr = 1; pr < ProcNum; pr++)
                      {
                      double curSum = 0;
                      MPI_Recv(&curSum,  1,  MPI_DOUBLE,  pr,  MPI_ANY_TAG,  MPI_COMM_WORLD,
&Status)
                                                                        allSum += curSum;
                      }
                      tempQ[qiter] = Q[qiter] - alpha * (allSum / m);
                }
                for (int qiter = 0; qiter < n; qiter++) {
                      Q[qiter] = tempQ[qiter];
                }
          }
    }
    else {

          for (int t = 0; t < n * 20; t++) {
                int k = 0;
                MPI_Recv(Q, n, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &Status);
                MPI_Recv(&k, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &Status);
                double procSum = costMpi(X, Y, Q, m, n, k, ProcNum, ProcRank);
                MPI_Send(&procSum, 1, MPI_DOUBLE, 0, ProcRank, MPI_COMM_WORLD);
          }}
    MPI_Finalize();
    DWORD dwRunTime = GetTickCount() - Start;
```

Figure2.7: Parallélisation de l'algorithme de descente du gradient avec MPI

La parallélisation de la descente du gradient stochastique se produit aussi par le fait que les lignes de la matrice X[i] sont distribuées entre les Threads. Chaque Thread recherche un modèle local, puis envoie le modèle trouvé au Thread principal, où tous les modèles locaux sont moyennés et on obtient un modèle global. L'inconvénient de cette méthode est qu'elle nécessite des calculs supplémentaires pour trouver le modèle.

```
DWORD Start = GetTickCount();
double* tempQ = new double[n];
double* localQ = new double[n];
if (ProcRank == 0) {

	int end = m / ProcNum;
	for (int i = 0; i < end; i++) {
		for (int j = 0; j < n; j++) {
		tempQ[j] = Q[j] - alpha * (h(X[i], Q, n) - Y[i]) * X[i][j];
		}
		for (int k = 0; k < n; k++) {
			Q[k] = tempQ[k];
		}
	}
	for (int p = 1; p < ProcNum; p++) {
		MPI_Recv(localQ, n, MPI_DOUBLE, p, 1, MPI_COMM_WORLD, &Status);
		for (int qiter = 0; qiter < n; qiter++) {
			Q[qiter] += localQ[qiter];
		}
	}
	for (int qiter = 0; qiter < n; qiter++) {
		Q[qiter] /= ProcNum;
	}
}
else {
	int start = ProcRank * (m / ProcNum);
	int end = ProcRank * (m / ProcNum) + (m / ProcNum);

	for (int i = start; i < end; i++) {
		for (int j = 0; j < n; j++) {
		tempQ[j] = Q[j] - alpha * (h(X[i], Q, n) - Y[i]) * X[i][j];
		}
		for (int k = 0; k < n; k++) {
			Q[k] = tempQ[k];
		}
	}
	MPI_Send(Q, n, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD);
}
```

Figure2.8: Parallélisation de l'algorithme de descente du gradient stochastique avec MPI

Dans ce même article, une accélération de 5 fois a été prouvée en utilisant l'architecture à six cœurs d'un ordinateur personnel.

Conclusion

Ce chapitre a permis de donner une idée sur les différentes propositions de a paralélisation de la descente du gradient stochastique dans différents contextes et ce , en utilisant es technoogies MPI et OpenMP.

CHAPITRE 3 ANALYSE EXPERIMENTALE / RESULTATS

Introduction

Avec l'ère du Big Data l'algorithme de la descente du gradient stochastique devient de plus en plus intéressant et de plus en plus pratique que l'algorithme de la descente du gradient. La descente du gradient est un algorithme utilisé pour retrouver les minimums locaux d'une fonction de façon itérative, et parmi ses utilisations les plus pertinentes est l'entraînement des modèles du Machine Learning par l'application de cette dernière sur ce qu'on appelle la fonction de l'erreur empirique.

Un modèle de Machine Learning est défini par un ensemble de paramètres qui sont eux aussi défini rigoureusement en fonction d'un ensemble de données d'entrées de sorte que ce modèle donne des prédictions précises.

Dans ce troisième chapitre on va voir de très près les algorithmes de la descente du gradient et la descente du gradient stochastique, on va présenter les outils avec lesquels on a travaillé, la parallélisation en pseudo code et mener notre propre expérimentation de la parallélisation le Multithreading en OpenMP.

3.1 Analyse expérimentale

3.1.1 La descente du gradient

Etant donné un modèle de Machine Learning, qui est défini par un ensemble de paramètres, l'algorithme de la descente du gradient commence avec un ensemble initial de valeurs de paramètres et se déplace itérativement vers un ensemble de valeurs minimisant la fonction $\mathbf{w} = \mathbf{w} - \boldsymbol{\eta}\, \boldsymbol{\Sigma \nabla C}(\mathbf{w})$

Avec $\mathbf{C}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{C_i}(\boldsymbol{w})$

La fonction de coût usuellement utilisée pour le processus d'entraînement d'un modèle du Machine Learning se présente comme suit : $\mathbf{C}(\mathbf{w}) = \frac{\mathbf{1}}{\boldsymbol{2N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} ||\mathbf{Y}_{\boldsymbol{actuelle}}(\mathbf{x}_{\boldsymbol{i}}) - \mathbf{Y}_{\boldsymbol{prédite}}(\mathbf{x}_{\boldsymbol{i}})||^{\mathbf{2}}$

Cette fonction est donc facile à paralléliser comme tous les termes sont indépendants les uns des autres, on peut idéalement affecter N unités de traitement pour calculer chaque terme de la série indépendamment.

3.1.2 La descente du gradient stochastique

Là, le gradient dans chaque itération est approximé par le gradient d'un seul exemple x_i avec son étiquette y_i où on choisit au hasard dans le lot d'échantillon pour chaque itération puis on met à jour le paramètre avant la prochaine itération. Le nombre total d'itérations nécessaire est plus élevé que celui de la descente du gradient mais comme un seul échantillon qui est choisi à chaque itération, le nombre total de calculs du gradient est nettement inférieur pour les grands ensembles de données.

$$\mathbf{w} = \mathbf{w} - \boldsymbol{\eta}\, \boldsymbol{\Sigma \nabla C_i}$$

La fonction de coût devient :

$\boldsymbol{C_i = \frac{1}{2}\sum_{i=1}^{N} ||\mathbf{Y}_{actuelle}(\mathbf{x}_i) - \mathbf{Y}_{prédite}(\mathbf{x}_i)||^2}$Pour un certain i ∈ 1 : n choisi aléatoirement.

Elle est plus rapide lorsque la taille des données est très importante et elle est couramment utilisée aujourd'hui dans le Machine Learning.

3.1.3 Présentation de Cygwin

Cygwin est une simulation de l'environnement UNIX sous les systèmes Windows. Il comprend une couche d'émulation et une collection d'outils qui fournissent un aspect et une sensation Linux.

Il se compose d'une DLL nommée cygwin1.dll, elle agit comme une couche d'émulation qui fournit la fonctionnalité d'appel système POSIX sur Windows. Les utilisateurs grâce à Cygwin ont accès aux utilitaires UNIX standard, qui peuvent être utilisés à partir du shell bash fourni ou via l'invite de commande Windows.

Il existe plusieurs avantages de l'utilisation du shell Cygwin par rapport à Invite de commande Windows.

- Il fournit un shell UNIX à Windows, permettant d'accéder à une gamme d'utilitaires du monde UNIX / Linux à Windows.
- Il n'est pas nécessaire d'installer un système d'exploitation Linux à part entière ou de configurer une machine virtuelle où l'exigence est satisfaite par les ressources disponibles via Cygwin.
- L'utilisation optimale des ressources et configuration système requise pour fonctionner sous Windows puisque l'environnement est émulé et fonctionne au-dessus de Windows.
- Il est idéal pour les tests / développement où l'exigence est d'utiliser les utilitaires UNIX / Linux sous Windows.
- Il est compatible avec les anciens systèmes d'exploitation Windows tels que Windows 7, etc., tandis que l'environnement WSL proposé par Windows n'est pris en charge que sur les versions plus récentes.

3.1.4 Présentation de MPI et d'OpenMP

MPI est une norme conçue en 1993-1994 pour le passage de messages entre ordinateurs distants ou dans un ordinateur multiprocesseur. Elle est devenue de facto un standard de communication pour des nœuds exécutant des programmes parallèles sur des systèmes à mémoire distribuée. Elle définit une bibliothèque de fonctions, utilisable avec les langages C, C++ et Fortran.

MPI a été écrite pour obtenir de bonnes performances aussi bien sur des machines massivement parallèles à mémoire partagée que sur des clusters d'ordinateurs hétérogènes à mémoire distribuée. Elle est disponible sur de très nombreux matériels et systèmes d'exploitation. Ainsi, MPI possède l'avantage par rapport aux plus vieilles bibliothèques de passage de messages d'être grandement portable car MPI a été implémentée sur presque toutes les architectures de mémoires, et rapide car chaque implémentation a été optimisée pour le matériel sur lequel il s'exécute.

OpenMP est une interface de programmation pour le calcul parallèle sur architecture à mémoire partagée. Cette API est prise en charge par de nombreuses plateformes, incluant GNU/Linux, OS X et Windows, pour les langages de programmation C, C++ et Fortran. Il se présente sous la forme d'un ensemble de directives, d'une bibliothèque logicielle et de variables d'environnement.

OpenMP est portable et dimensionnable. Il permet de développer rapidement des applications parallèles à petite granularité en restant proche du code séquentiel.

3.1.5 Description de l'environnement de travail

Durant ce travail on va utiliser le simulateur de l'environnement UNIX Cygwin version 2.909 sous le système d'exploitation Microsoft Windows 10 Professionnel version 10.0.19042 Build 19042. La machine utilisée est un ordinateur personnel modèle HP EliteBook 840 G1 à base de x64 avec un processeur d'Intel Core i5 4ème génération (i5-4300U). Le CPU est composé de deux cœurs et 4 processeurs logiques. La fréquence de base du processeur est de 1,90 GHz et la fréquence maximale du turbo est de 2501 MHz. La mémoire

physique RAM installée dans ma machine est de 8.00 Go, la mémoire physique totale est de 7.90 Go, et la mémoire physique disponible est de 4.08 Go.

3.1.6 Travail relatif

L'article de Martin A. Zinkevich, Markus Weimer, Alex Smola et Lihong Li discute les problèmes rencontrés lors de l'utilisation des données à l'échelle industrielle, l'algorithme du SGD prenait des heures pour de grands ensembles de données sur un seul disque dur avec une capacité donnée.

L'utilisation de l'architecture à mémoire distribuée s'avère intéressante en assignant des ensembles de données sur différents disques durs à différents processeurs.

Ils performent l'algorithme du SGD de manière indépendante sur leurs données respectives avec un taux d'apprentissage fixe pour un nombre fixe d'étapes et une fonction de coût appropriée, puis on fait la moyenne des gradients calculés par chaque processeur à la fin.

La parallélisation proposée de la descente du gradient stochastique est la suivante en pseudo code :

NB_BLOC ← NB_PROCESSEURS

//Diviser l'ensemble d'entrée en NB_PROCESSEURS

1 : NB_SOUSBLOC← NB_BLOC / NB_PROCESSEURS

//Chaque processeur P_j a son propre vecteur de paramètre W_j avec $j \in$ [1 à NB_PROCESSEURS]

//Chaque precesseur P_j exécute le code suivant:

Pour i = 1 **à** NB_SOUSBLOCS **faire**

2 : Obtenir x_i aléatoirement du sous-bloc(i) du bloc(j)

3 : Calculer *grad(*x_i)

4 : $W_j = W_j$ - *grad(*x_i)

Fin pour

//Faire la moyenne

5 : $W \leftarrow \frac{1}{NB\_PROCESSEURS} \sum_{j=1}^{NB\_PROCESSEURS} W_j$

Le jeu de données utilisé est pris de Kaggle provient de l'institut national du diabète et des maladies digestives et rénales. L'objectif de ce jeu de données

est de prédire de manière diagnostique si un patient est diabétique ou non (problème de classification binaire) sur la base de certaines mesures diagnostiques incluses dans le jeu de données. En particulier, tous les patients sont des femmes âgées d'au moins 21 ans et d'origine indienne.
L'ensemble de données est de 768 observations sur 9 variables 8 de ces variables sont des variables prédictives médicales la dernière et 9ème variable est une variable cible qui est le résultat.
Ces variables sont respectivement :

1- **Nombre de grossesses,**
2- **Concentration de glucose dans le plasma (test de tolérance au glucose)**
3- **Tension artérielle diastolique (mm Hg)**
4- **Epaisseur du pli cutané du triceps (mm)**
5- **Insuline sérique à 2 heures (mu U/ml)**
6- **Indice de masse corporelle (poids en kg/(taille en m)^2)**
7- **Fonction pédigrée du diabète**
8- **Age (années)**
9- **Variable de classe (test de dépistage du diabète)**

1	Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age,Outcome
2	6,148,72,35,0,33.6,0.627,50,1
3	1,85,66,29,0,26.6,0.351,31,0
4	8,183,64,0,0,23.3,0.672,32,1
5	1,89,66,23,94,28.1,0.167,21,0
6	0,137,40,35,168,43.1,2.288,33,1
7	5,116,74,0,0,25.6,0.201,30,0
8	3,78,50,32,88,31,0.248,26,1
9	10,115,0,0,0,35.3,0.134,29,0
10	2,197,70,45,543,30.5,0.158,53,1
11	8,125,96,0,0,0,0.232,54,1
12	4,110,92,0,0,37.6,0.191,30,0
13	10,168,74,0,0,38,0.537,34,1
14	10,139,80,0,0,27.1,1.441,57,0
15	1,189,60,23,846,30.1,0.398,59,1
16	5,166,72,19,175,25.8,0.587,51,1
17	7,100,0,0,0,30,0.484,32,1
18	0,118,84,47,230,45.8,0.551,31,1
19	7,107,74,0,0,29.6,0.254,31,1
20	1,103,30,38,83,43.3,0.183,33,0
21	1,115,70,30,96,34.6,0.529,32,1
22	3,126,88,41,235,39.3,0.704,27,0
23	8,99,84,0,0,35.4,0.388,50,0
24	7,196,90,0,0,39.8,0.451,41,1

Figure3.1 : Extrait du fichier diabetes.csv téléchargé depuis Kaggle

Dans mon travail, j'ai supprimé la première ligne puis je l'ai converti en fichier texte afin de bien le manipuler et je l'ai nommé input.txt.

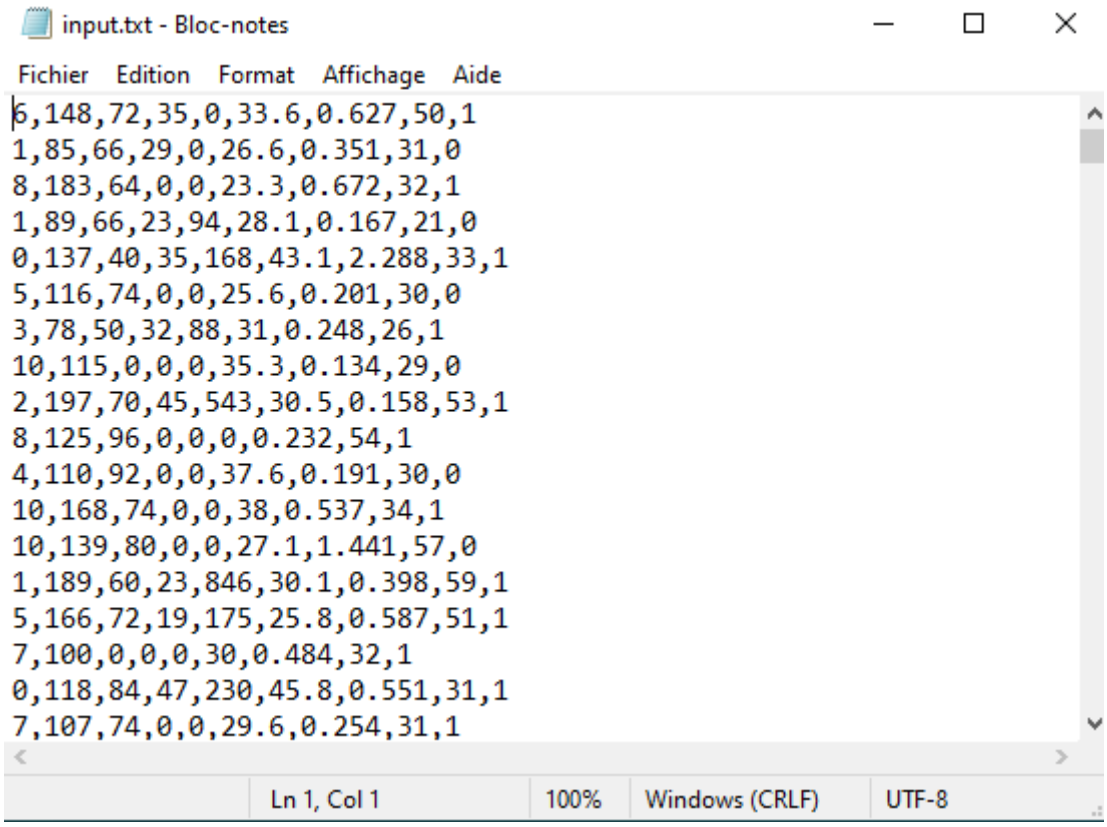

Figure3.2 : Extrait du fichier input.txt utilisé

Le modèle utilisé ici est la régression logistique $Y_i = h(z_i) = \frac{1}{1+e^{-z_i}}$ avec $z_i = W^T X_i + B_i$

Il s'agit d'un des modèles de Machine Learning les plus simples et interprétables qui existe, il prend des données à la fois continues ou discrètes, et les résultats obtenus avec sont loin d'être risibles.

La régression logistique est un modèle statistique permettant d'étudier les relations entre un ensemble de variables qualitatives Xi et une variable qualitative Y. Il s'agit d'un modèle linéaire généralisé utilisant une fonction logistique comme fonction de lien.

En mathématiques, les fonctions logistiques sont les fonctions qui s'écrivent de la manière $f(t) = \frac{K}{1+\alpha e^{-rt}}$ où K et r sont des réels positifs et α un réel quelconque.

Un modèle de régression logistique permet aussi de prédire la probabilité qu'un événement arrive (valeur de 1) ou non (valeur de 0) à partir de l'optimisation des coefficients de régression. Ce résultat varie toujours entre 0 et 1. Lorsque la valeur prédite est supérieure à un seuil, l'événement est susceptible de se produire, alors que lorsque cette valeur est inférieure au même seuil, il ne l'est pas.

Tout le problème de classification par régression logistique apparaît alors comme un simple problème d'optimisation où, à partir de données, nous essayons d'obtenir le meilleur jeu de paramètre permettant à notre courbe sigmoïde de coller au mieux aux données.

Il ne reste plus, à partir du seuil défini, qu'à classer les points en fonction de leurs positions par rapport à la régression et notre classification est faite.

Les variables indépendantes prises sont : la **tension artérielle diastolique (mm Hg)** et l'**indice de masse corporelle (poids en kg/(taille en m)²**.

Au début j'ai commencé par l'implémentation de l'algorithme du gradient stochastique en MPI et puis j'ai migré vers OpenMP c'est le choix le plus approprié vu que la machine de test est une machine à mémoire partagée multi-cœurs.

Ce qui suit les extraits du code de l'algorithme du gradient stochastique utilisant OpenMP.

```
#include <iostream> //Fonctions d'entrée/sortie standard
#include <fstream> //Manipulation des fichiers
#include <sys/time.h> //Calcul du temps
#include <math.h> //Fonctions mathématiques
#include <omp.h> //Fonctions de OpenMP
#include <stdlib.h> //fonctions générales
```

Figure3.3 : Inclure les fichiers Header

```
#define THRDS 3
#define MAXITER 10000000
#define SZ 768 //Nombre de lignes du fichier d'entrée
```

Figure3.4 : Définition des macros

```
//Définition du modèle de la régression logistique
double Y_model(double W1,double W2,double B,double X1,double X2)
{
    double Z = B + W1*X1 + W2*X2;
    return 1.0/( 1.0 + exp(-Z));
}
```

Figure3.5 : Définition du modèle de la régression logistique

Les extraits du code qui viennent sont dans la fonction main().

```
int main(int argc, char *argv[]) {
    struct timeval start, end;
    const int nThreads = THRDS;


//Création de 8 tableaux de même taille que le nombre de lignes du fichier
//Pour stocker toutes les valeurs de la même variable
    double *X1 = new double[SZ];
    double *X2 = new double[SZ];
    double *X3 = new double[SZ];
    double *X4 = new double[SZ];
    double *X5 = new double[SZ];
    double *X6 = new double[SZ];
    double *X7 = new double[SZ];
    double *X8 = new double[SZ];
//Pour stocker toutes les valeurs de la dernière variable (variable de classe)
    double *Y = new double[SZ];


    double W1 = 0, W2 = 0, B = 0, dW1 = 0, dW2 = 0, dB = 0;
    double y;
    double h = 0.0001;
```

Figure3.6 : Création de 9 tableaux pour stocker chaque valeur d'une même colonne du fichier d'entrée

```
    std::ifstream inputD("input.txt");
    char ch;
    int idx,i = 0;


//Stockage des valeurs dans les tableaux
    for(i = 0; i < SZ; i++)
    {

        inputD>>X1[i]>>ch>>X2[i]>>ch>>X3[i]>>ch>>X4[i]>>ch>>X5[i]>>ch>>X6[i]>>ch>>X7[i]>>ch>>X8[i]>>ch>>Y[i];

    }


    omp_set_num_threads(nThreads); //initialisation du nombre de Threads
```

Figure3.7 : Stockage des valeurs du fichier dans chaque tableau créé

```
    gettimeofday(&start,NULL);
//directive de la région parallèle
#pragma omp parallel shared(X3,X6,Y,W1,W2,B,h) private(i,dW1,dW2,dB,idx,y)
    {

#pragma omp for schedule(dynamic)

        for(i = 0; i <MAXITER;++i)
        {

            idx = rand()%SZ;
            y = Y_model(W1,W2,B, X3[idx], X6[idx]);

            dW1 = h*(Y[idx] - y)*y*(1.0 - y)*X3[idx];
            dW2 = h*(Y[idx] - y)*y*(1.0 - y)*X6[idx];
            dB  =  h*(Y[idx] - y)*y*(1.0 - y);

            W1 = W1 + dW1;

            W2 = W2 + dW2;

            B =  B +  dB;

        }

    }
    gettimeofday(&end,NULL);
```

Figure3.8 : La région parallèle et calcul du temps d'exécution

```
        double error = 0;

        //Calcul de l'erreur quadratique
        for(i =0; i<SZ; ++i)
        {
            error += pow((Y[i] - Y_model(W1,W2,B, X3[i], X6[i]) ),2);
        }
        error = sqrt(error);
        error = error/SZ;
```

Figure3.9 : Calcul de l'erreur quadratique

```
//Affichage du temps d'exécution et de l'erreur quadratique
    std::cout<<"Temps d'execution de chaque Thread "<<(end.tv_sec - start.tv_sec)*1000000 + end.tv_usec - start.tv_usec<<" microseconds. \n"<<'\n';
    std::cout<<"Calcul de l'erreur quadratique "<<error<<'\n';

    return 0;
}
```

Figure3.10 : Affichage du temps d'exécution et de l'erreur quadratique

3.2 Résultats et discussion

Pour la compilation de mon code, j'ai créé le fichier Makefile qui va être utilisé par la commande *make* que je vais appeler pour compiler le programme.

```
CXX = g++
CXXFLAGS = -Wall -std=c++11
CPPFLAGS = -I .
LDFLAGS = -fopenmp
SRC := LOGReg_OMP.cpp
OBJ := $(SRC:%.cpp=%.o)

EXEC = log_reg

all: $(EXEC)

$(EXEC): $(OBJ)
    @$(CXX) $(LDFLAGS) $^ -o $(EXEC)

%.o: %.cpp
    @$(CXX) -c $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) $< -o $@

clean:
    rm -f $(EXEC) $(OBJ)
```

Figure3.11 : Contenu du fichier Makefile

Pour exécuter mon code, il suffit d'aller dans le répertoire où se trouvent mes fichiers et éxecuter les commandes $ make pour la compilation et $ ./log_reg avec log_reg est le nom de mon exécutable que j'ai défini dans le Makefile.

Les figures suivantes représentent le résultat d'exécution de SGD implémenté en OpenMP et en changant le nombre de Threads respectivement de 1 à 2 à 3 et à 4.

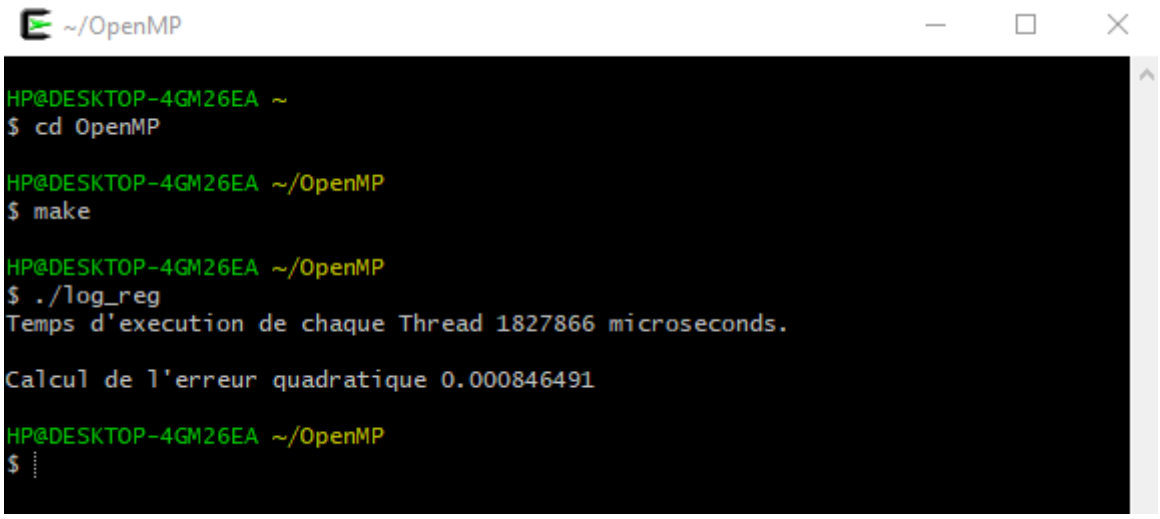

Figure 3.12 : Résultat d'exécution avec 1 seul Thread

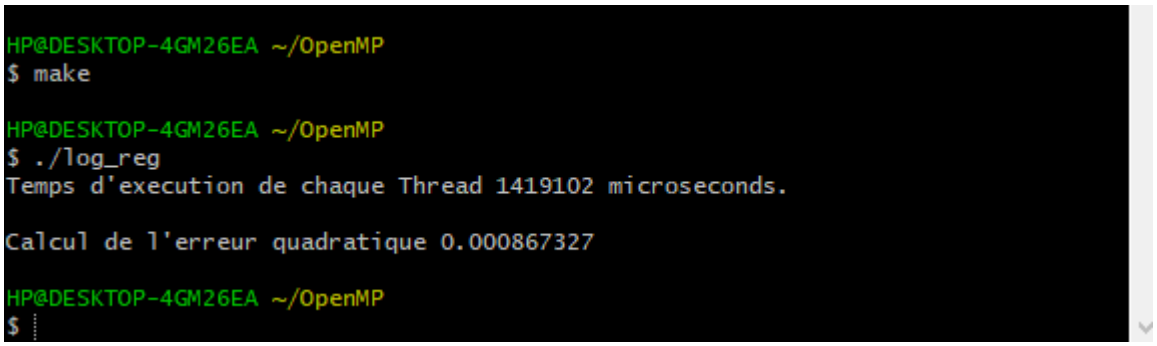

Figure 3.13 : Résultat d'exécution avec 2 Threads

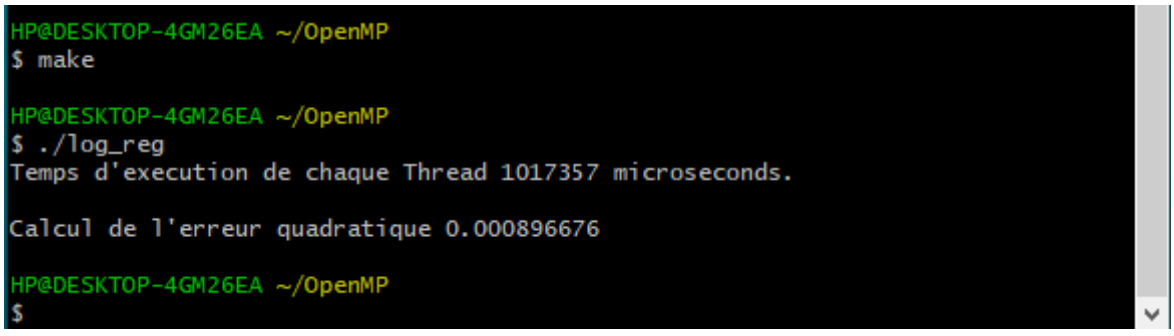

Figure 3.14 : Résultat d'exécution avec 3 Threads

```
HP@DESKTOP-4GM26EA ~/OpenMP
$ make

HP@DESKTOP-4GM26EA ~/OpenMP
$ ./log_reg
Temps d'execution de chaque Thread 842567 microseconds.

Calcul de l'erreur quadratique 0.000909402

HP@DESKTOP-4GM26EA ~/OpenMP
$
```

Figure 3.15 : Résultat d'exécution avec 4 Threads

Le tableau suivant représente le temps d'exécution en µs de l'algorithme du gradient stochastique avec OpenMP selon le nombre de processeurs logiques (Threads). L'accélération représente l'efficacité de la parallélisation par rapport à l'utilisation d'un seul Thread.

Nombre de Threads	Temps d'exécution (µs)	Accélération
1	1827866	1
2	1419102	1,288044129
3	1017357	1,796681008
4	842567	2,169401365

Figure3.16 : Tableau des temps d'exécutions et des accélérations en fonction des nombres des Cores

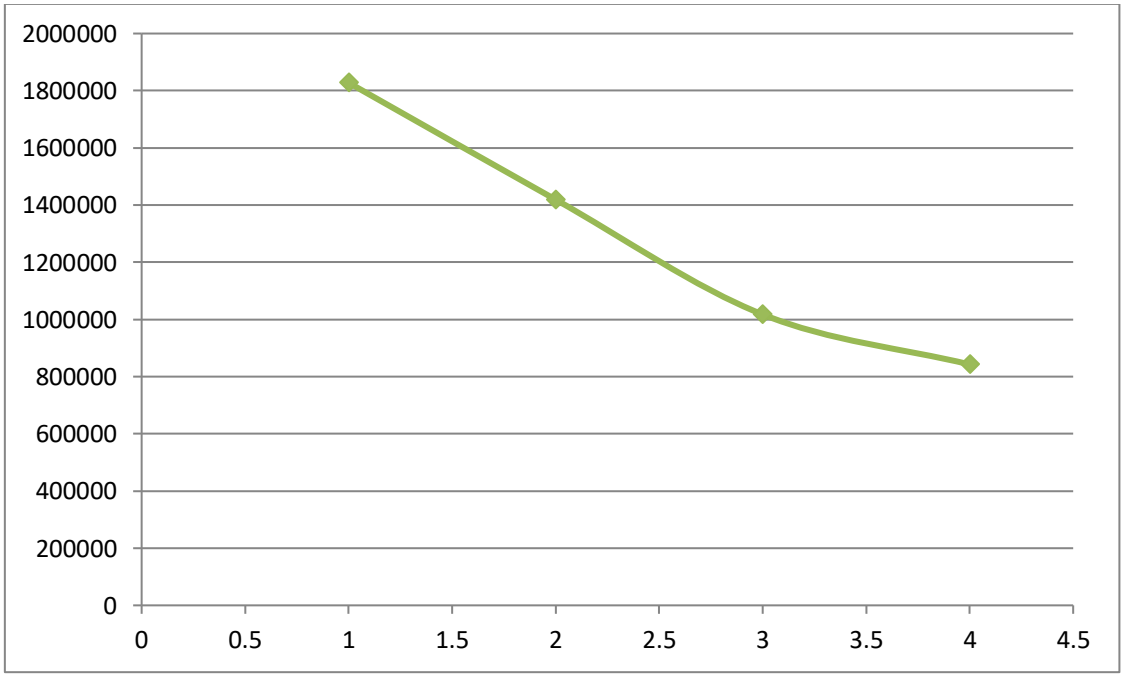

Figure3.17 : Schéma du temps d'exécution (µs) en fonction du nombre des Threads

On remarque nettement que le temps d'exécution diminue avec l'augmentation du nombre de Threads.

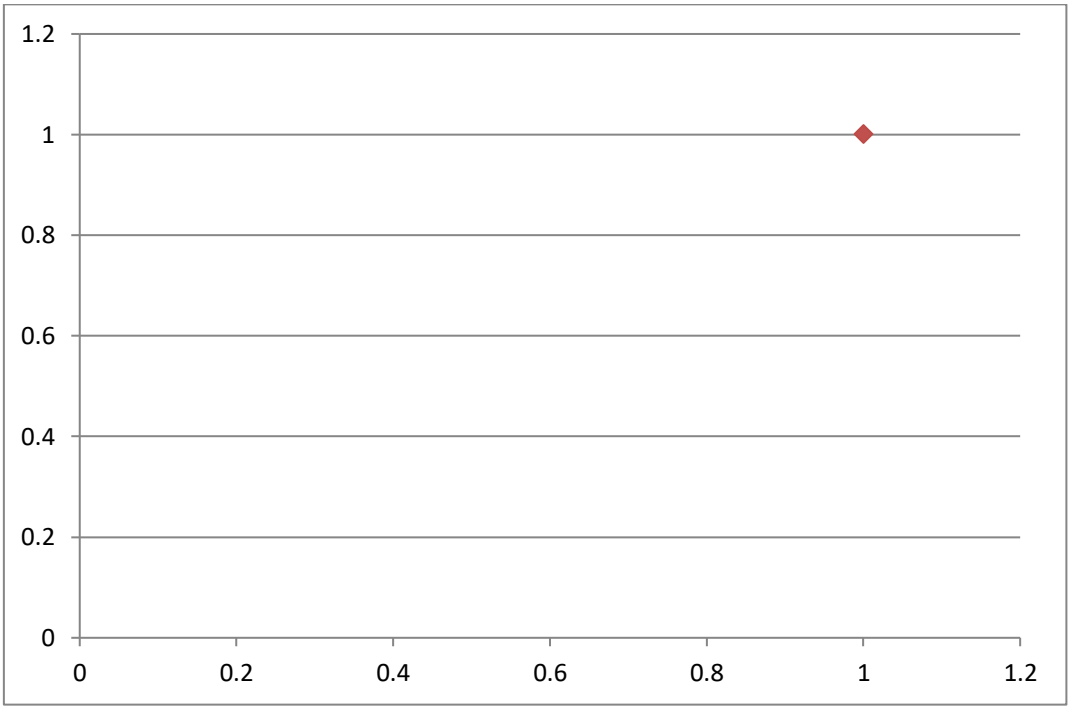

Figure3.18 : Schéma de l'accélération en fonction du nombre des Threads

On remarque bien une accélération évolutive avec l'augmentation du nombre des Threads.

L'erreur quadratique moyenne a été calculée (Voir les figures 3.12, 3.13, 3.14, et 3.15). A titre de rappel, l'erreur quadratique moyenne est la moyenne de la différence quadratique entre chaque point prédit et le point réel. C'est usuellement un nombre élevé parce qu'on met chaque différence au carré pour éliminer les nombres négatifs avant de prendre la moyenne.

Nombre de Threads	Erreur quadratique
1	0,000846491
2	0 ,000867327
3	0,000896676
4	0,000909402

Figure 3.19 : Tableau du l'erreur quadratique en fonction du nombre de Threads

Puisque nous avons utilisé un modèle logistique pour la classification (0,1), l'erreur n'est pas très significative.

Conclusion

Dans ce dernier chapitre, on a proposé notre parallélisation de l'algorithme de la descente du gradient stochastique en pseudo code et à partir de ce dernier on l'a implémenté avec OpenMP.

Annexes

Annexe A: Installation de Cygwin version 2.909

Dans ce qui suit, on va présenter les étapes d'installation de la dernière version de Cygwin.

Etape1 : On télécharge la version adaptée à l'architecture de notre système, dans notre cas on télécharge la version 64 via ce lien https://cygwin.com/setup-x86_64.exe.

Etape2 : On Lance le fichier exécutable téléchargé à partir de la première étape, et on clique sur suivant pour continuer le processus de configuration

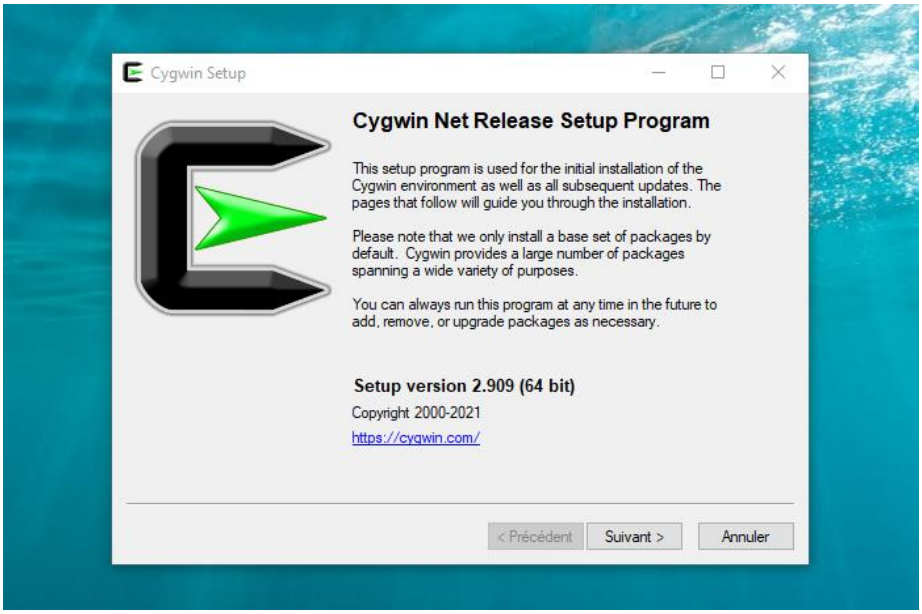

Figure A.1 : Lancement du fichier exécutable de Cygwin

Etape3 : Une invite pour sélectionner une source de téléchargement s'affiche. Dans la plupart des cas, l'option par défaut 'Installer à partir d'Internet' est correcte et doit être conservée. Si on a un référentiel téléchargé local ou si on souhaite simplement télécharger et installer Cygwin plus tard, choisissez l'une des autres options disponibles. Et on cliquer sur Suivant pour continuer.

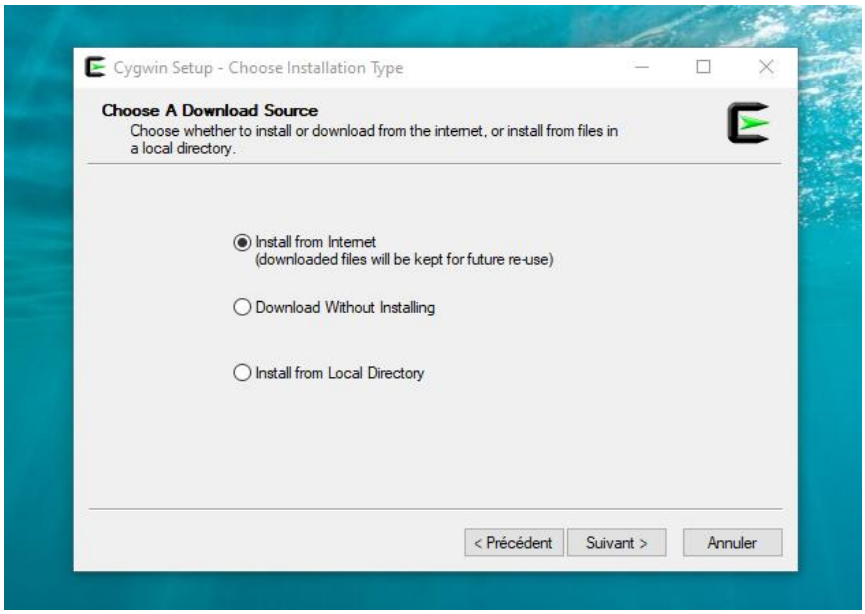

Figure A.2 : Sélectionnement d'une source de téléchargement

Etape4 : On nous invite ensuite à spécifier 'Répertoire racine' pour l'installation ainsi que s'il sera installé pour 'Tous les utilisateurs' ou 'Juste moi'. S'il n'y a pas de besoin spécifique, on conserve les options par défaut et on clique sur Suivant pour continuer.

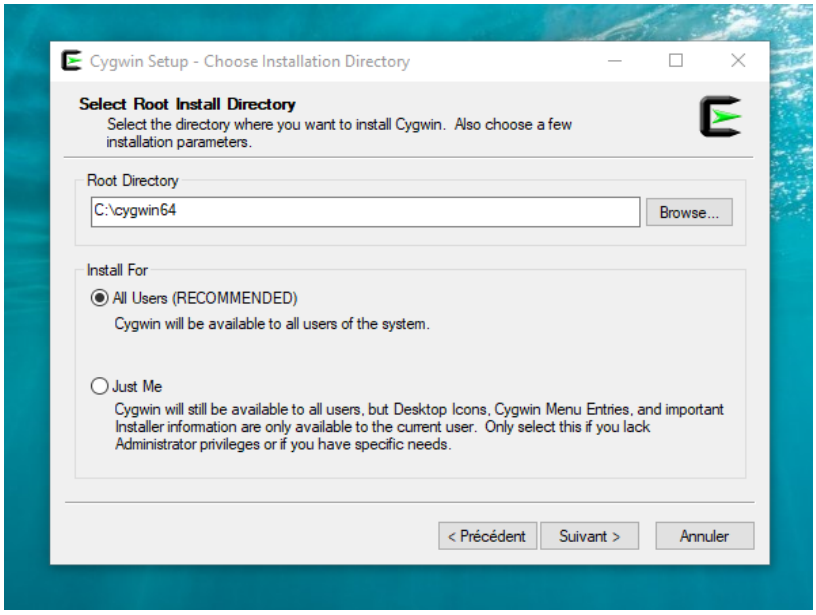

Figure A.3 : Répertoire racine et spécification de besoins

Etape5 : Pour télécharger les fichiers de package, le programme d'installation nous demandera de spécifier un ‘Répertoire de packages local’ qui, par défaut, pointe vers le dossier «Downloads» de notre système. Si on souhaite spécifier un autre emplacement, on spécifie son chemin dans cette étape et on clique sur Suivant pour procéder à la configuration.

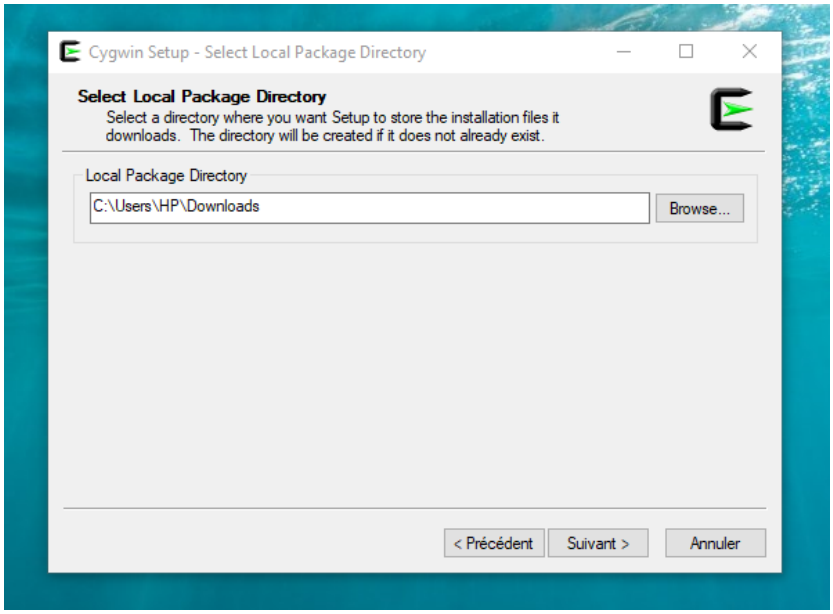

Figure A.4 : Répertoire de package local

Etape6 : Dans la plupart des cas, ‘Utilisation des paramètres du proxy système’ devrait fonctionner pour le téléchargement des packages. Si on n'a pas l'intention d'utiliser de proxy et d'utiliser une connexion directe à Internet, on utilise l'option ‘Connexion directe’ ou on peut également spécifier un paramètre de proxy personnalisé. On Clique sur Suivant pour continuer.

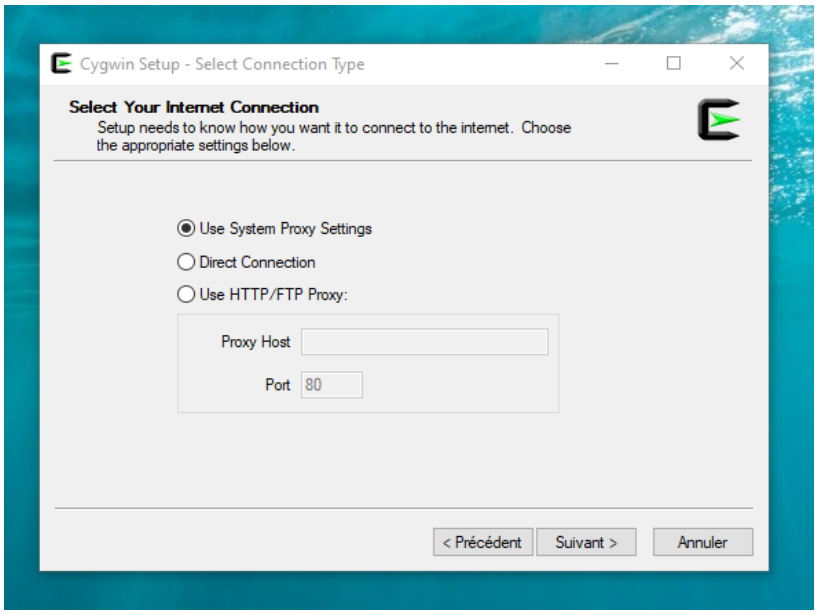

Figure A.5 : Sélectionnement de la connexion Internet

Etape 7 : Ensuite, on nous demandera de sélectionner un miroir à partir duquel Cygwin téléchargera ses fichiers de package. Si un miroir se trouve à proximité de notre emplacement, il peut fournir une vitesse plus rapide, on choisi le dans la liste. En cas de doute, on sélectionne n'importe qui dans la liste et on clique sur Suivant pour continuer.

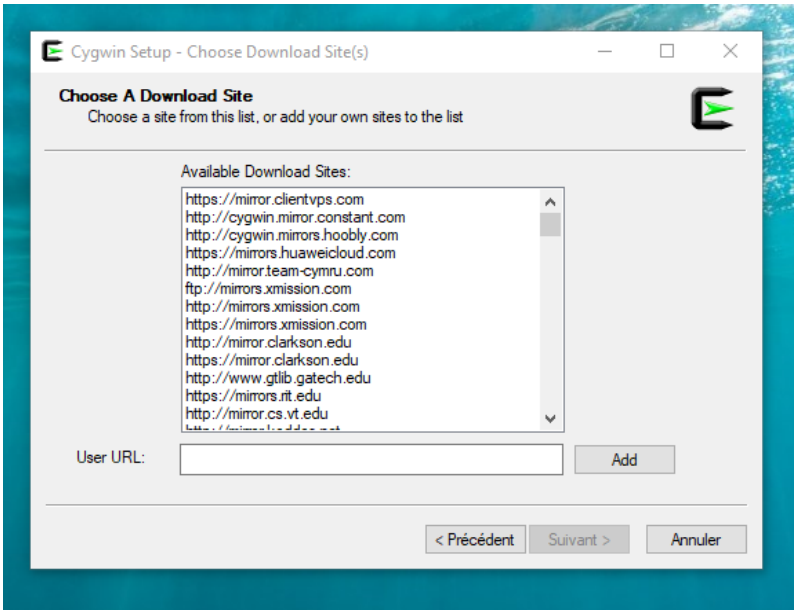

Figure A.6 : Sélectionnement du miroir à partir duquel Cygwin téléchargera ses fichiers de package

Annexe B : Installation des packages Cygwin : compilateur, éditeur et bibliothèques OpenMP et MPI

Etape 8 : Cygwin fournit de nombreux utilitaires UNIX / Linux qui s'exécutent sous Windows. Tous les packages ne sont pas installés par défaut. Seul un ensemble minimal de packages importants est installé.

Dans cette étape, on va personnaliser notre configuration Cygwin en sélectionnant catégories de packages. Cela peut également être fait plus tard pour ajouter / supprimer des packages à notre installation Cygwin. On sélectionne maintenant notre liste spécifique de packages importants pour notre travail comme le montre les figures ci-dessus et on continue en cliquant sur Suivant.

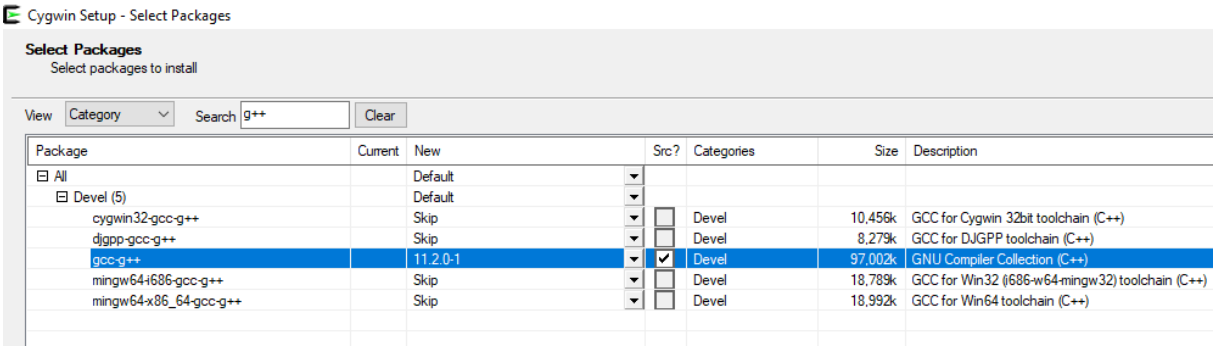

Figure B.1 : GNU Compiler Colection (C++) version 11.2.0-1

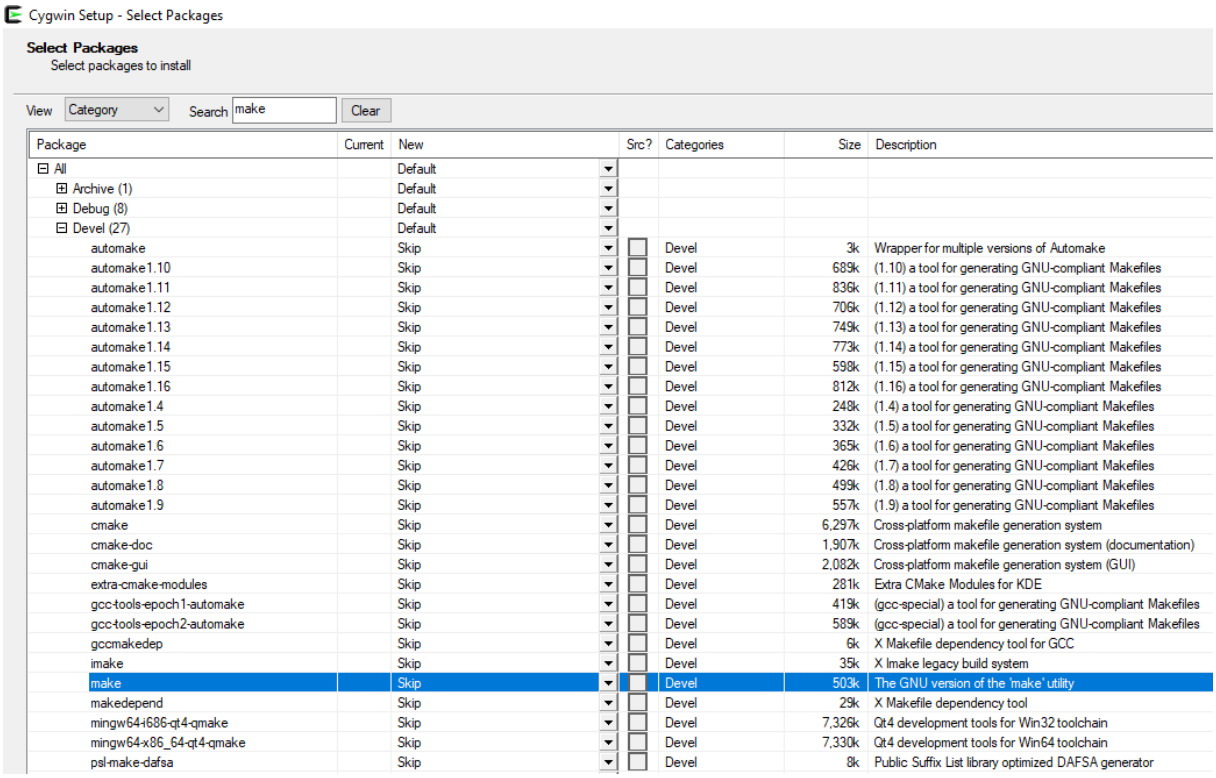

Figure B.2: The GNU version of the 'make' utility version 4.3-1

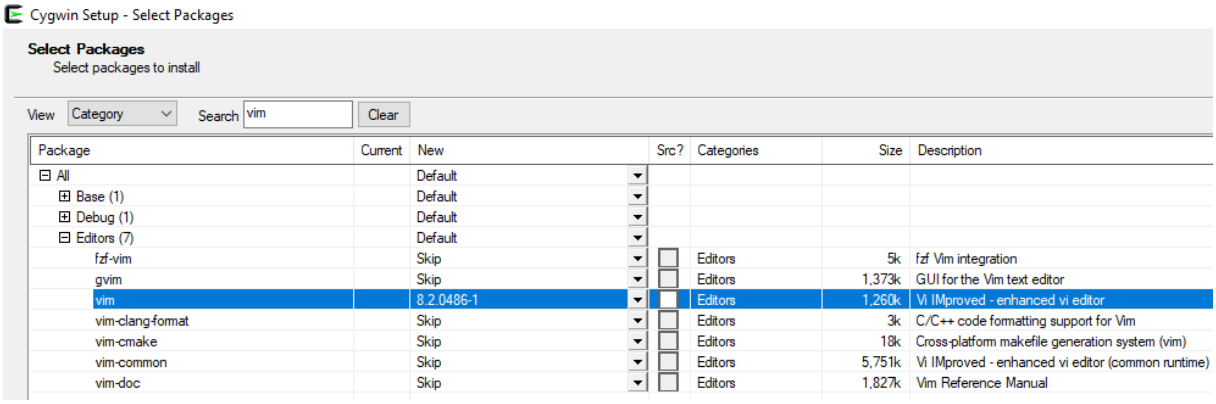

Figure B.3: Vi IMproved – enhanced vi editor version 8.2.0486-1

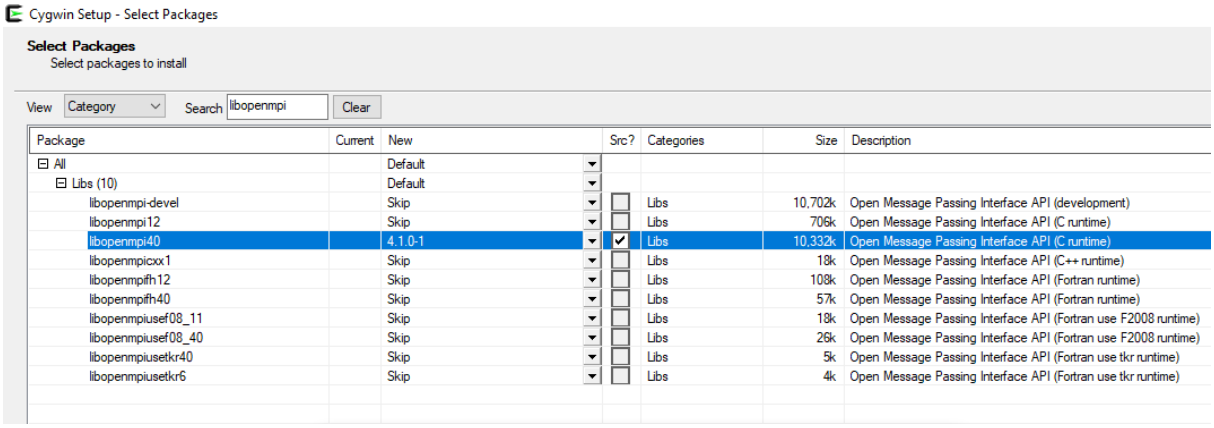

Figure B.4: Open Message Passing Interface API (C runtime) version 4.1.0-1

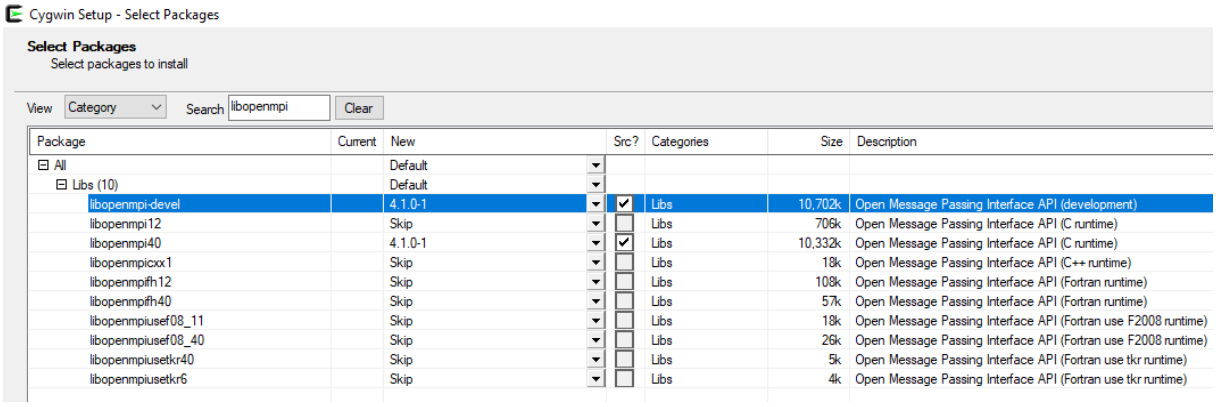

Figure B.5: Open Message Passing Interface API (development) version 4.1.0-1

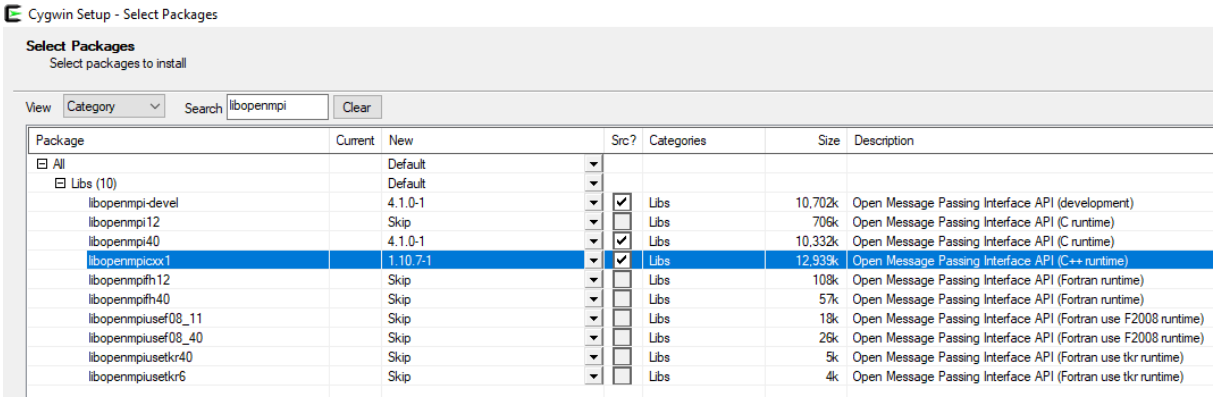

Figure B.6: Open Message Passing Interface API (C++ runtime) version 1.10.7-1

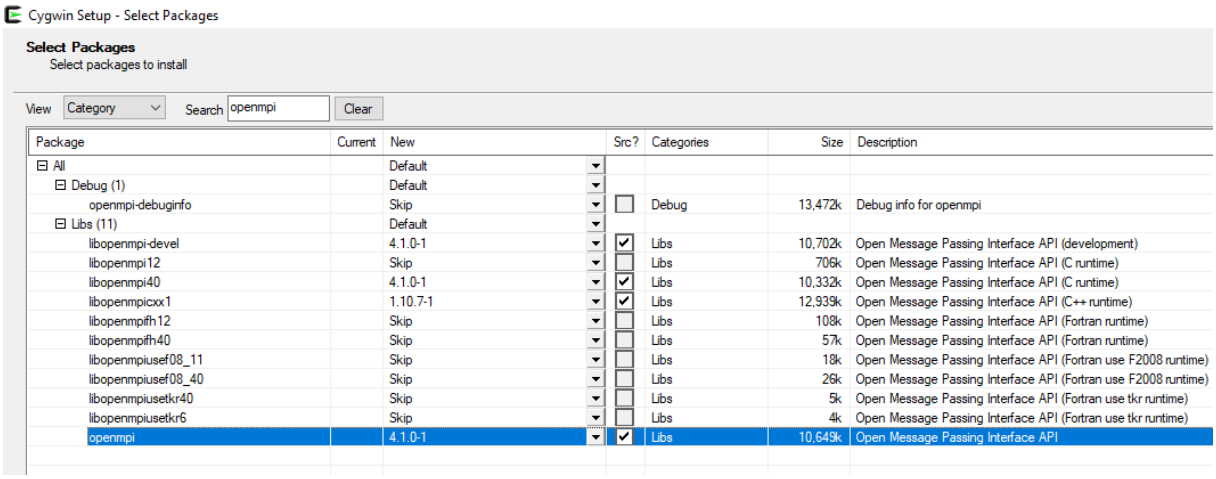

Figure B.7: Open Message Passing Interface API version4.1.0-1

Etape 9 : La fenêtre répertorie les packages sélectionnés et nous demande de les revoir et de les confirmer. On clique sur Suivant pour confirmer et continuer.

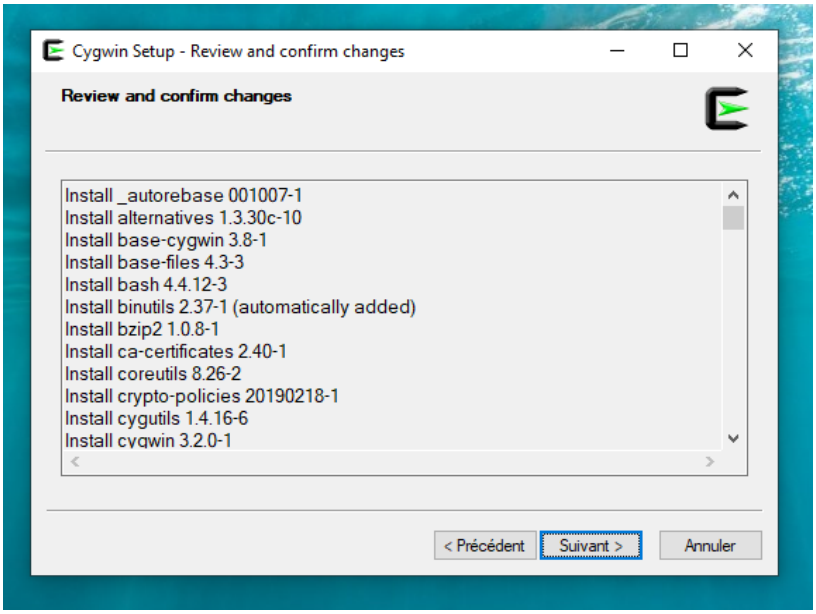

Figure B.8 : Packages sélectionnés

Etape 10 : Le téléchargement des packages sélectionnés commencera, Il va prendre un certain temps en fonction de la vitesse du miroir de téléchargement sélectionné et du nombre de packages sélectionnés.

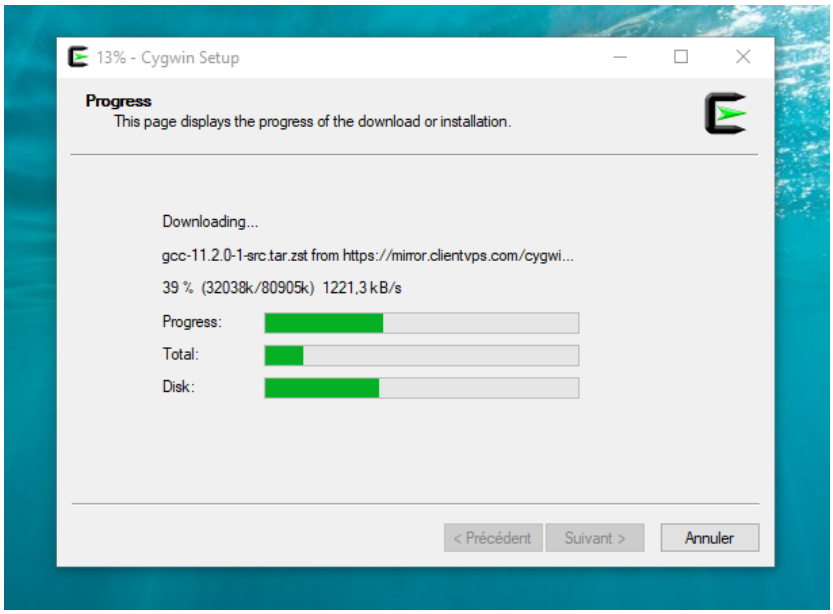

Figure B.9 : Téléchargement des packages

Etape11 : Une fois tous les packages téléchargés, le programme d'installation continuera à installer les packages.

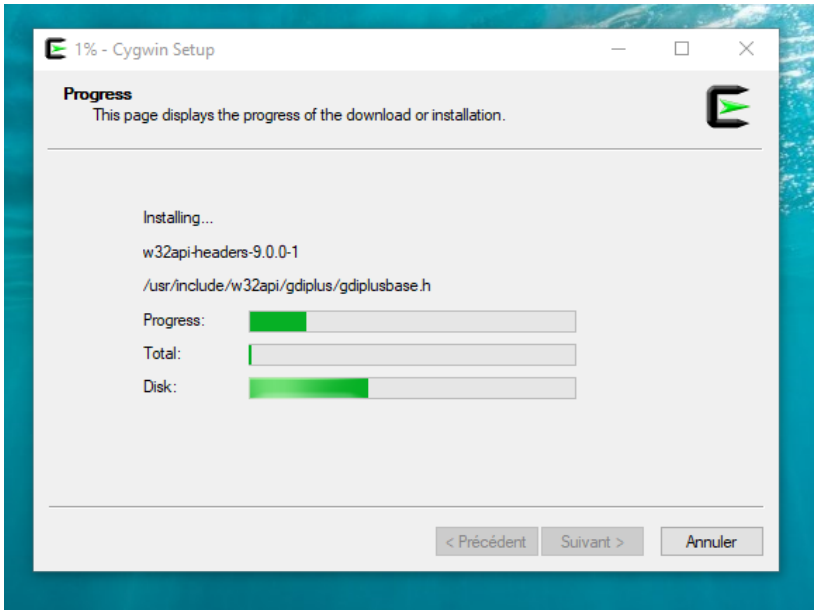

Figure B.10 : Installation des packages

Etape12 : Une fois la configuration terminée, On nous donne la possibilité de créer des icônes sur le bureau et le menu Démarrer pour un accès facile. On clique sur Terminer pour terminer l'assistant de configuration.

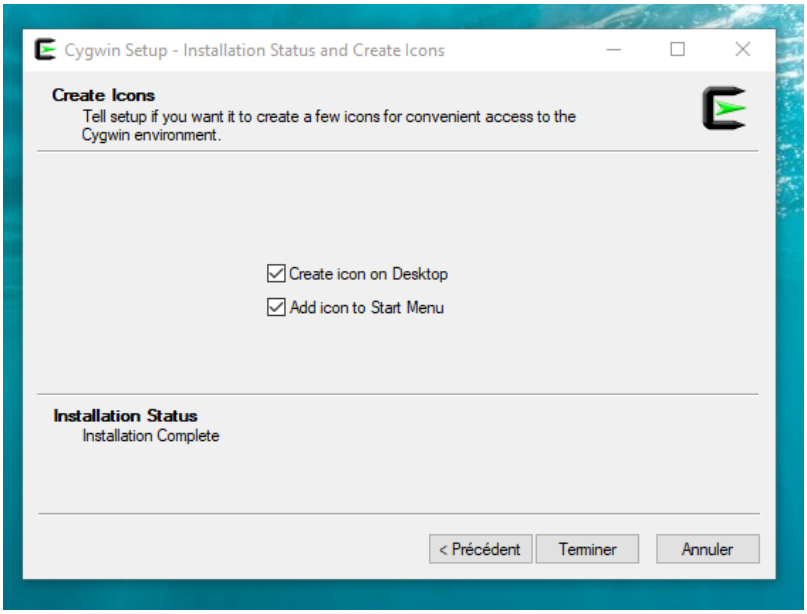

Figure B.11 : Création des icônes

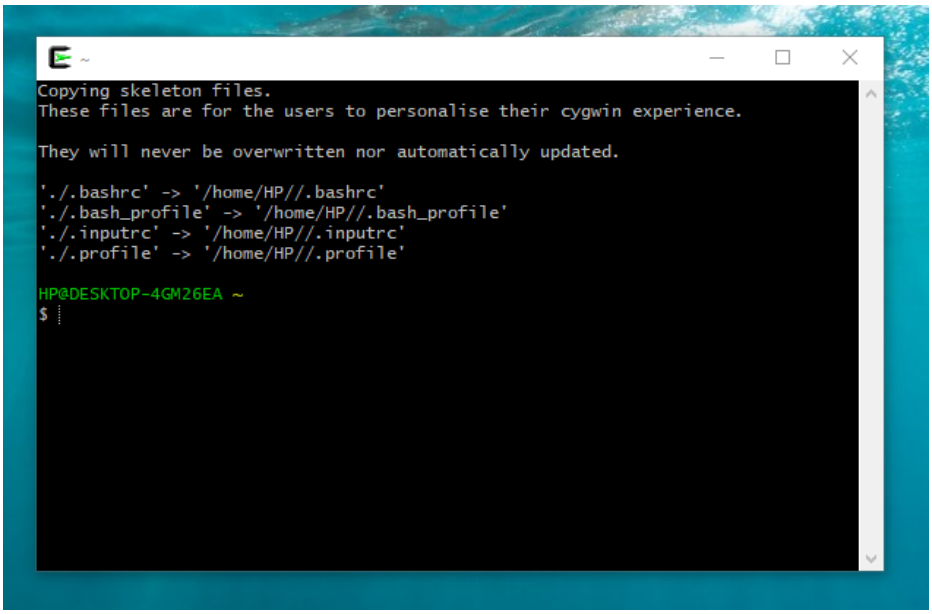

Figure B.12 : Shell Cygwin

Cygwin utilise par défaut notre répertoire d'origine Windows, ce répertoire est dans notre cas: *C:\cygwin64\home\HP* où HP est notre nom d'utilisateur Windows, Il s'agit de notre répertoire 'home' dans le 'Monde UNIX'.

Etape 13: L'installation ne semble pas configurer la variable d'environnement Path, nous devons donc le faire à la main, O se dirige à la fin de la variable système Path et on ajoute ceci : *C:\cygwin64\usr\bin*

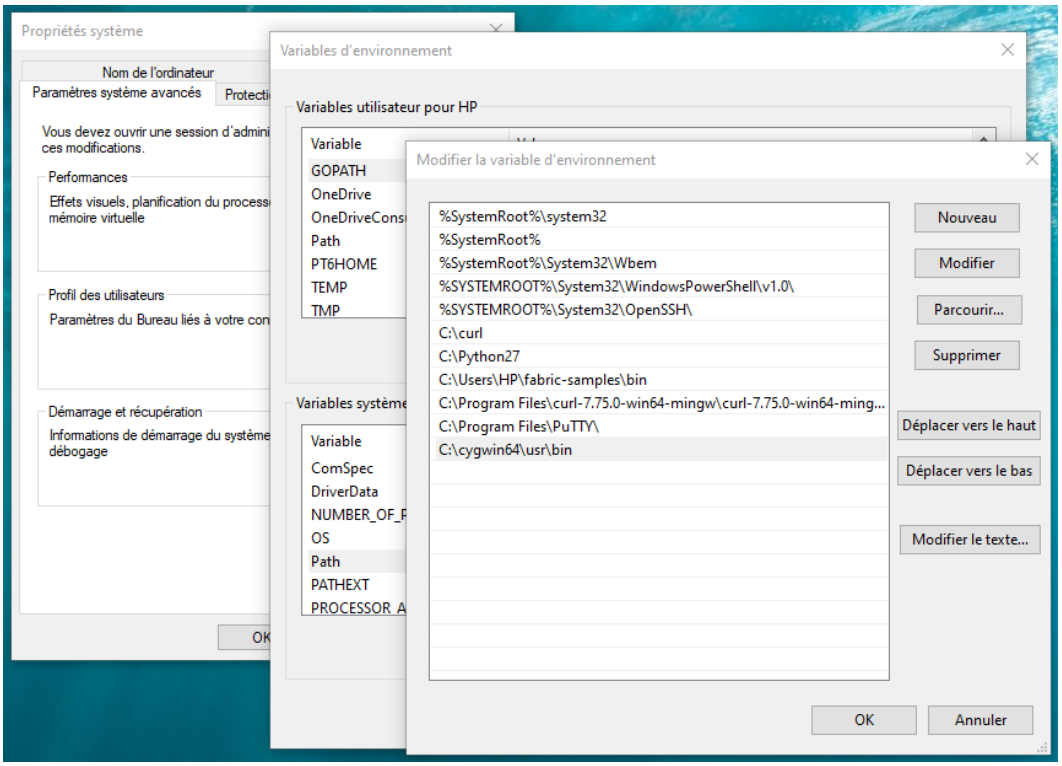

Figure B.13 : Modification de la variable d'environnement Path

Annexe C: Test du compilateur C++ de GCC

Le fichier hello.c est un programme qui affiche 'Hello World', on va se servir de ce fichier pour tester le compilateur c++

Etape1 : on lance la compilation avec la commande *gcc hello.c* ou bien *gcc –o hello.exe hello.c*

Etape2 : on lance l'exécution avec la commande *./a* ou bien *./hello*

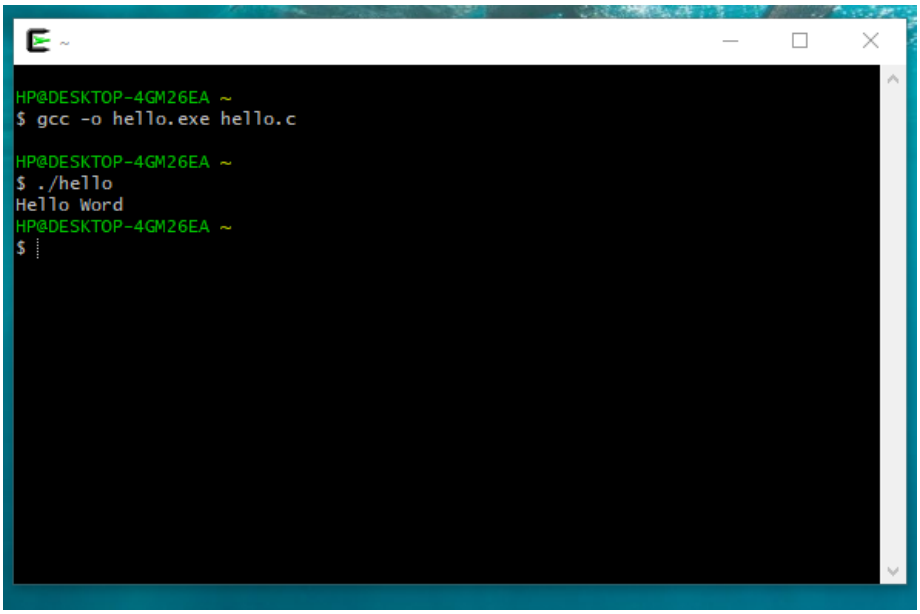

Figure C.1 : compilation, exécution et affichage du résultat

CONCLUSION GENERALE ET PERSPECTIVES

Ce projet de fin d'études a été une opportunité inestimable pour mettre en pratique les connaissances théoriques et techniques et aussi de faire connaissance avec les technologies utilisé pour tester l'efficacité de la parallélisation, notamment MPI et OpenMP.

On a eu l'occasion, à travers ce projet, de voir de près les différents algorithmes de classification dans le contexte de l'intelligence artificielle, d'établir un état d'art et une étude comparative entre ces algorithmes.

On a eu l'occasion aussi de prendre en main un de ses algorithmes et essayer d'améliorer son temps d'exécution dans un contexte donnée et tester la parallélisation via OpenMP.

Par ailleurs, ce projet a également permis de découvrir le monde de la recherche scientifique à travers les revues et les articles consultés pour élaborer ce présent document. Sans oublier la démarche scientifique suivie par des chercheurs pour apporter leur contribution face à une problématique donnée.

Dorénavant, plusieurs perspectives s'ouvrent à nous notamment la possibilité de continuer une thèse doctorale dont le sujet serait en relation à la thématique abordée pendant la réalisation de notre projet.

Références

[1] Rachel Wolff; 5 Types of Classification Algorithms in Machine Learning; 26/08/2020
https://monkeylearn.com/blog/classification-algorithms/ (Consulté le 05/04/2021)

[2]Rohit Garg; 7 Types of Classification Algorithms; 19/01/2018
https://analyticsindiamag.com/7-types-classification-algorithms/ (Consulté le 05/04/2021)

[3] Rohit Garg; Classification; 06/01/2018
https://github.com/f2005636/Classification (Consulté le 05/04/2021)

[4] Ricco Rakotomalala; Principe de la descente de gradient pour l'apprentissage supervisé Application à la régression linéaire et la régression logistique; Université Lumière Lyon 2
https://eric.univ-lyon2.fr/~ricco/cours/slides/gradient_descent.pdf (Consulté le 02/06/2021)

[5] Abhishek Nair; Qu'est-ce que Cygwin et comment installer sur Windows?; 26/04/2021
https://geekflare.com/fr/cygwin-installation-guide/ (Consulté le 17/09/2021)

[6] Wikipédia; Message Passing Interface
https://fr.wikipedia.org/wiki/Message_Passing_Interface (Consulté le 18/09/2021)

[7] Wikipédia; OpenMP
https://fr.wikipedia.org/wiki/OpenMP (Consulté le 18/09/2021)

[8] Thanh-Nghi Do; Parallel multiclass stochastic gradient descent algorithms for classifying million images with very-high-dimensional signatures into thousands classes; 21/01/2014

[10] Lesia Mochurad; Optimization of Regression Analysis by Conducting Parallel Calculations; 22–23/04/2021

[11] Martin A. Zinkevich; Markus Weimer; Alex Smola; Lihong Li; Parallelized Stochastic Gradient Descent

[12] Biplab Kumar Pradhan; Siddhant Panda; Parallel Stochastic Gradient Descent;

[13] Mohsine Eleuldj, HPC et architectures parallèles, EMI, 17/09/2019

[14] https://www.kaggle.com/uciml/pima-indians-diabetes-database

[15] https://datascientest.com/regression-logistique-quest-ce-que-cest