الجمهورية الجزائرية الديمقراطية الشعبية

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

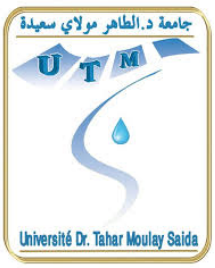
وزارة التعليم العالي و البحث العلمي

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Dr. Tahar Moulay SAIDA — **جامعة د. الطاهر مولاي سعيدة**

Faculté : Technologie — **كلية : التكنولوجيا**

Département : Informatique — **قسم : الإعلام الآلي**



MEMOIRE DE MASTER

Option : Réseaux Informatiques et Systèmes Répartis (RISR)

THEME

Implémentation de l'algorithme KNN sous GPU (application à la détection d'intrusion)

Présenté par :
Djebbar Abderrahmane
Souar Zakaria Abderrahmane

Encadré par :
Khobzaoui Abdelkader

2015 - 2016

Remerciements

Nous remercions en premier notre grand Dieu pour nous avoir donné le courage et la volonté durant les moments difficiles.

Au terme de ce travail, on tient à exprimer notre profonde gratitude et nos sincères remerciements aux personnes qui nous ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire ainsi qu'à la réussite de cette formidable année universitaire et durant toutes les années de notre étude.

Nos profonds remerciements vont à notre encadreur Khobzaoui Abdelkader qui a accepté d'encadrer notre travail, Nous exprimons nos gratitudes à tous les consultants et internautes rencontrés lors des efforts effectuées et qui ont accepté de répondre à nos questions avec gentillesse.

Nous tenons également à remercier les membres du Jury. Nos plus vifs remerciements s'adressent aussi à tout le cadre professoral et administratif de Faculté de technologie de l'université de Dr. Moulay Taher, Saida. Et bien sûr nous gardons une place toute particulière à nos parents, nos frères et nos sœurs, nos amis qui sont toujours à nos côtés.

Nos remerciements vont enfin à toute personne qui a contribué de près ou de loin à l'élaboration de ce travail.

Dédicace

Merci Allah (notre dieu) de nous avoir donné la capacité d'écrire et de réfléchir, la force d'y croire, la patience d'aller jusqu'au bout du rêve et le bonheur de lever nos mains vers le ciel et de dire « Ya Kayoum ».
Je dédie ce modeste travail à nos parent, à nos frères et Sœurs, à La Famille Djebbar et La famille Souar, à nos amis, à tous l'ensemble des étudiants du département d'informatique et à tous ceux qui nous aiment.

Djebbar Abderrahmane
Souar Zakaria Abderrahmane

Table des matières

Table des Figure

Table des Tableaux

Introduction générale

L'apprentissage automatique n'est pas une discipline nouvelle. Mais elle prend tout son sens avec l'arrivée des Big Data. Le processus d'apprentissage automatique consiste à créer un modèle, grâce à des exemples initialement disponibles, permettant d'« identifier » des objets nouveaux. Il existe de nombreux types de problèmes en machine learning : classification, régression, détection d'anomalies, moteurs de recommandation, etc. Bien que les techniques d'apprentissage automatique aient déjà été utilisées depuis plusieurs décennies, deux tendances récentes ont contribué à populariser leur adoption : la disponibilité progressive de gros volumes de données à étudier, et l'incroyable puissance de traitement parallèle offerte par le calcul par le GPU. Les GPU sont aujourd'hui massivement utilisés pour « apprendre » aux réseaux de neurones artificiels, par exemple, à analyser des volumes de données toujours plus importants en un laps de temps toujours plus faible, sans compter qu'ils mobilisent des infrastructures matérielles moins importantes que les configurations traditionnelles. Les GPU sont également utilisés pour exécuter des modèles complexes d'apprentissage automatique dans des domaines comme la classification et la prédiction de données dans le Cloud, car ils permettent de traiter rapidement de gros volumes d'informations avec une infrastructure matérielle et un coût énergétique plus faibles. ***Professeur Ian Lane de l' université Carnegie-Mellon*** affirme qu'il est possible exécuter algorithmes de reconnaissance jusqu'à 33 fois plus vite qu'avec une simple configuration CPU. Dans le cadre de ce mémoire, nous tentons d'explorer ce monde « merveilleux » des GPU par la parallélisassions d'un algorithme très basique de classification en prenant comme champ d'application pour cet algorithme la détection d'intrusion qui est vue ou traitée dans la littérature spécialisée comme étant un pure problème de classification.

Le présent document sera structuré comme suit :

Dans un premier chapitre, nous reviendrons sur les principes généraux de l'apprentissage automatique en mettons l'accent plus particulièrement sur le problème de classification supervisée. Le deuxième chapitre sera consacré à un survol de la détection d'intrusion dans lequel nous présenterons les différentes approches de détection d'intrusion, les différentes architectures des systèmes de détection d'intrusion

ainsi que les limites et perspectives de ce moyen de lutter contre les attaques électroniques qui cibles les cypber-infrastructures. Le troisième chapitre quant à lui, il sera consacré à l'introduction de la technologie des GPU. Dans le quatrième chapitre nous présenterons les différents algorithmes et outils de développement utilisés dans le cadre de cette étude et bien sur les résultats commentés des différents tests effectués. Une conclusion générale, clôtura ce mémoire comme il est d'usage dans ce type de documents.

.

Chapitre I

L’apprentissage Automatique

Introduction

L'**apprentissage automatique (ou artificiel)** (*machine-learning* en anglais) est un des champs d'étude de l'intelligence artificielle. Il fait référence à la capacité d'un système à acquérir et intégrer de façon autonome des connaissances. Arthur Samuel le définit comme étant un "champ d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés". Cette notion englobe toute méthode permettant de construire un modèle de la réalité à partir de données, soit en améliorant un modèle partiel ou moins général, soit en créant complètement le modèle [8]. Ainsi l'intérêt de l'apprentissage automatique el le développement, l'analyse et l'implémentation de méthodes qui permettent à une machine (au sens large) d'évoluer et de remplir des tâches associées à une intelligence artificielle grâce à un processus d'apprentissage. Cet apprentissage permet d'avoir un système qui s'optimise en fonction de l'environnement, les expériences et les résultats observés [8].

Les algorithmes d'apprentissage automatique ont été appliqués à divers domaines, notamment le traitement du langage naturel et de la parole, la reconnaissance de l'écriture manuscrite, la vision robotisée, la fouille de données, les moteurs de recherche sur Internet, le diagnostic médical, la bioinformatique, etc. Globalement ces algorithmes sont répartis en plusieurs classes de méthodes dont les plus basiques sont :

– la **Classification**: modélisation de plusieurs groupes de données dans des classes existantes. Par exemple: la classification des orchidées, la tendance d'un parti politique…

– le **Clustering**: ressemble à la classification mais ce ne sont pas des classes connues.

– la **Régression**: les données sont liées à d'autres données numériques par une corrélation (une droite, une courbe, une tendance).

Dans le cadre du présent mémoire nous nous intéressons exclusivement au problème de classification.

- **Quelques mots d'Histoire...**

L'apprentissage artificiel est une discipline jeune, à l'instar de l'Informatique et de l'Intelligence artificielle. Il se situe au carrefour d'autres disciplines: philosophie, psychologie, biologie, logique, mathématique [8]. les premières études remontent à des travaux de statistique dans les années 1920. C'est après la seconde guerre mondiale que les premières expériences deviennent possibles. Se développent ensuite dans les années 1960 les approches connexionnistes avec des perceptrons, et la reconnaissance des formes. La mise en évidence des limites du perceptron simple arrête toutes les recherches dans ce domaine jusqu'à la renaissance dans les années 1980. Les années 1970 sont dominées par des systèmes mettant l'accent sur les connaissances, les systèmes experts, Les limites de tels systèmes se font sentir dans les années 1980, pendant lesquelles a lieu le retour du connexionnisme avec un nouvel algorithme d'apprentissage [8].

Les mathématiciens commencèrent à s'éloigner du cadre cognitif de l'apprentissage pour envisager le problème sous l'angle de l'optimisation, pendant qu'apparaissaient de nouvelles méthodes comme les arbres de décision ou l'induction de programmes logiques. L'influence de la théorie statistique de l'apprentissage s'est réellement fait sentir dans les années 1990, avec l'ouvrage de Vapnik [21].

1. Concepts et Sources de l'apprentissage automatique

L'apprentissage de l'être humain se compose de plusieurs processus qu'il est difficile précisément à décrire. Les facultés d'apprentissage chez l'humain lui ont conféré un avantage évolutif déterminant pour son développement [8].

Par " faculté d'apprendre " on entend un ensemble d'aptitudes comme :

- ✓ L'obtention de la capacité de parler en observant les autres.
- ✓ L'obtention de la capacité de lire, d'écrire, d'effectuer des opérations arithmétiques et logiques avec l'aide d'un tuteur.
- ✓ L'obtention d'habilités motrices et sportives en s'exerçant.

1.1. Qu’est-ce que l’apprentissage automatique

La faculté d’apprendre de ses expériences passées et de s’adapter est une caractéristique essentielle des formes de vies supérieures. Elle est essentielle à l’être humain dans les premières étapes de la vie pour apprendre des choses aussi fondamentales que reconnaître une voix, un visage familier, apprendre à comprendre ce qui est dit, à marcher et à parler [22].

L’apprentissage automatique est une tentative de comprendre et reproduire cette faculté d’apprentissage dans des systèmes artificiels. Il s’agit, très schématiquement, de concevoir des algorithmes capables, à partir d’un nombre important d’exemples (les *données* correspondant à “l’expérience passée”), d’en assimiler la nature afin de pouvoir appliquer ce qu’ils ont ainsi appris aux cas futurs [22].

1.2. Définition

Un programme d'ordinateur est capable d'apprendre à partir d'une expérience E et par rapport à un ensemble T de tâches et selon une mesure de performance P, si sa performance à effectuer une tâche de T, mesurée par P, s'améliore avec l'expérience E [8].

1.3. Modélisation

L’apprentissage automatique d’une machine toujours concerne un ensemble de tâches concrètes - T. Pour déterminer la performance de la machine, on utilise une mesure de la performance P. La machine peut avoir à l’avance un ensemble d’expérience E ou elle va enrichir cet ensemble plus tard.

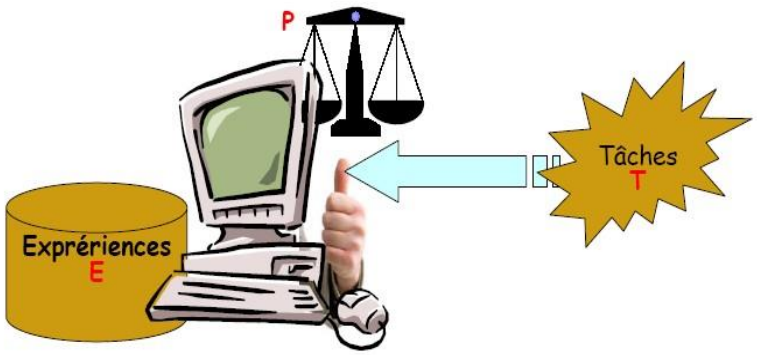


Figure 0-1 Schéma de modélisation d’une machine d’apprentissage

Donc, l’apprentissage automatique pour la machine est qu’avec l’ensemble de tâches T que la machine doit réaliser, elle utilise l’ensemble d’expériences E telle que sa performance sur T est améliorée.

1.4. Domaines d’applications de l'apprentissage automatique:

L'apprentissage automatique s'applique à un grand nombre d'activités humaines et convient en particulier au problème de la prise de décision automatisée. Il s’agira, par exemple:

- D'établir un diagnostic médical à partir de la description clinique d'un patient;
- De donner une réponse à la demande de prêt bancaire de la part d'un client sur la base de sa situation personnelle;
- De déclencher un processus d'alerte en fonction de signaux reçus par des capteurs ;
- De la reconnaissance des formes;
- De la reconnaissance de la parole et du texte écrit;
- De contrôler un processus et de diagnostiquer des pannes;

1.5. Mémoriser n’est pas généraliser

Le terme **apprentissage** dans la langue courante est ambigu. Il désigne aussi bien l’apprentissage “par cœur” d’une poésie, que l’apprentissage d’une tâche complexe telle que la lecture. Clarifions la distinction [22]:

- Le premier type d’apprentissage correspond à une simple mémorisation. Or les ordinateurs contemporains, avec leurs mémoires de masse colossales, n’ont aucune difficulté à mémoriser une encyclopédie entière, sons et images inclus.
- Le second type d’apprentissage se distingue fondamentalement du premier en cela qu’il fait largement appel à notre faculté de généraliser. Ainsi pour apprendre à lire, on doit être capable d’identifier un mot écrit d’une manière que l’on n’a encore jamais vue auparavant.

2. Types d'apprentissage

Les algorithmes d'apprentissage peuvent se catégoriser selon le mode d'apprentissage qu'ils emploient :

2.1. L'apprentissage supervisé

Si les classes sont prédéterminées et les *exemples* connus, le système apprend à classer selon un modèle de classement ; on parle alors d'apprentissage supervisé (ou d'analyse discriminante).

Un expert (ou oracle) doit préalablement correctement étiqueter des exemples. L'apprenant peut alors trouver ou approximer la fonction qui permet d'affecter la bonne « étiquette » à ces exemples. Parfois il est préférable d'associer une donnée non pas à une classe unique, mais une probabilité d'appartenance à chacune des classes prédéterminées (on parle alors d'apprentissage supervisé probabiliste). L'analyse discriminante linéaire ou les SVM sont des exemples typiques. Autre exemple : en fonction de *points communs* détectés avec les symptômes d'autres patients connus (les « exemples »), le système peut catégoriser de nouveaux patients au vu de leurs analyses médicales en risque estimé (probabilité) de développer telle ou telle maladie.

2.2. L'apprentissage non-supervisé

Quand le système ou l'opérateur ne dispose que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminés, on parle d'apprentissage non supervisé (ou clustering). Aucun expert n'est disponible ni requis. L'algorithme doit découvrir par lui-même la structure plus ou moins cachée des données.

Le système doit ici dans l'espace de description (la somme des données) cibler les données selon leurs attributs disponibles, pour les classer en groupe homogènes d'exemples. La similarité est généralement calculée selon la fonction de distance entre paires d'exemples. C'est ensuite à l'opérateur d'associer ou déduire du sens pour chaque groupe. Divers outils mathématiques et logiciels peuvent l'aider. On parle aussi d'analyse des données en régression. Si l'approche est probabiliste (c'est à dire que chaque exemple au lieu d'être classé dans une seule classe est associé aux probabilités d'appartenir à chacune des classes), on parle alors de *« soft clustering »* (par opposition au *« hard clustering »*).

Exemple : Un épidémiologiste pourrait par exemple dans un ensemble assez large de victimes de cancers du foie tenter de faire émerger des hypothèses explicatives, l'ordinateur pourrait différencier différents groupes, qu'on pourrait ensuite associer par exemple à leur provenance géographique, génétique, à l'alcoolisme ou à l'exposition à un métal lourd ou à une toxine telle que l'aflatoxine.

2.3. L'apprentissage semi-supervisé

Effectué de manière probabiliste ou non, il vise à faire apparaitre la distribution sous- jacente des *« exemples »* dans leur espace de description. Il est mis en œuvre quand des données (ou *« étiquettes »*) manquent... Le modèle doit utiliser des exemples *non-étiquetés* pouvant néanmoins renseigner.

Exemple : En médecine, il peut constituer une aide au diagnostic ou au choix des moyens les moins onéreux de tests de diagnostics.

2.4. L'apprentissage partiellement supervisé (probabiliste ou non)

Quand l'étiquetage des données est partiel. C'est le cas quand un modèle énonce qu'une donnée n'appartient pas à une classe *A*, mais peut-être à une classe *B* ou *C* (*A, B* et *C* étant 3 maladies par exemple évoquées dans le cadre d'un diagnostic

différentiel).

2.5. L'apprentissage par renforcement

L’algorithme apprend un comportement étant donné une observation. L'action de l'algorithme sur l'environnement produit une valeur de retour qui guide l'algorithme.

3. Les algorithmes utilisés

- Les machines à vecteurs support
- Le boosting
- Les réseaux de neurones pour un apprentissage supervisé ou non-supervisé
- La méthode des k plus proches voisins pour un apprentissage supervisé
- Les arbres de décision
- Les méthodes statistiques comme par exemple le modèle de mixture gaussienne
- La régression logistique
- L'analyse discriminante linéaire
- La logique floue
- Les algorithmes génétiques et la programmation génétique

Ces méthodes sont souvent combinées pour obtenir diverses variantes d'apprentissage. L'utilisation de tel ou tel algorithme dépend fortement de la tâche à résoudre (classification, estimation de valeurs, etc.).

Facteurs de pertinence et d'efficacité

La qualité de l'apprentissage et de l'analyse dépendent du besoin en amont et *a priori* compétence de l'opérateur pour préparer l'analyse. Elle dépend aussi de la complexité du modèle (spécifique ou généraliste) et de son adaptation au sujet à traiter. Enfin, la qualité du travail dépendra aussi du mode (de mise en évidence visuelle) des résultats pour l'utilisateur final (un résultat pertinent pourrait être caché dans un schéma trop complexe, ou mal mis en évidence par une représentation graphique inappropriée). Avant cela, la qualité du travail dépendra de facteurs

initiaux contraignants, liées à la base de données :

1) **Nombre d'exemples *:*** moins il y en a plus l'analyse est difficile, mais plus il y en a plus le besoin de mémoire informatique est élevé et plus longue est l'analyse.

2) **Nombre et qualité des attributs :** décrivant ces exemples (La distance entre deux "exemples" numériques (prix, taille, poids, intensité lumineuse, intensité de bruit, etc) est facile à établir, celle entre deux attributs catégoriels (couleur, utilité, est plus délicate)

3) **Pourcentage de données renseignées** et manquantes

4) **Bruit :** Le nombre et la « *localisation* » des valeurs douteuses (erreurs) ou naturellement non conformes au modèle de distribution générale des « *exemples* » sur leur espace de distribution.

La classification en théorie

3.1 Modèle général

Le modèle général du problème d'apprentissage à partir d'un échantillon d'observations est composé de trois parties :

- **Un environnement** : il engendre des formes X_i tirées indépendamment et de manière identiquement distribuée selon la loi de probabilité D_i fixe mais inconnue d'un espace d'entrée X.

- **Un oracle ou superviseur** : il retourne pour chaque forme X_i une réponse désirée ou étiquette (label) U_i.

- **Un apprenant** : Il est capable de réaliser une fonction h d'un espace d'hypothèses H qui vérifie $y_i = h(x_i)$ ou y_i est sa sortie.

  La recherche de la fonction désirée dans F est basée sur un échantillon d'apprentissage

  S_m tel que $S_m = \{ x_1, u_1 , \ldots \ldots , x_m , u_m \}$ La figure 1.2 illustre ces trois modules.

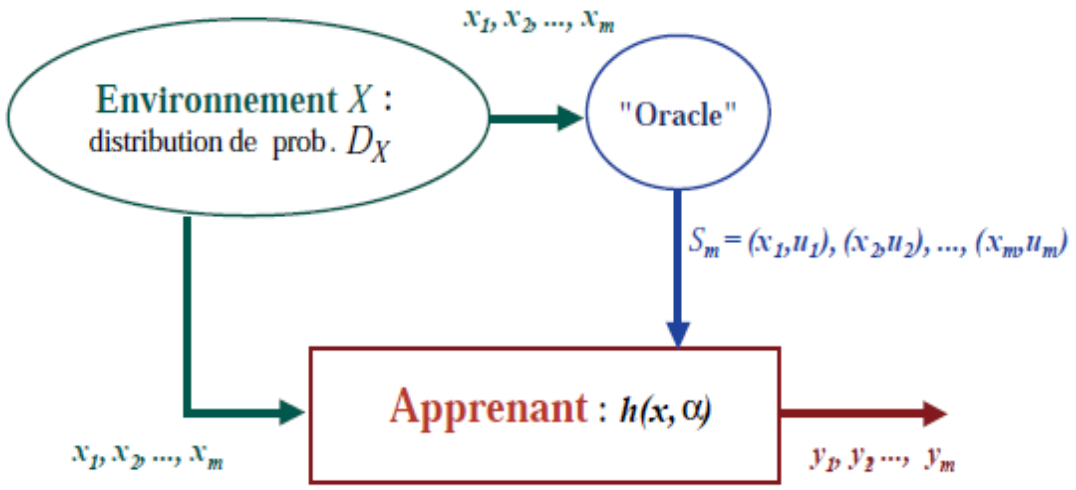


Figure 1-2 Les modèles d'un système d'apprentissage [8].

3.2 Formulation

La tâche qu'un classificateur doit effectuer peut être exprimée par une fonction que l'on appelle fonction de décision [6][11]

$$f : x \longrightarrow u$$

Où X est l'ensemble des objets à classer (aussi appelé espace d'entrée), u est l'ensemble des catégories (aussi appelé espace d'arrivée)

Dans la suite de ce chapitre, nous nous limiterons à la classification binaire. Dans ce cas, l'ensemble u correspond à {+1, -1} La plupart du temps on interprétera +1 et -1 respectivement comme l'appartenance et la non-appartenance à une catégorie déterminée.

Une grande partie de la littérature se focalise sur le cas binaire parce que les classifications faisant intervenir plus de 2 catégories (cas multi-classes) peuvent toujours être ré exprimées sous forme de classifications binaires [6].

En effet, si l'on considère une classification multi-classes, nous pouvons effectuer, pour chaque catégorie, une classification binaire indépendante et regarder ensuite pour quelle catégorie l'appartenance s'est manifestée [6].

Jusqu'a présent nous n'avons fait que définir le domaine et l'image de la fonction à estimer. On ne peut bien entendu pas accepter n'importe quelle fonction de cette forme. Nous devons en quelque sorte imposer que la fonction représente bien la relation entre les objets et leurs catégories. Pour ce faire nous avons besoin de modéliser le processus selon lequel les données sont générées et d'introduire une

fonction de coût indiquant à quel point notre fonction f s'écarte de ce processus.

3.3 Fonction d'erreur et risque

Si nous disposons de m exemples bien classés x_1, u_1 , ... , (x_m, u_m), une première approche pour déterminer les performances d'un classificateur est de comparer ses prédictions avec les classes u_i attendues. A cette fin, on introduit une fonction d'erreur.

- **Définition (fonction d'erreur)**

Soit le triplet $(x, u, f(x)) \in x \times u \times u$ ou x est un objet, u sa catégorie et f(x) la prédiction du classificateur. Toute fonction C : $x \times x \times x \rightarrow [0, \infty]$ telle que C(x, u, u) = 0 est appelée fonction d'erreur [6].

Par la suite nous utiliserons une fonction d'erreur simple et bien adaptée à la classification binaire

$$E(x, u, f(x)) = \frac{1}{2}|f(x) - u| \qquad (1.1)$$

Nous introduisons à présent la notion de risque qui représente l'erreur moyenne commise sur toute la distribution P(x; u) par la fonction f(x) [6]:

$$R[f] = \int \frac{1}{2}|f(x) - u| dp(x, u) \qquad (1.2)$$

Nous sommes à présent en mesure d'introduire un critère selon lequel la fonction f devra être optimale :

$$f^{opt} = arg_f \min(R[f]) \qquad (1.3)$$

En d'autres termes f^{opt} devra être telle que l'erreur moyenne sur toute la distribution soit minimale.

3.4 Machine d'apprentissage

On désigne par machine d'apprentissage, une machine dont la tâche est d'apprendre une fonction au travers d'exemples. Une machine d'apprentissage est donc définie par la classe de fonctions F qu'elle peut implémenter [6]. Dans notre

cas, ces fonctions sont des fonctions de décision.

3.5 Risque empirique

Si on dispose d'une machine d'apprentissage et d'un ensemble de m exemples, on peut exprimer le risque empirique d'une fonction f_γ:

$$R_{emp}[f_\gamma] = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}|f_\gamma(x_i) - u_i| \qquad (1.4)$$

De manière similaire à la section précédente, on peut dériver un critère de sélection de la fonction optimale f^n :

$$f^n = f_{\gamma^*} \; telle \; que \; \gamma^* = arg_\gamma \min(R_{emp}[f_\gamma] \qquad (1.5)$$

Ce critère est appelé minimisation du risque empirique (MRE) [6].

3.5.1 Analyse statistique de l'apprentissage

3.5.1.1 Insuffisance de minimisation du risque empirique (MRE)

On peut légitimement se demander si le critère de minimisation du risque empirique mène toujours à un classificateur possédant un bon pouvoir de généralisation.

Si on n'impose aucune restriction sur la classe *F* on peut toujours trouver une fonction *f* qui se comporte bien vis à vis du risque empirique, sans pour autant assurer une bonne généralisation sur de nouveaux exemples.

La restriction des fonctions implémentables nous confronte à un dilemme que les statisticiens appellent biais-variance [18].

Dans la terminologie apprentissage automatique, on l'appelle sur généralisation, et apprentissage par cœur, concrètement cela dit deux choses :

- D'une part, en limitant fortement la taille de l'ensemble *F*, on tend à expliquer la relation entre les objets et leurs classes de manière trop grossière (sur généralisation). Dans ce cas le risque empirique sera élevé mais le modèle ne collera pas aux exemples de trop près (voire figure 1.3 (b)).

- D'autre part, si on admet un grand nombre de fonctions, la relation sera modélisée de manière complexe et le bruit associé aux mesures risque également d'être appris. On parle souvent d'apprentissage par cœur parce que le classificateur aura un risque empirique très faible mais ses performances sur d'autres jeux de données seront mauvaises (voire figure 1.3 (a)).

On voit que la “taille” ou “complexité” de l’ensemble de fonctions F joue un rôle fondamental. Ce que l’on nomme la *capacité h* (*F)* est une mesure de cette complexité.

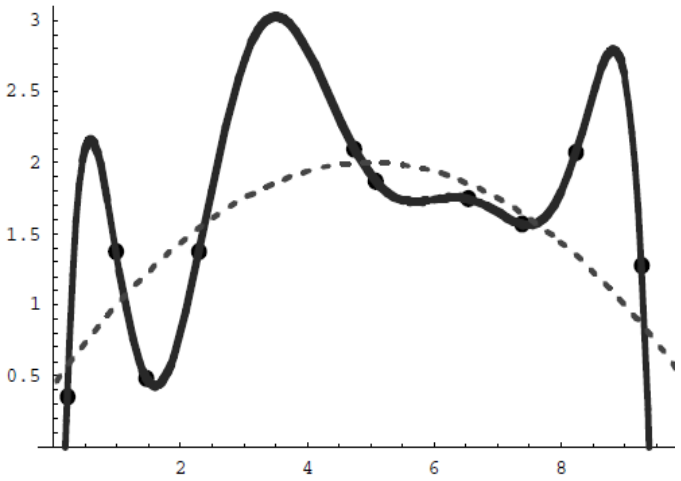


Figure 1-3 (a) : l'apprentissage par cœur (overfitting)

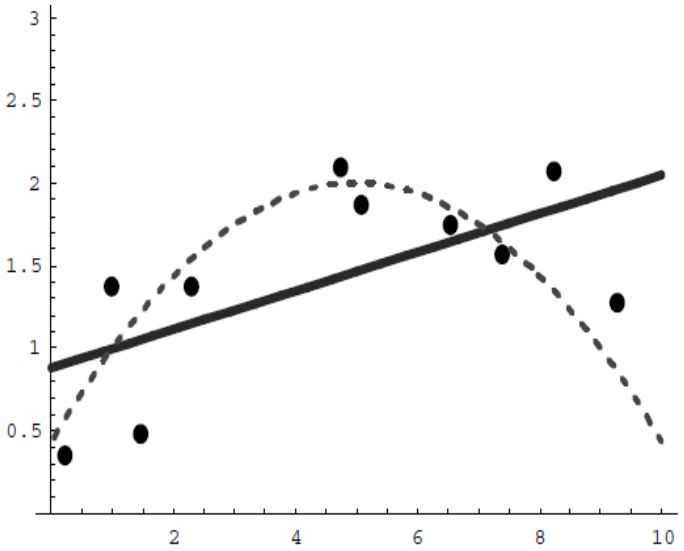


Figure 1-4 (b) : sur apprentissage (underfitting)

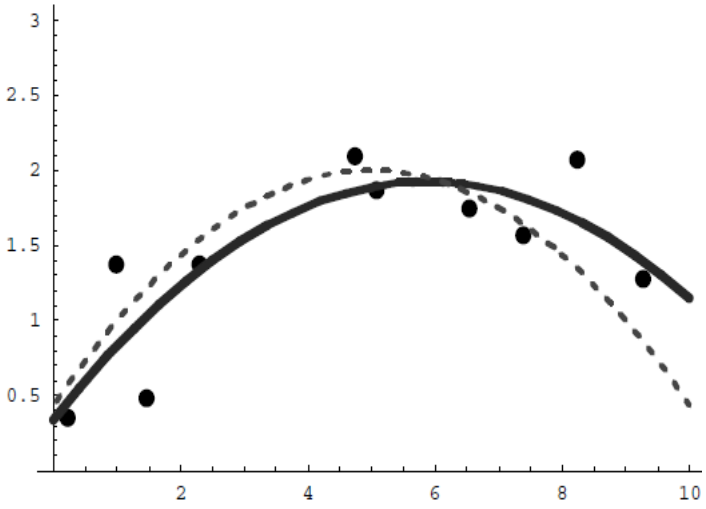


Figure 1-5 (c) : compromis entre a et b (bon modèle)

3.1.1.1. **Dimension VC [6]**

La dimension VC, notée h d'un ensemble de fonctions F est le nombre maximum de points pouvant être séparés de toutes les manières possibles par les fonctions de F. Cela veut dire qu'il doit exister une configuration de h (= VC (F)) points, telle que les fonctions $f \in F$ peuvent leur assigner les 2^h combinaisons de labels (classes) possibles. Cela n'est pas garanti pour tout ensemble arbitraire de h points.

Pour illustrer cette idée, considérons trois points représentés dans R^2 . Supposons que la famille de fonctions f corresponde aux droites de $R^2 : y = y_1x + y_0$

La dimension VC de F est de 3 car on peut trouver une configuration de trois points séparables de toutes les façons possibles.

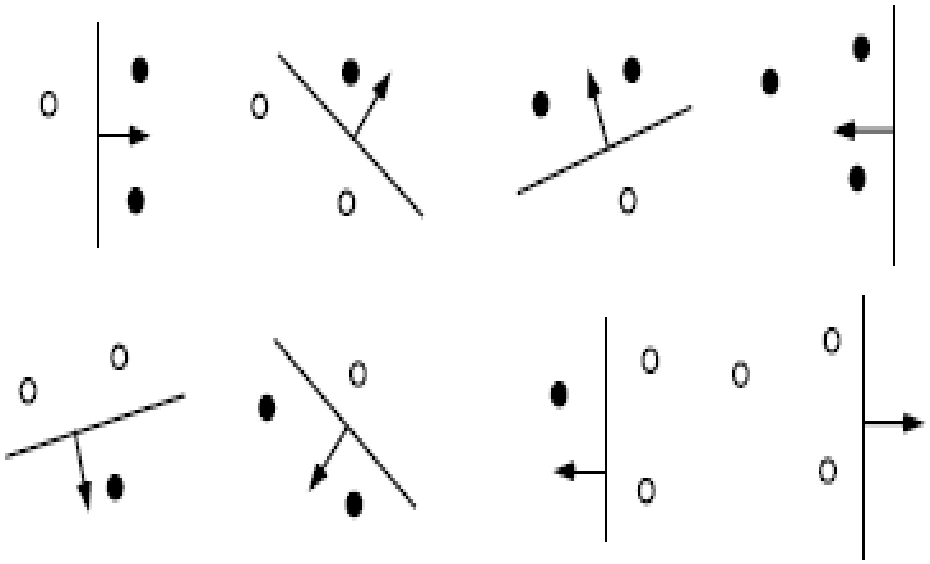


Figure 0-6 trois points de R^2 et leurs 23=8 configurations possibles .ces trois points sont éclatés par la famille des hyperplans orientés de R^2 [6]

Les ronds vides et pleins représentent respectivement les points assignés positivement et négativement. La flèche représente le coté de la droite où les points seront classés positivement.

3.1.1.2. **Minimisation du risque structurel**

Soit F une famille de fonctions de classificateurs telle que ses éléments sont indexés par un ensemble Λ, c'est-à-dire:

$$F = \{f_\alpha : f_\alpha : R^d \to \{-1, +1\}, \alpha \in \Lambda\}$$

D'après Vapnik [21], pour toute famille F, la majoration du risque en généralisation d'un classificateur $f_\alpha \in$ F appris sur un ensemble S de taille m vérifie :

$$R(\alpha)f(x) \leq R_{emp}(\alpha) + \sqrt{\frac{1}{m}\left(h\left(\ln\frac{2m}{h} + 1\right) - \ln\frac{n}{4}\right)} \qquad (1.6)$$

Avec la probabilité 1-η, *h* est la dimension VC de F.

On voit qu'il est difficile de généraliser à partir d'un échantillon de petite taille : plus m est petit, plus la différence entre le risque réel *R* et le risque empirique R_{emp} est susceptible d'être élevée. De même, un classificateur choisi parmi une classe très large de fonctions (*h* élevé) peut ``apprendre par cœur" et donner lieu à un risque plus élevé que celui mesuré [6].

L'inégalité (1.6) est particulièrement intéressante pour justifier le choix d'une machine d'apprentissage qui possède une bonne capacité de généralisation. Une bonne généralisation est obtenue quand *R*(*α*) est petit. Le deuxième terme de droite est appelé terme de confiance de Vapnik (la VC-confiance) [6].

3.1.1.3. **Stratégies de minimisation du risque**

Regardons à présent comment utiliser une borne du type (1.6) pour entraîner une machine d'apprentissage. Premièrement remarquons que dans ce type de borne, le risque empirique concerne une seule fonction alors que le terme de confiance est relatif à un ensemble de fonctions.

L'idée de base va être de fixer une série de sous-ensembles de F, possédant chacun un terme de capacité propre, et ensuite d'effectuer un entraînement par MRE sur chacun d'entre eux. Concrètement, nous allons diviser l'ensemble F de la machine d'apprentissage en une structure qui consiste en l'imbrication successive de sous-ensembles dont la dimension VC va en augmentant (voir figure 1.5).

Une fois les entraînements sur chaque sous-ensemble terminés, nous allons déterminer le modèle optimal, en regardant quelle est la configuration qui

minimise (1.6). On parle de sélection de modèle.

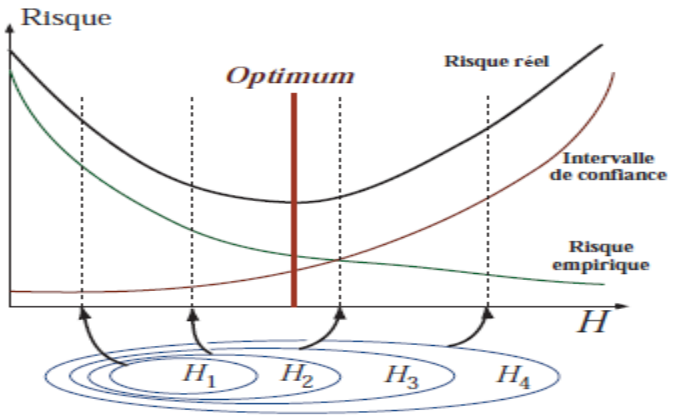


Figure 1-7 Comportement du risque empirique, l'intervalle de confiance et le risque garanti en fonction de la VC dimension [8]

3.1.2. La classification en pratique

La machine d'apprentissage, dont la tâche est d'apprendre une fonction à travers des exemples (dans notre cas c'est la fonction de décision) ne nous indique nullement comment obtenir un classificateur adapté à la tâche considérée.

Regardons à présent les différentes étapes que l'approche Machine Learning préconise pour atteindre un tel objectif.

3.1.2.1. Ensemble de données

Lorsque l'on construit un modèle afin qu'il minimise le risque empirique calculé sur les données d'apprentissage, l'erreur d'apprentissage ainsi obtenue ne peut être considérée comme une bonne mesure de l'erreur de généralisation : elle est évidemment biaisée. Pour obtenir un estimé non biaisé de l'erreur de généralisation, il est crucial de mesurer l'erreur sur des exemples qui n'ont pas servi à entraîner le modèle. Pour cela, on divise l'ensemble des données disponibles en deux parties :

- Un sous ensemble d'entraînement, dont les données serviront à l'apprentissage(ou entraînement) du modèle ;
- Un sous ensemble de test, dont les données seront utilisées uniquement pour évaluer la performance du modèle entraîné. Ce sont les données

"hors-échantillon d'entraînement". On obtient ainsi l'erreur de test qui est un estimé bruiter, mais non biaisé de l'erreur de généralisation.

Par ailleurs l'ensemble d'entraînement est souvent lui-même partagé entre un sous- ensemble qui sert à apprendre les paramètres d'un modèle à proprement dit, et un autre sous- ensemble dit "de validation", qui sert à la sélection de modèle.

Les différentes opérations que l'on doit effectuer avant de présenter les données à l'algorithme d'apprentissage sont :

a. Acquisition des données

Si les données proviennent d'une source analogique, il faut commencer par les transformer de manière à en avoir une représentation manipulable par un programme informatique.

b. Prétraitement

Les bons résultats qu'un classificateur automatique peut fournir reposent, en grande partie, sur la phase de prétraitement. Les données issues d'un mauvais prétraitement vont mettre en péril la qualité du classificateur.

Cette phase consiste en une succession de traitements sur les données brutes afin d'extraire de l'information et de ne garder que celle qui est utile à la classification. L'application de la phase de prétraitements sur les données brutes peut voir deux apports :

- Réduire la taille de l'information qui va être présentée au classificateur, ce qui se traduit par un gain en temps.
- Éliminer l'information non pertinente qui peut être une source de confusion pour le classificateur.

Nous allons présenter quelques méthodes utilisées lors de la phase de prétraitements.

A/ Extractions des caractéristiques

Les techniques d'extractions des caractéristiques permettent d'éliminer l'information redondante et de ne garder que l'information pertinente. On peut, par exemple, utiliser les techniques de réduction telle que la décomposition en ondelettes et l'analyse en composantes principales (ACP) pour compresser les données tout en préservant au mieux l'information utile.

B/ Sélections des caractéristiques

Il arrive souvent que même après l'application d'une technique de compression de données, le nombre de caractéristiques reste élevé pour servir d'information d'entrée d'un classificateur. En outre, la réduction ne permet pas nécessairement de sélectionner des caractéristiques permettant de bien discriminer entre éléments distincts. Certaines peuvent même être une source de confusion lors de l'affectation des éléments à des classes.

Une caractéristique est pertinente si elle contribue fortement à une bonne différenciation entre deux éléments appartenant à deux ensembles hétérogènes et un bon rapprochement entre les éléments d'un même ensemble.

Les méthodes de sélection des caractéristiques tentent de sélectionner un sous ensemble de caractéristiques adéquates parmi celles obtenues dans l'étape d'extraction. L'objectif de cette sélection est double :

- Surmonter les limitations techniques : aucun classificateur automatique ne peut fonctionner correctement avec plus d'une cinquantaine de caractéristiques.
- Accélérer le processus d'apprentissage : plus le nombre de caractéristiques est petit plus les calculs et l'apprentissage sont rapides.

c. Conversion

Il s'agit de convertir les données dans le format spécifié par l'algorithme utilisé lors de la phase de classification. La représentation vectorielle est assez populaire, notamment parce qu'elle permet de concevoir des algorithmes relativement indépendants du type d'objets à classer. Dans ce format, les données sont représentées sous forme de vecteurs dont chaque composante correspond à une caractéristique de l'objet. La plupart des algorithmes de classifications gèrent difficilement des vecteurs de grande dimension [6].

d. Post-traitement

Dans certains cas, on doit normaliser les données dans le format d'entrée. Par exemple, dans le cas d'un réseau de neurones, on conseille que la moyenne de chaque composante du vecteur sur l'ensemble des exemples d'entrainement soit proche de zéro [6].

e. Entraînement ou apprentissage

La phase d'entraînement consiste à sélectionner une fonction $f \in F_\alpha$ c'est-à-dire à

trouver une évaluation des paramètres α_i . La sélection de ces paramètres est effectuée par un algorithme d'apprentissage qui reçoit en entrée l'ensemble d'apprentissage ainsi qu'un ensemble de paramètres d'apprentissage : p_1 ….. p_k . En ce sens, ce sont les données (de l'ensemble d'apprentissage) qui induisent l'apprentissage. L'ensemble des paramètres α_i résultant de l'apprentissage est appelé modèle.

Une machine d'apprentissage munie d'un modèle est appelée machine entraînée. La figure (1.8) schématise un tel entraînement.

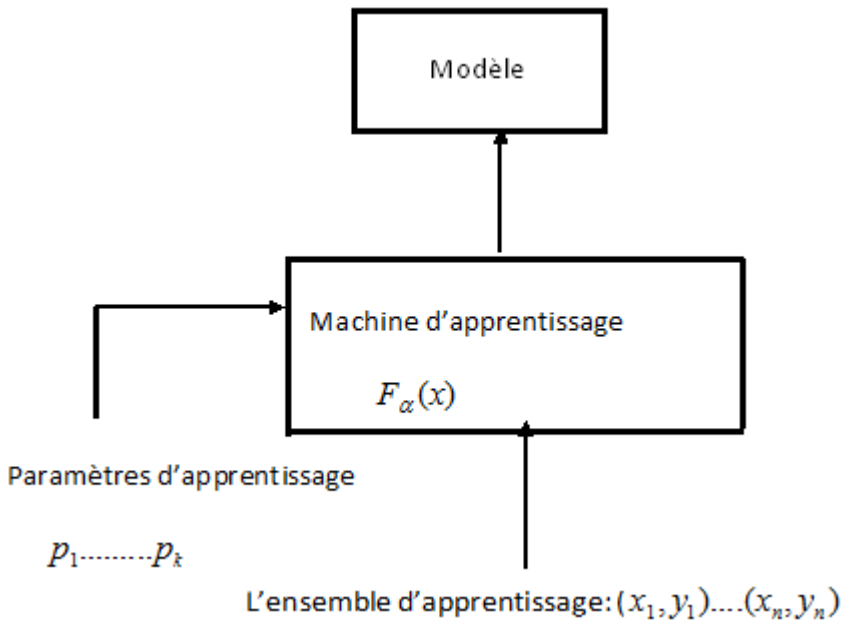


Figure 1-8 Entraînement d'une machine d'apprentissage [6]

La sélection de modèle est ici comprise au sens large : il peut s'agir de choisir le meilleur entre plusieurs familles de modèles très différents, ou entre des variantes très semblables d'un même modèle, dues uniquement à des variations des valeurs d'un hyper- paramètre contrôlant la définition du modèle.

f. Evaluation du modèle (test)

Une fois le modèle obtenu, il est intéressant d'évaluer ses performances sur un ensemble indépendant de données : le test set. En effet, on ne peut se fier aux résultats obtenus sur l'ensemble d'apprentissage car la machine d'apprentissage a perdu son indépendance face à ces données. Cette phase permet de se rendre compte du pouvoir de généralisation du classificateur, c'est à dire sa capacité à obtenir de bons résultats sur n'importe quel ensemble de données provenant de la même distribution.

Dans les cas où l'on dispose de trop peu de données, on peut utiliser une technique de *validation croisée*, pour générer plusieurs paires de sous-ensembles entraînement / test. Cette technique est coûteuse en temps de calcul, mais permet d'obtenir un bon estimé de l'erreur de test, tout en conservant suffisamment d'exemples pour l'entraînement.

- **Validation croisée**

La validation croisée est une technique qui permet de tester un modèle d'apprentissage. La validation croisée se décline en plusieurs sous-méthodes. La plus répandue est la méthode « k-Fold » avec typiquement $k \in [4,10]$. Si l'on a une base d'apprentissage A_p contenant p éléments: $A_p = x_1, \dots, x_p$ la validation croisée consiste à appliquer les cinq étapes suivantes [14]:

1. Découper l'ensemble des exemples en k sous-ensembles disjoints de taille p/k.
2. Apprendre sur les k-1 sous-ensembles.
3. Calculer l'erreur sur la $k^{iéme}$ partie.
4. Réitérer le processus p fois.
5. Obtenir l'erreur finale en calculant la moyenne des k erreurs précédentes.

La validation croisée est simple à mettre en œuvre et utilise toutes les données.

Elle permet d'obtenir une estimation de l'erreur de généralisation. Cela permet d'éviter le sur apprentissage

g. Exploitation

Lorsque l'on dispose d'un modèle efficace pour une tâche considérée, on peut utiliser la machine d'apprentissage pour faire des prédictions sur de nouveaux exemples.

Un classificateur correspond donc à une machine entraînée. Les sociétés qui vendent ce type de classificateurs entraînent souvent la machine d'apprentissage sur un corpus relatif aux catégories spécifiées par le client. De cette manière, le client reçoit uniquement une machine entraînée sans avoir à se soucier de questions d'entraînement et de paramétrage.

Certaines applications grand public telle que la reconnaissance vocale (Babel Speech) ou l'OCR (Read Iris, Omnipage) proposent également des classificateurs entraînés. Il y a souvent une phase d'adaptation (à la voix ou à l'écriture) dont le but est de sélectionner le modèle correspondant le mieux à l'utilisateur.

h. Quelques méthodes de classification

4. K plus proches voisins

Parmi la panoplie de classificateurs utilisés, nous nous intéressons à l'algorithme des k-voisins les plus proches traduction de *nearest neighbor (kNN)* en anglais. Ces performances le situent parmi les meilleures méthodes de classification dont divers domaines d'application. Dans un contexte de classification d'une nouvelle observation x, l'idée fondatrice simple est de faire voter les plus proches voisins de cette observation. La classe de x est déterminée en fonction de la classe majoritaire parmi les k plus proches voisins de l'observation x. La méthode KNN est donc une méthode à base de voisinage, non-paramétrique ; Ceci signifiant que l'algorithme permet de faire une classification sans faire d'hypothèse sur la fonction $y=f(x_1,x_2,\ldots x_p)$ qui relie la variable dépendante aux variables indépendantes. Afin de trouver les K plus proches d'une donnée à classer, on peut choisir entre plusieurs distances ; mais la plus utilisée est la distance euclidienne définie comme suit :

Soient deux données représentées par deux vecteurs x_i et x_j dans $\mathfrak{R}^d$, la distance entre ces deux données est donnée par :

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{d} \left(x_i^r - x_j^r\right)^2}$$

Dans l'exemple illustré par la figure 1. 9, on a 3 classes ω_1, ω_2, ω_3, et le but est de trouver la valeur de la classe de la nouvelle donnée x. On prend la distance Euclidienne et k=5 voisins. Parmi les 5 plus proches voisins, 4 appartiennent à ω_1 et 1 appartient à ω_3, donc la donnée x est affectée à la classe à ω_1(la classe majoritaire).

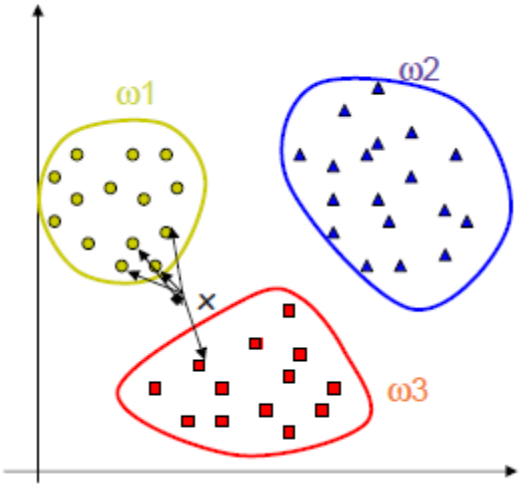


Figure 0-9 Exemple

L'affectation d'une nouvelle observation à une des classes préalablement définie est régie par l'algorithme suivant :

Input :

Données d'apprentissage $X^{train} = (X_1^{train} \ldots . X_n^{train})$;

Classes des données d'apprentissage $Z^{train} = (Z_1^{train} \ldots . Z_n^{train})$;

Données de test $X^{test} = (X_1^{test} \ldots . X_m^{test})$;

Nombre des ppv K

Algorithme KNN :

For i← 1to m do

For j ← 1 to n **do**

Calculer la distance euclidienne d_{ij} entre X_i^{train} et X_j^{test} en utilisant l'équation euclidienne $d_j \leftarrow d_{ij}$

End

Calculer la classe Z_i^{test} du i^{éme} exemple qui vaut la classe de ses k ppv ;

/* trouver les k-ppv de X_i^{test} */

Trier les distance d_j selon un ordre croissant pour j=1,…, n

Récupérer en même temps les indice IndVoisin avant le tri des d_j

Récupérer les classes des K premier ppv à partir des indice IndVoisin et identifier la classe majoritaire:

$C_k \leftarrow$ 0(k=1,…..K)

For k$\leftarrow$1 to K **do**

$\text{Ind\_voisink}_k \leftarrow \text{IndVoisin}_k$

K $\leftarrow$ $Z^{test}_{\text{Ind\_voisink}_k}$

$C_k = z_k + 1$

End

/* trouver la classe de ppv de X_i^{test}

(La classe majoritaire de celle de ces K-ppv) */

$Z_i^{test} = max_{k=1}^{k} C_k$

End

Le paramètre k doit être déterminé par l'utilisateur : k∈N. En classification binaire, il est utile de choisir k impair pour éviter les votes égalitaires. Le meilleur choix de k dépend du jeu de donnée. En général, les grandes valeurs de k réduisent l'effet du bruit sur la classification et donc le risque de sur-apprentissage, mais rendent les frontières entre classes moins distinctes. Il convient donc de faire un choix de compromis entre la variabilité associée à une faible valeur de k contre un 'oversmoothing' ou surlissage (i.e gommage des détails) pour une forte valeur de k. Un bon k peut être sélectionné par diverses techniques heuristiques, par exemple, de validation-croisée. Nous choisirons la valeur de k qui minime l'erreur de classification.

Dans l'algorithme KNN de base, on choisit la classe majoritairement représentée par les K plus proches voisins. Une autre solution consiste à pondérer la contribution de chaque K plus proche voisin en fonction de sa distance avec le nouveau cas à classer. Notez qu'avec cette méthode de pondération, on pourrait utiliser les N exemples au lieu des K plus proches voisins : en effet, plus un exemple sera éloigné du nouveau cas à classer et moins sa classe contribuera au résultat final. Le seul désavantage est la perte de temps.

Les expériences menées avec les KNN montrent qu'ils résistent bien aux

données bruitées. Toutefois, il présente certaines limites: (i) coût mémoire car il faut stocker l'ensemble d'apprentissage en entier et (ii) coût élevé de calcul car il doit explorer l'ensemble d'apprentissage pour classer un nouveau document. Nous proposons, dans le chapitre 4, une version parallèle de cette algorithme à fin de réduire le temps de classification sans dégrader les performances de classification. L'implémentation de cette version parallèle sera implémentée sur une machine dotée d'une carte GPU.

Conclusion

Dans le présent chapitre, nous avons présenté les principaux concepts de l'apprentissage automatique, nous avons donné sa définition, son historique et ses domaines d'application, nous nous sommes intéressés particulièrement au problème de classification et plus précisément à l'algorithme du K plus proches voisins dont sa parallélisassions à l'aide de la technologies des GPU fait l'objet de notre études. Avant d'entamer cette étude, nous avons jugé utile de présenter le domaine d'application que nous avons choisi pour évaluer cet algorithme.

Chapitre II

Détection d’intrusion

1. Introduction :

Avec le développement des réseaux de communication, l'internet est devenu une chose très important dans la vie des gens, La croissance explosive d'utilisateurs d'Internet a motivé l'expansion rapide de commerce électronique et d'autres services en ligne.
Malheureusement derrière la convenance et l'efficacité de ces services, les risques et les chances d'intrusions malveillantes sont aussi augmentés. La sécurité des systèmes informatiques est devenue un défi majeur dont l'objectif est d'assurer la disponibilité des services, la confidentialité et l'intégrité des données et des échanges.
De nombreux mécanismes ont été développés pour assurer la sécurité des systèmes informatiques, particulièrement pour détecter les intrusions dont le but principal est de construire un système sécurisé en déterminant et éliminant les vulnérabilités de sécurité.
Citons par exemple l'authentification qui consiste à prouver l'identité des utilisateurs, le contrôle d'accès qui consiste à définir les droits d'accès accordés aux utilisateurs sur les données et les pare-feu qui filtrent l'accès aux services du système informatique vis-à-vis de l'extérieur.

2. Définition de détection d'intrusion :

La détection d'intrusions consiste à analyser les informations collectées par les mécanismes d'audit de sécurité, à la recherche d'éventuelles attaques. [19]
C'est la capacité à identifier les individus utilisant un système informatique sans autorisation et identifier ceux qui ont un accès légitime au système mais qui abusent de leurs privilèges. [4].

3. Les Approches de la détection d'intrusion :

Les systèmes de détection d'intrusion déterminent si une séquence d'actions constitue une intrusion sur la base d'un ou plusieurs modèles d'intrusion. Un modèle classifie une séquence d'états ou d'actions, ou une caractérisation des états ou des actions, comme " bonne " (aucunes intrusions) ou " mauvaise " (intrusions possibles). Les modèles de détection d'anomalie emploient une caractérisation statistique, et des actions ou des états qui sont statistiquement inhabituel sont classifiés en tant que "mauvaises". Les modèles d'abus d'utilisation comparent des actions ou des états aux séquences connues pour indiquer des intrusions, ou des séquences censées indiquer des intrusions, et classifient

ces séquences en tant que « mauvaises ».

Dans la pratique, des modèles sont souvent combinés, et les systèmes de détection d'intrusion emploient un mélange de deux ou trois types différents de modèles. Et nous on va focaliser sur les deux grandes familles d'approches de détection d'intrusion :

a. Détection des malveillances (approche par scénario) :

La détection des malveillances ou des abus d'utilisation est l'approche la plus basique et la plus ancienne. Elle repose sur le concept de bibliothèque de signatures d'attaque et consiste à surveiller (monitoring) le trafic réseaux à la recherche des empreintes (signatures) d'attaques connues. Cette démarche appliquée à la détection d'intrusion, est très similaire à celle des outils antivirus et présente les mêmes inconvénients que celle-ci. Il est aisé de comprendre que ce type de systèmes de détection est purement réactif ; il ne peut détecter que les attaques dont il possède la signature. De ce fait ce modèle nécessite une connaissance des vulnérabilités du système à surveiller ou des vulnérabilités potentielles que les pirates tentent d'exploiter. Ces connaissances sont incorporées au système de détection sous forme des règles qui seront utilisées lors du processus d'explorations des données d'audit. De tel systèmes utilisent, généralement, des systèmes experts pour analyser les dites données. Et sont incapables de détecter de nouvelles attaques ou des variantes des attaques connues.

Pour mettre en œuvre cette approche de détection des intrusions, on peut avoir recours à l'une de ces techniques :

a- **Systèmes experts :** La composante essentielle des systèmes experts est la base des règles qui représente les signatures d'intrusions avec des règles d'implication. Les signatures décrivent un aspect d'une attaque ou d'une classe d'attaque. La base des règles peut être mise à jour pour détecter de nouvelles attaques. Toutefois, les règles peuvent devenir rapidement très spécifiques au système cible et donc peu portables.

b- **Raisonnement sur des modèles :** Il s'agit de tenter de modéliser un comportement intrusif à un niveau élevé d'abstraction en termes de séquences d'événements qui définissent l'intrusion.

Le raisonnement sur des modèles permet aussi de cibler les données sur lesquelles une analyse approfondie doit être faite. L'inconvénient de cette approche est qu'elle ne permet de dévoiler que des attaques déjà connues.

c- **Analyse des transitions d'états :** Cette analyse consiste à créer un modèle tel qu'à l'état initial le système ne soit pas compromis. L'intrus accède au système. Il exécute une série d'actions qui génèrent des transitions sur les états du modèle. Généralement, ces derniers décrivent le système entant qu'un système compromis.

d- **Réseaux de neurones :** Les réseaux neuronaux permettent l'analyse (linéaire ou non) des données même si elles sont incomplètes ou déformées via leur flexibilité. Leur rapidité permet l'analyse d'importants flux d'audit en temps réel. En outre les réseaux de neurones servent à filtrer et à sélectionner les informations douteuses pour permettre une analyse détaillée par un système expert. On peut aussi les utiliser directement pour la détection de malveillances. Mais leur apprentissage s'avère fastidieux et délicat.

e- **Algorithmes génétiques :** Ces algorithmes ont été proposés par JOHN HOLLAND dans les années 70 [13]. Leur application dans le domaine de la détection d'intrusion, et précisément dans l'approche par scénarios, stipule que chaque scénario d'attaque est représenté par un ensemble pas forcément ordonné d'événements. Les algorithmes génétiques revêtent une importance cruciale face à un mélange possible entre ces ensembles, l'explosion combinatoire qui en résulte rend l'utilisation des algorithmes de recherche traditionnels irrationnelle.

b. Détection d'anomalies :

Les modèles comportementaux sont apparus bien plus tard que les systèmes à base de signatures. Initialement proposés par JP. ANDERSON [1] puis repris et étendus par D.E. DENNING[10], ces modèles se basent sur l'hypothèse selon laquelle l'exploitation d'une vulnérabilité du système implique un usage anormal de celui-ci. Une intrusion est donc identifiable en tant que déviation par rapport au comportement habituel d'un utilisateur [Abr]. Comme exemples étayant cette hypothèse on cite :

- Une tentative d'intrusion par un utilisateur inconnu du système donnera lieu à un taux anormal de mots de passe erronés
- Un intrus par déguisement se connecte à une heure inhabituelle, il utilise abondamment des commandes lui permettant de changer de répertoire, peut-être n'utilise-t-il jamais l'utilitaire favori de l'utilisateur habituel.

- Un utilisateur connecté légitimement au système et qui essaye de contourner la politique de sécurité se connectera la nuit, exécutera des programmes qu'il n'a pas l'habitude d'utiliser, donnera lieu à un plus grand volume d'enregistrement d'audit, utilisera une imprimante sur laquelle il n'a jamais imprimé
- Un cheval de Troie différera du programme légal dont il a pris la place en termes d'utilisation des ressources d'entrée/sortie.
- Une attaque par déni de service donnera lieu à un taux d'utilisation anormalement élevé de certaines ressources du système.

La détection d'anomalie suppose que tout comportement inattendu est l'évidence d'une intrusion. Elle analyse un ensemble de caractéristiques du système et compare leur comportement à un ensemble de valeurs prévues. Dans le cas où les statistiques calculées ne concordent pas avec les mesures prévues une tentative d'intrusion est signalée. Implicite est la croyance qu'un certain ensemble de métrique peut caractériser le comportement prévu d'un utilisateur ou d'un processus. Il est a noter qu'il existe plusieurs approches et techniques pour construire ou décrire un comportement normale de l'utilisateur :

a- **Observation de seuils** : Il s'agit de fixer le comportement normal d'un utilisateur par la donnée de seuils à certaines mesures (par exemple, le nombre maximum de mots de passe erronés). Cette méthode peut entraîner beaucoup de fausses alarmes et d'événements malveillants non détectés.

b- **Approche bayésienne** : les réseaux Bayésiens mettent l'accent sur les relations de causalité existantes dans les situations et permettent de les modéliser. Etant donné que la connaissance de l'ensemble des relations entre les phénomènes est incomplète, il devient nécessaire de les décrire de manière probabiliste.

c- **Profilage d'utilisateurs** [10] : On établit des profils individuels du travail des usagers, auxquels ils sont censés adhérer ensuite. Au fur et à mesure que l'utilisateur change ses activités, son profil de travail attendu se met à jour. Il reste cependant difficile de déterminer un profil pour un utilisateur irrégulier ou très dynamique.

d- **Profilage de groupes :** Pour réduire le nombre de profil à gérer, on classe les utilisateurs par groupe. Chaque groupe est caractérisé par genre de travail commun. Un profil de groupe est calculé en fonction de l'historique des

activités du groupe entier. On vérifie que les individus du groupe travaillent en conformité et ne dévrillent pas par rapport à ce qu'a été défini comme profil de groupe. Mais il est parfois pas évident de trouver le groupe le plus approprié à une personne. D'ailleurs, il est parfois nécessaire de créer un groupe pour un seul individu.

e- **Profilage d'utilisation de ressources** : Il s'agit d'observer l'utilisation de certaines **ressources** comme les processeurs, les ports de communication, les comptes, les applications, les mémoires de masse, la mémoire vive sur de longues périodes, on vérifie et on compare par rapport à ce qui a été observé par le passé. On peut aussi observer les changements dans l'utilisation des protocoles réseau, rechercher les ports qui voient leur trafic augmenter anormalement. L'expérience a montré qu'il est difficile d'interpréter les écarts par rapport au profil normal.

f- **Profilage de programmes exécutables** : Les virus, les chevaux de Troie et autres programmes malveillant peuvent être démasqués profilant la façon dont les objets du système comme les fichiers ou les imprimantes sont utilisés. Donc, le profilage de programmes exécutables stipule qu'on observe l'utilisation des ressources du système par les programmes exécutables. Ce profilage peut se faire par type d'exécutable. On peut par exemple détecter le fait qu'un serveur d'impression se mette à attendre des connexions sur des ports autres que ceux qu'il utilise d'habitude.

g- **Profilage statistique** : DENNING a défini un modèle statistique de comportement utilisateur dans [10]. Ce modèle statistique permet de déterminer, au vue de n observations x1, ..., xn faites sur une variable x, si la valeur xn+1 de l'observation (n+1) est normale ou non.

Explicitement, un profil est constitué d'un ensemble de variables représentant une quantité accumulée d'événements (nombre de fois qu'une commande système particulière a été exécutée par un utilisateur, nombre de quantum de temps CPU occupé par un programme, etc.) pendant une certaine période de temps.

c. Comparaison entre détection d’anomalie et abus d’utilisation :

La différence entre ces deux approches réside dans les points suivants :

- Un système de détection d'intrusion utilisant l'approche de détection des malveillances doit " connaître " toutes les signatures possibles. Il doit identifier les détails d'une attaque aussi bien que son modèle à un niveau d'abstraction élevé qui caractérise la classe de l'attaque. De son côté un système se basant sur les modèles comportementaux doit disposer d'une complète connaissance sur les différents comportements probables du système pour être en mesure de détecter toutes les attaques. En réalité n'est pas possible et représente une situation idéale.
- un système de détection utilisant une base de signatures exige un effort de configuration moins que celui exigé par un système de détection basé sur les modèles comportementaux. Cependant il nécessité plus de données, d'analyse et de mise à jour. Par contre, les systèmes de détection basés sur les modèles comportementaux plus difficiles à configurer par ce qu'il demande une définition compréhensive des comportements connus et probables du système. En général, un support automatique est fourni mais nécessite beaucoup de temps dans son développement et les données qu'il utilise doivent être claires.
- Les systèmes de détection d'intrusions utilisant une base de signatures produisent des conclusions basées sur " pattern matching ". Alors que les conclusions des systèmes de détection basés sur les modèles comportementaux sont basées sur des corrélations statistiques entre les profils actuels et probables.
- l'exactitude des signatures qui ne sont pas spécifiques et des profils non correctement spécifiés pour décrire des comportements probables, produit potentiellement un nombre élevé de « faux positif » et du « faux négatif »

Afin de contourner les inconvénients et de tirer profits des avantages de chacune des approches, certaines systèmes de détection utilisent une combinaison des modèles comportementaux (détections des anomalies) et de la détections des malveillances.

4. Définition des Systèmes de détection d'intrusion « SDI » :

IDS signifie Intrusion Détection System (Système de détection d'intrusions). Il s'agit d'un équipement permettant de surveiller l'activité d'un réseau ou d'un hôte donné, afin de détecter toute tentative d'intrusion et éventuellement de réagir à cette tentative. [3] Debar [9] simplifie le système de détection d'intrusions dans un détecteur qui analyse les informations en provenance du système surveillé (voir figure 2-1).

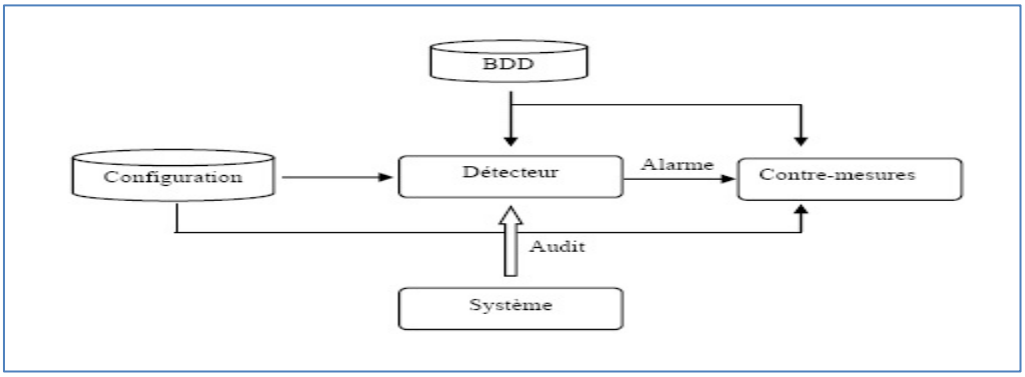


Figure 2-1 Modèle simplifié d'un système de détection d'intrusions.

La technologie des systèmes de détection d'intrusions permet d'analyser les données recueillies de trois façons :

Analyse centralisée L'IDS : possède plusieurs capteurs, il centralise les alertes pour les analyser sur une seule machine. Ce type d'analyse présente l'avantage d'avoir une vue globale sur toutes les machines protégées. Toutefois, il a l'inconvénient d'occupation très longue du réseau pour acheminer l'information.

Analyse locale : Chaque machine dispose d'un capteur et analyse l'information à son niveau. Avec ce type d'analyse le trafic réseau est diminué mais les attaques distribuées peuvent échapper à la détection.

Analyse distribuée : Des petits programmes appelés agents sont déployés sur les nœuds du réseau. Pour les besoins d'analyse un agent est envoyé sur une machine pour traiter l'information.

5. L’architecture d’un système de détection d’intrusion :

A un niveau macroscopique, un système de détection d'intrusions est constitué, essentiellement de trois composantes :

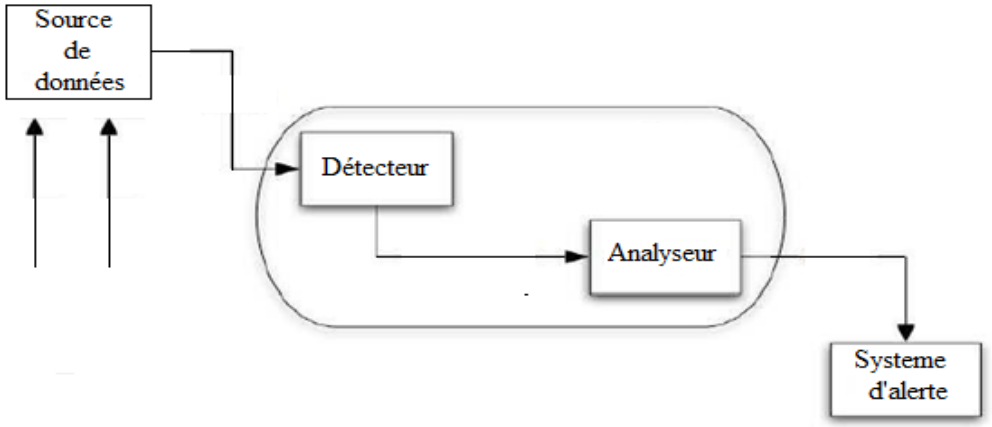


Figure 2-2 architecture d’un système de détection d’intrusion

a. Détecteur (Sonde) :

Le détecteur peut être réalisé sous forme de logiciel seulement ou d'une combinaison de Matériel/Logiciel. Il est responsable de la collecte des informations sur l'usage du système et de leur transmission, d'une manière sûre, vers l'analyseur. Il est à signaler que ces information, qui nécessitent par fois des opérations de filtrage, sont stockées sous forme d'enregistrement d'audit dans un format spécial (format du système de détection) Aussi, il est à noter que le principal défi d'un système de détection d'intrusion reste la sécurisation de communication des "rapports" d'activité générés par les sondes. Pour cela on a recours généralement à des techniques cryptographiques.

b. L'analyseur :

L'analyseur, récupère les informations d'audit générées par une ou plusieurs sondes ou d'un autre analyseur. Après une phase de prétraitement qui consiste à réduire la quantité de données collectée en éliminant les données inutiles ou redondantes, il procède à l’analyse de ces données à la recherche des traces d’une éventuelle intrusion (achevée ou en cours). Pour effectuer son analyse, il emploie soit une seule technique ou approche de détection soit une combinaison d’une ou plusieurs approches. L’utilisation d’une combinaison de techniques est l’issue la plus utilisée car elle accentue l’efficacité

du système de détection.

L'analyseur peut être implémenté sur les systèmes qu'il surveille ou un système séparé. Un système séparé empêche un attaquant ayant réussi son exploit d'effacer ou de modifier l'information de détection d'une part et d'autre part il emploie peu de ressources du système surveillé.

c. Système d'alerte :

Le système d'alerte reçoit l'information de l'analyseur, et, en cas d'une intrusion détectée, prend la mesure appropriée. Dans certains cas, il se contente, simplement, d'aviser le chargé de sécurité par une un simple message ou un beep. Dans d'autres cas, il peut prendre une certaine mesure pour répondre à l'attaque. La réponse peut prendre plusieurs formes : coupure de la connexion, filtrage de paquets, ordonner le ou les détecteurs de collecter plus de données … système d'alerte peut, également, disposer d'une interface graphique qui permet de visualiser l'état du système surveillé et de contrôler son comportement.

En plus de ces trois composantes, un système de détection d'intrusion peut éventuellement contenir un "pot de miel" qui n'est autre qu'un sous-système conçu et configuré pour qu'il soit visible et accessible par un intrus. Lors de toute attaque détectée, l'intrus est dirigé vers ce sous-système afin de récolter suffisamment d'informations pour le contourner. Un système de détection d'intrusion dispose, aussi, d'une ou plusieurs bases de données et/ou de connaissances. Ces bases contiennent les informations de configuration et les règles relatives aux modèles de détections dictant le comportement du système de détection lors de son fonctionnement.

6. Les types des SDI :

a. Les systèmes de détection orientée application :

Les systèmes de détection d'intrusion orientée application détectent les attaques visant une application spécifique. L'information d'audit nécessaire à la détection d'intrusion est, habituellement, obtenue en utilisant l'appel système « syslog » ou en dotant l'application, elle-même, par des mécanismes spécifiques d'audit.

Ajouter des mécanismes d'audit à une application existante exige la modification de l'application de sorte qu'elle produise l'information d'audit en réponse aux événements appropriés de sécurité. Ceci peut être accompli par différentes manières :

En modifiant directement le code source de l'application pour inclure le code d'audit. Cette approche exige que le code d'application soit disponible et modifiable. Interposer

le code responsable d'extraire l'information d'audit dans les interfaces utilisées par l'application tel que les appels systèmes ou les bibliothèques standards (C standards library). Mais avec cette approche, des applications autre que celles surveillées seront affectées en terme de performances (deviendra plus lourde à exécuter). Utiliser les « hooks extention » fournies par l'application elle-même pour implémenter l'ensemble des fonctionnalités d'audit, cette approche offre plus de flexibilités, mais les applications n'offre pas toute cette possibilité.

b. Les Systèmes de détection par l'hôte :

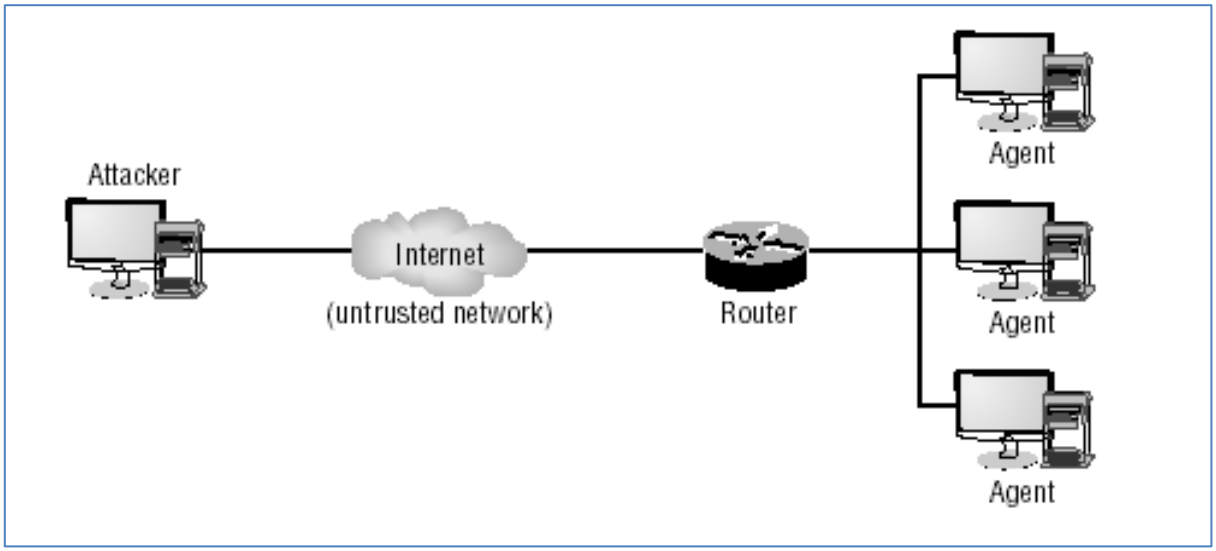


Figure 0-3 Exemple de détection par l'hôte

Les systèmes de détection par l'hôte opèrent au niveau d'une machine et analysent les données d'audit générées par son système d'exploitation à la recherche des éventuelles intrusions. Les sources de données d'audit peuvent être :

a) Le système d'information :

Les systèmes d'exploitation rendent disponible, au profit des processus, dans l'espace d'utilisateur des informations relatives à leurs fonctionnements internes et à leur sécurité. Ces informations sont collectées et traitées par des programmes tels que « ps, vmstat, netstat ». Elles sont, le plus souvent, très complètes et très fiables car elles sont produites par le noyau. Malheureusement, peu de systèmes d'exploitation sont dotées de mécanismes collectant, systématiquement et sans interruption cette information.

b) Service de syslog :

Le syslog est un service d'audit fourni par beaucoup de systèmes d'exploitation de type UNIX. Il permet à des programmeurs d'indiquer un message textuel décrivant un événement à enregistrer. L'information additionnelle, comme le moment où l'événement s'est produit et l'hôte sur lequel le programme fonctionne, est automatiquement ajoutée. En raison de leur simplicité, des événements de syslog sont employés intensivement par des applications. Cependant, les applications enregistrent habituellement des informations utiles pour le débogage qui ne sont pas nécessairement utiles pour la détection d'intrusion. En outre, un format spécifique d'audit n'est pas imposé par le service mais change selon le programme qui les a générées. Ainsi, il peut être difficile d'extraire des données d'audit à partir des logs.

En conclusion, les fichiers logs peuvent être facilement pollués par des messages crées par l'intrus.

Un système de détection d'intrusion par l'hôte a les avantages suivants:

- Il détecte les tentatives de contourner le système de détection d'intrusion réseaux tel quel les attaques par fragmentation des paquets et « time live » attaque.
- il détecte des attaques masquées avec des techniques de chiffrage
- il permet de vérifier si une attaque a réussi réellement. (un système de détection réseau détecte l'attaque mais il ne peut pas déterminer si l'attaque a réellement réussie)
- Il ne demande pas de matériel spécialisé

D'un autre coté il doit être installé sur chacun des postes à surveiller et doit être compatible avec plusieurs systèmes d'exploitation.

c. Les Système de détection réseau :

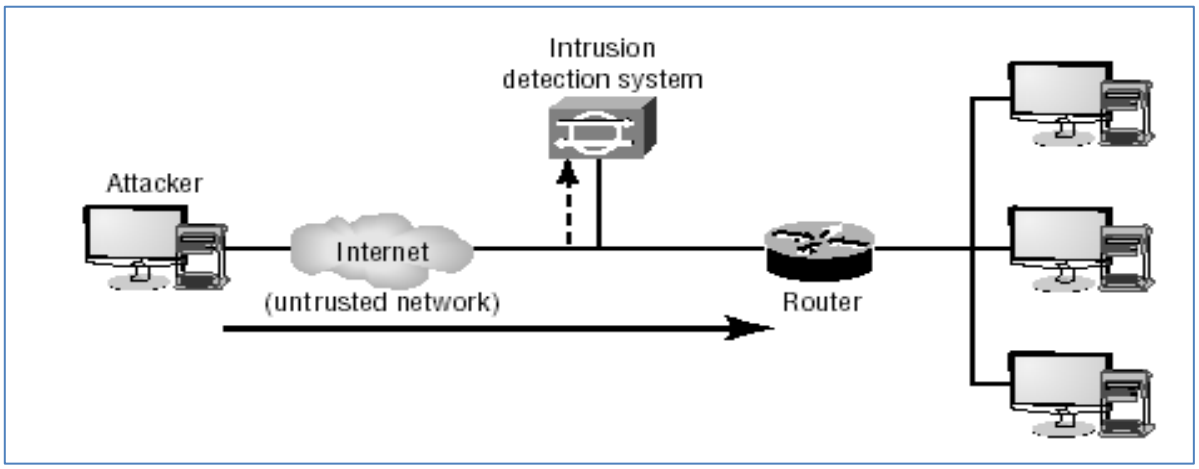


Figure 2-4 Exemple de détection par réseau

Les systèmes de détection d'intrusion réseaux s'installent sur le réseau ou sur un segment du réseau et surveillent, en temps réel, le trafic des paquets à la recherche d'une signature d'attaque. Par conséquent, dans de tels systèmes de détection les détecteurs sont des sniffers réseau. L'analyse du contenu des paquets de réseau peut être effectuée à différents niveaux de sophistication. Elle peut, par exemple, concerner les entêtes des paquets circulant dans le réseau ou elle inclue la totalité de son contenu, comme elle peut exploiter la connaissance sur les protocoles utilisés lors d'une connexion. Des niveaux d'analyse plus élevés supportent des analyses plus sophistiquées mais elles sont très lentes et exigent plus de ressources.

Les systèmes de détection réseaux sont très attrayants parce qu'ils sont facile à déployer et en n'ont presque pas d'impact sur les hôtes surveillés. Un système de détection d'intrusion réseau a deux avantages : le premier avantage est son émission d'alerte en temps réel, ce qui donne aux administrateurs la possibilité d'arrêter ou de contenir une intrusion avant qu'elle ne réussisse. C'est particulièrement valable pour les attaques de type DOS qui doivent être traitées immédiatement pour atténuer les dommages. D'autre part les systèmes de détection d'intrusion réseau sont capables de stocker des informations sur la session, ce qui aide les administrateurs à déterminer les vulnérabilités présentent dans leur système. Le type et nature de l'attaque lancée par un intrus contre un système indique ou détermine les vulnérabilités existant sur le réseau.
Les systèmes de détection d'intrusion réseau son indépendant des systèmes d'exploitation, mais nécessitent un nœud dédié à la surveillance, d'une carte réseau fonctionnant en mode « promiscues » et une connexion sécurisée entre les différentes sondes et la console maîtresse.
La classification des systèmes de détection d'intrusion en systèmes orientés réseau et à ceux orientés systèmes se rapporte à la manière dont sont collectées les informations par les systèmes de détection et non pas à la façon dont à lieu leurs traitement. Pour garantir cette distinction, nous introduisons le terme collecte de données orientée système ou hôte et collecte de données orientée réseau. Ainsi une autre classification s'impose. Elle consiste à distinguer des systèmes de détection distribués et ceux centralisés.
Dans les systèmes de détection d'intrusion distribués, les données sont collectées et traitées au niveau de multiples hôtes contrairement aux systèmes centralisés dont lesquels les données peuvent être collectées d'une manière distribuée mais traitées d'une façon centralisée. En général, on croit que la collecte des données orientée hôte est

meilleure que celle orientée réseau pour les raisons suivantes :

- Les techniques, orientée hôte, de collecte de données permettent d'avoir des données reflétant exactement se qui se passe au niveau de l'hôte. Tandis qu'un moniteur réseau pourrait potentiellement manquer des paquets.
- Les mécanismes, orientés réseau, de collecte de données sont sujets à des attaques d'insertion et d'évasion (Ptacek et Newsham).

7. Les caractéristiques des Systèmes de détection d'intrusion :

Aujourd'hui les systèmes de détection d'intrusion sont réellement devenus indispensables lors de la mise en place d'une infrastructure de sécurité opérationnelle. Ils s'intègrent donc toujours dans un contexte et une architecture qui imposent des contraintes pouvant être très diverses. C'est pourquoi il n'existe pas de grille d'évaluation unique pour ce type d'outil. Pourtant un certain nombre de critères peuvent être dégagés ; ceux-ci devront nécessairement être pondérés en fonction du contexte de l'étude.

a. Fiabilité : Un détecteur d'intrusion doit être fiable ; les alertes qu'il génère doivent être justifiées et aucune intrusion ne doit pouvoir lui échapper. Un système de détection d'intrusion générant trop de fausses alertes sera à coup sûr désactivé par l'administrateur et un autre ne détectant rien ne sera rapidement considéré comme inutile.
b. Réactivité : Un système de détection d'intrusion doit être capable de détecter les nouveaux types d'attaque le plus rapidement possible ; pour cela il doit rester constamment à jour. Des capacités de mise à jour automatique sont pour ainsi dire indispensables.
c. Facilité de mise en œuvre et adaptabilité : Un système de détection d'intrusion doit être facile à mettre en œuvre et doit pouvoir surtout s'adapter au contexte dans lequel il doit opérer; il est inutile d'avoir un système de détection d'intrusion émettant des alertes en moins de 10 secondes si les ressources nécessaires à une réaction ne sont pas disponible pour agir dans les mêmes contraintes de temps.
d. Performance : la mise en place d'un système de détection d'intrusion ne doit en

aucun cas affecter les performances des systèmes surveillés. De plus, il faut toujours avoir la certitude que le système de détection d'intrusion a la capacité de traiter toute l'information à sa disposition (par exemple un système de détection d'intrusion réseau doit être capable de traiter

e. l'ensemble du flux : pouvant se présenter à un instant donné sans jamais dropper de paquets) car dans le cas contraire il devient trivial de masquer les attaques en augmentant la quantité d'information.
f. Multi canal : Un bon système de détection d'intrusion doit pouvoir utiliser plusieurs canaux d'alerte (email, téléphone, fax...) afin de pouvoir garantir que les alertes seront effectivement émises.
g. Information : Le système de détection d'intrusion doit donner un maximum d'information sur l'attaque détectée afin de préparer la réaction. Classification : il doit être aisé de hiérarchiser la gravité des attaques détectées afin d'adapter le mode d'alerte.

8. Interopérabilité des systèmes de détection d'intrusion :

La détection d'intrusion a suscité ces dernières années une très grande attention et son déploiement au sein des entreprises, se fait d'une manière très rapide. Pour faire face à cette demande grandissante, une pléthore de produits de détection d'intrusion, en version commerciale aussi bien qu'en « open source » a été lancée (environ 130 ont été répertoriés dans [7] en janvier 2003). Or le problème crucial de ces produits fermés et incompatible entre eux, réside dans le fait qu'on ne disposait pas d'un moyen standard pour les faire communiquer. Dans ce contexte, plusieurs efforts ont été consentis afin de résoudre ce problème et de définir ainsi des outils d'interopérabilité entre les systèmes de détection d'intrusion. Parmi les aboutissements de ces efforts, on cite :

- ✓ CIDF

Common Intrusion Detection Framework" [4] fut le résultat d'un projet de recherche entamé par DARPA[1] et ne visait pas d'être un standard commercial. Cette plate-forme est composée d'un générateur d'évènements, d'un analyseur, d'une base de données et d'un système de réponse. La communication entre ces composants se fait à l'aide de

CISL2 un langage appartenant à la famille Lisp. N'ayant pas données satisfaction à plusieurs auteurs de systèmes de détection ; il fut délaissé

- ✓ IDMEF

Intrusion Detection Message Exchange Format: a été proposé par le groupe de travail de détection d'intrusion au sein de l'Internet Engineering Task Force (IETF) pour être un format standard d'échange de données entre différents systèmes de détection d'intrusion. D'une manière évidente, il devrait être implémenté au niveau du canal entre la sonde et le moniteur auquel elle envoie les alertes, néanmoins d'autres emplacements sont possibles :

- Au niveau du système gestion de la base de données devant stocker les résultats emmenant de différents systèmes de détection d'intrusion ce qui permettrait une analyse globale de ces résultats au lieu d'une analyse séparée de chaque jeux.
- Au sein du système de corrélation d'événements devant recevoir des alertes de différents systèmes de détection d'intrusion. Ce la rendrait possible une cross-corrélation plus sophistiqué d'alertes provenant de plusieurs systèmes de détection au lieu de se contenter d'une simple corrélation limitée aux alertes produites par un seul système de détection.
- Au niveau de l'interface graphique devrant afficher les alertes produites par différents systèmes de détection. Un format standard d'échange de données pourra, non seulement, faciliter considérablement cette tache d'affichage, mais aussi permettra la communication d'information sur ces alertes.

En fin, il est à noter que le XML a été retenu par IDWG pour implémenter l'IDMEF pour sa popularité et sa flexibilité mais aussi parce qu'il :

- Donne la possibilité de définir un langage spécifique pour la détection d'intrusion ainsi que celle d'étendre ce langage pour des révisions ultérieures
- XML est un support substantiellement disponible, ainsi que ses outils de traitement
- Disponibilité de documents et des APIs pour analyser et valider XML pour plusieurs langages tel que Java, C/C++. L'accès répondu à ces outils rend l'adoption de l'IDMEF plus facile, rapide et beaucoup plus conviviale.
- XML projette de devenir un standard reconnu mondialement.
- XML peut supporter le filtrage et l'agrégation une fois associer à XSL
- XML et libre.

9. Une vue générale de quelque systèmes de détection d'intrusions existants :

Il existe plusieurs systèmes de détection d'intrusions qui ont été développés. Dans cette section, nous présenterons quelques systèmes de détection d'intrusions existant :

- IDES :

IDES (Intrusion-Detection Expert System) a été développé par SRI International System Design Laboratory. Il représente le modèle de référence pour un grand nombre de systèmes de détection d'intrusions. Il a été conçu pour surveiller un seul hôte et il traite uniquement les données d'audit. Ce système de détection d'intrusions est indépendant du système surveillé, il fonctionne sur une machine dédiée, reliée au système par un réseau. Afin de détecter les violations de sécurité en temps réel, IDES s'appuie aussi bien sur une approche statistique que sur un système expert.[17][15] Ainsi, il est constitué de deux éléments importants :

- Le détecteur d'anomalie : qui est responsable de la détection des comportements atypiques, en utilisant des méthodes statistiques du modèle de Denning. [10]
- Le système expert : qui est chargé de détecter les attaques suspectes en s'appuyant sur une base de connaissances de scénarios d'attaques connus. [15]

- NIDES :

NIDES (Next- Generation IDES)[15][17] est une version améliorée du système de détection d'intrusions IDES. Il assure la détection d'intrusions sur plusieurs hôtes (distribuées) en se basant toujours sur les données d'audit. Il n'y a aucune analyse du trafic réseau. Il utilise les mêmes algorithmes qu'IDES.

- NADIR :

NADIR (Network Anomaly Detection and Intrusion Reporter) [15][17][2] est un système expert qui a été conçu pour le réseau ICN (Integrated Computing Network) du Laboratoire National Los Alamos. Son but est d'analyser les activités réseaux des utilisateurs et d'ICN en se basant sur les règles du système expert qui définissent la politique de sécurité et les comportements suspects. L'inconvénient majeur de ce système est qu'il ne peut être porté sur d'autres réseaux, étant donné que les

protocoles réseaux d'ICN ne sont pas standards.

- DIDS :

DIDS (Distributed Intrusion Detection System)[15][17][23] est un système détection d'intrusions basé réseau qui se base sur l'approche hiérarchique. Afin d'évite la dégradation des performances de système, DIDS délègue certaines analyses locales **aux** hôtes locaux. Son architecture se compose de trois entités :

- Le « Host Monitor » : Il en existe un par hôte. Il collecte les données de l'hôte surveillé, fait une première analyse simple sur ces données puis transmet les événements pertinents au « DIDS Director ».
- Le « LAN Monitor » : Il en existe un pour chaque segment LAN. Il surveille le trafic sur le LAN, collecte les informations réseaux et reporte au « DIDS Director» les activités suspectes et non autorisées qui se sont produites sur le réseau.
- Le « DIDS Director » : Il analyse les rapports reçus du « LAN Monitor » et des « Host Monitor » afin de détecter les attaques potentielles.

- CSM :

CSM (Cooperating Security Manager)[21][9] est un système de détection d'intrusions qui peut être utilisé dans un environnement de réseau distribué. Son principal objectif est de détecter les activités intrusives de façon non centralisée car utiliser un directeur central qui coordonnerait toutes les activités limiterait la taille du réseau « le problème d'incrémentabilité ». Pour cela, CSM doit s'exécuter sur chaque hôte connecté au réseau. Ainsi, au lieu de reporter les activités anormales à un directeur central, le CSM communique entre eux pour détecter d'une manière coopérative les intrusions réseaux. Les composants principaux de ce système de détection d'intrusions sont :

- Un système de détection d'intrusions local (SDI) : qui assure la détection d'intrusions pour un hôte local.
- Un gestionnaire de sécurité : qui coordonne la détection d'intrusions distribué entre les CSM.
- Un gestionnaire d'intrus (IH : intruder handling component) : dont le rôle est d'entreprendre les actions nécessaires lorsqu'une intrusion est détectée.

Conclusion :

Nous avons présenté dans ce chapitre une étude des systèmes de détection d'intrusions. Il nous est paru évident que ces systèmes sont à présent indispensables aux entreprises afin d'assurer leur sécurité. La plupart des systèmes de détection d'intrusions sont construits dans une architecture hiérarchique dont une grande quantité de données transférée à travers le réseau peut résulter une congestion de réseau et sont susceptibles d'être attaqués.

Il est plus que fondamental de former correctement les personnes chargées de la mise en œuvre et de l'exploitation des IDS. Malheureusement, il semble que c'est encore là où aujourd'hui encore subsiste la plus grande partie de la difficulté. Cette technologie n'est pas encore arrivée à maturité et les outils existants ne sont pas toujours à la hauteur des besoins. Certaines approches théoriques doivent encore être validées dans la pratique. De nouvelles approches demandent encore à être approfondies comme l'immunologie et les systèmes orientés agents.

Chapitre III

GPU

1. Introduction

Les applications informatiques sont traditionnellement réalisées avec des programmes de nature séquentiels, ces programmes exécutent une séquence d'instructions sur un seul processeur(CPU) comme illustré dans la figure suivante [2].

Figure 3-1 Exécution Séquentiels

Vue les avancées technologiques en microélectronique et le degré d'intégration attient, la programmation parallèle est devenue un outil indispensable, notamment pour des applications à haut débit. Un algorithme parallèle consiste à partitionner le problème étudié en tâches élémentaires afin que ces tâches soient exécutées simultanément dans plusieurs processeurs en vue que le temps d'exécution soit plus rapide que celui de l'algorithme séquentiel.

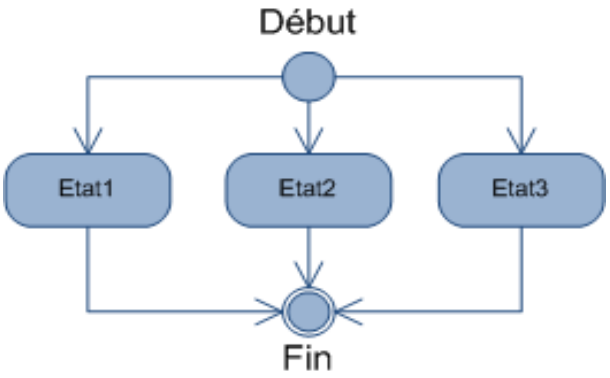


Figure 3-2 Exécution Parallèle

L'arrivée des unités graphiques de calcul (GPUs) programmables rend les cartes graphiques extrêmement intéressantes pour déporter les calculs sur ces processeurs en permettant un parallélisme massif, dans un environnement de programmation correcte à bas coût. Il s'agit alors d'une niche qu'il faut exploiter dans les différentes applications.

Les processeurs graphiques sont habituellement destinés aux calculs 3D issus des jeux et aux applications multimédia. Cependant, ces applications ne constituent qu'une très faible proportion des tâches attendues de ces processeurs. Les processeurs graphiques sont considérés alors sous-exploités. Afin de profiter pleinement de leurs performances, une compagnie appelée NVIDIA à lancer des versions de cartes graphiques dédiées uniquement pour des calculs de haut débit plutôt que pour des jeux vidéo. L'utilisation du GPU comme coprocesseur permet également d'alléger le CPU d'une certaine quantité de travail. Les motivations du GPGPU sont nombreuses dans le calcul à haute performances notamment. L'évolution perpétuelle des applications et des données à gérer demande de plus en plus de ressources en termes de stockage et de calculs. Ainsi, cette technologie présente des atouts intéressants avec un rapport puissance de calcul / prix imbattable. Elle permet également de réduire la consommation électrique et occupe très peu d'espace en comparaison aux traditionnels CPU. Les capacités parallèles des GPU permettent d'exécuter un partitionnement d'une tâche complexe en une multitude de tâches élémentaires, pouvant être réalisées par ses différents processeurs travaillant simultanément, rendant l'exécution globale bien plus rapide qu'une exécution séquentielle. Des algorithmes de domaines variés (météorologie, finance, traitement d'image. . .), se prêtent bien à cette découpe.

2. Historique des GPU

Depuis l'apparition dans les années 80 des accélérateurs graphiques matériels, leur évolution n'a cessé de s'accélérer. Les premières cartes graphiques accessibles au grand publiques sont apparues dans le milieu des années 90, avec en particulier les Voodoo Graphics. C'est NVidia qui a introduit le terme GPU, remplaçant le terme VGA Controller, alors insuffisant pour désigner l'ensemble des possibilités offertes par ces cartes. L'industrie du jeu vidéo, en particulier, n'a depuis cessé de motiver le développement de nouveaux processeurs dédiés au traitement graphiques. Cette évolution peut être catégorisée en différentes générations, selon leurs capacités.

Les forces ayant poussé les industriels à améliorer sans cesse ces cartes graphiques sont

dues principalement à un aspect économique de compétitivité, pour faire face à un appétit de plus en plus grand de complexité visuelle ou de réalisme dans les représentations cinématographiques ou les simulations vidéo-ludiques, poussé par une volonté assurément humaine de divertissement. Suivant la loi de Moore, faite pour les CPU qui ne la respectent plus aujourd'hui, et stipulant que le nombre de transistors sur un processeur double tous les 18 mois, les architectures des GPU ont évolué depuisune décennie, franchissant parfois des sauts technologiques importants. Nous nous proposons ici de classer ces architectures en différentes géné- rations. Avant les GPU Dans les années 80, des sociétés comme Silicon Graphics ou Evans and Sutherland proposèrent des solutions matérielles extrêmement couteuses, réservé à quelques professionnels spécialisés, et ne pouvant effectuer que quelques opérations graphiques très simples comme des transformations de vertex ou des applications de textures. De façon générale, le reste des opérations graphiques étaient accomplies par le CPU.

La première génération de GPU à proprement parler a débuté avec l'arrivée des Voodoo Graphics de 3dfx Interactive en 1996, et dura jusqu'en 1999. Les principales cartes de cette génération sont les TNT2 de NVidia (architecture NV5), les Rage d'ATI et les Voodoo3 de 3dfx. Les premières opérations graphiques disponibles sont la rastérisation de triangles et l'application de texture. Ces cartes implémentent également le jeu d'instruction de DirectX 6, API (Application Programming Interface, Interface de Programmation) alors standard. Cependant, elles souffrent de certaines limitations, principalement la grande faiblesse du jeu d'instructions mathématiques et l'impossibilité de transformer matériellement des vertex, opération lourde encore à la charge du CPU. Deuxième génération Les premières GeForce 256 de NVidia (NV10) font leur apparition en 1999, à peu près en même temps que les Radeon 7500 d'ATI (architecture RV200) et Savage 3D de S3. Ces cartes permettent maintenant une prise en charge complète de la transformation des vertex et du calcul des pixels (Transform and Lightning, T&L). Les deux API principales, DirectX 7 et OpenGL, sont maintenant supportées par ces cartes. Les améliorations apportées au jeu d'instructions rendent ces cartes plus facilement configurables, mais pas encore programmable : les opérations sur les vertex et les pixels ne peuvent être modifiés par le développeur. Troisième génération Dès cette génération, les constructeurs NVidia et ATI se partagent la quasi-totalité du marché, NVidia ayant fait l'acquisition de 3dfx. Les Geforce 3 (NV20) en 2001 et GeForce 4 Ti (NV25) en 2002 de NVidia, la Radeon 8500 d'ATI (R200) en 2001 forment cette génération de GPU, permettant enfin au développeur de diriger la

transformation des vertex par une suite d'instructions qu'il spécifie. Néanmoins, la programmabilité des opérations sur les pixels n'est toujours pas possible. DirectX 8 et quelques extensions à OpenGL permettent tout de même une plus grande souplesse de configuration dans le traitement de ces pixels. Quatrième génération Courant 2002, ATI propose sa Radeon 9700 (R300), et NVidia sa GeForce FX (NV30) à la fin de la même année. Ces deux cartes, en plus de supporter le jeu d'instructions DirectX 9 et de nouvelles extensions OpenGL, apportent de plus la possibilité de programmer le traitement des pixels. C'est à partir de cette quatrième génération de cartes que les premières opérations GPGPU ont pu se faire.

Cinquième génération Les GeForce 6 (NV40, avril 2004) et GeForce 7 (G70, juin 2005) de NVidia, les Radeon X800 (R420, juin 2004) et X1800 (R520, octobre 205) et dérivées composent cette génération. Ces cartes permettent quelques fonctions intéressantes, en particulier en calcul GPGPU : l'accès aux textures lors de la transformation des vertex, le rendu dans différentes textures (Multiple Render Target, MRT) et le branchement dynamique, uniquement pour la transformation des sommets, améliorant grandement la vitesse d'exécution. Le traitement des pixels ne supporte pas ce branchement dynamique. Sixième génération C'est la génération en plein essor actuellement, équipant la plupart des ordinateurs sur le marché. Ses limites sont plus floues, chaque constructeur apportant ses innovations propres. Elle est composée des GeForce 8 (G80) et GeForce 9 (G92), lancées respectivement en 2006 et 2008, apportant des modifications dans la façon de concevoir le pipeline graphique. Entre la transformation des vertex et la rastérisation des primitives (leur transformation en fragments, ou pixels), une étape supplémentaire est insérée, permettant de modifier la géomé- trie du maillage des primitives : insertions et suppression de vertex sont possibles, ce qui ouvre de nouvelles voies au calcul GPGPU. L'ajout du type int utilisable en interne (registres) et surtout en externe (textures) a également contribué à élargir le nombre d'applications possibles. De plus, sur les cartes GeForce 8, il n'existe plus de différence physique entre les différents processeurs. L'architecture matérielle modifiée est dite uni- fiée, cette caractéristique étant exploitée par le langage de programmation CUDA, de NVidia, utilisable avec des cartes à architectures G80 ou supérieures. La série GeForce 9 ne connait pas un succès véritable, ayant moins de mémoire et ses performances étant égales voire inférieures aux cartes équivalentes de la série GeForce 8, pour un prix semblable. Les séries de cartes d'ATI de cette génération sont les Radeon HD2000 et Radeon HD3000 (R600) , lancées en 2007, pour contrer la

série GeForce 8. Malgré quelques améliorations intéressantes proposées par ce constructeur (support de DirectX 10.1, de la norme Shader 4.1 par exemple), ces cartes peinent à s'imposer à cause d'un prix trop élevé, de pièces bruyantes et de puces chauffant beaucoup. Septième génération La dernière génération en date n'est pas encore répandue à l'heure actuelle. Les GeForce 200 (G200), lancées en juin 2008, apportent des améliorations principalement techniques : augmentation de la mémoire disponible, du nombre de processeurs, des fréquences mé- moire et GPU, de la bande passante par élargissement du bus de données, pour des performances annoncées jusqu'à deux fois supérieures aux sé- ries précédentes. Pour ATI, les Radeon HD4000, en juin 2008 également, sont censées rivaliser avec la série GeForce 9, mais les performances des modèles haut de gamme les mettent au même niveau que les GeForce 200, proposant les mêmes types d'améliorations techniques, mais affichant des tarifs plus attractifs et une consommation électrique revue à la baisse.

Il est important de noter que les programmes écrits pour un type ou une génération de cartes restent utilisable avec les générations futures de cartes, même si toutes les capacités ne sont plus nécessairement exploitées. C'est, par ailleurs, un problème important de développement que de devoir faire avec les différentes générations de cartes présentes sur le marché.

3. Architecture des GPU

Le GPU (Graphics Processing Unit) est un circuit intégré présent sur les cartes graphiques qui présente une architecture parallèle. Il suit le paradigme Single Instruction Multiple Data (SIMD) c'est-à-dire qu'il permet d'exécuter une même instruction sur plusieurs données. Son rôle premier est d'accélérer l'exécution des tâches graphiques (à base de calcul vectoriel). Les GPU étant initialement prévus pour les applications graphiques (jeux vidéo), ils bénéficient de débouchés commerciaux importants, ce qui en fait une ressource matérielle peu onéreuse. Cependant, les applications graphiques ne sont pas les seules à pouvoir bénéficier de cette architecture matérielle. Il est possible d'utiliser le GPU pour exécuter les applications parallèles. C'est le GPGPU (General-purpose computing on graphics processing units). Il permet d'obtenir une grande puissance de calcul en utilisant un GPU ou une combinaison de GPU.

Le GPU possède une mémoire propre et est relié au processeur et à la mémoire par l'intermédiaire du bus PCI. Ce bus a des performances moins élevées que le bus

mémoire. Le GPU est composé d'une grande quantité d'unités de calcul dont le jeu d'instruction est plus simple que celui d'un processeur classique. Cette architecture est orientée vers l'exécution d'instructions en parallèle sur un ensemble de données. Exécuter des applications sur une telle architecture nécessite de prendre en compte l'ensemble de ces nouveaux paramètres. Il faut pouvoir spécifier quelles sont les instructions à exécuter sur le processeur central, celles qui peuvent être parallélisées sur le GPU et les transferts de données entre les mémoires du GPU et du processeur. Cela demande encore une fois de contrôler très précisément l'exécution de l'application.

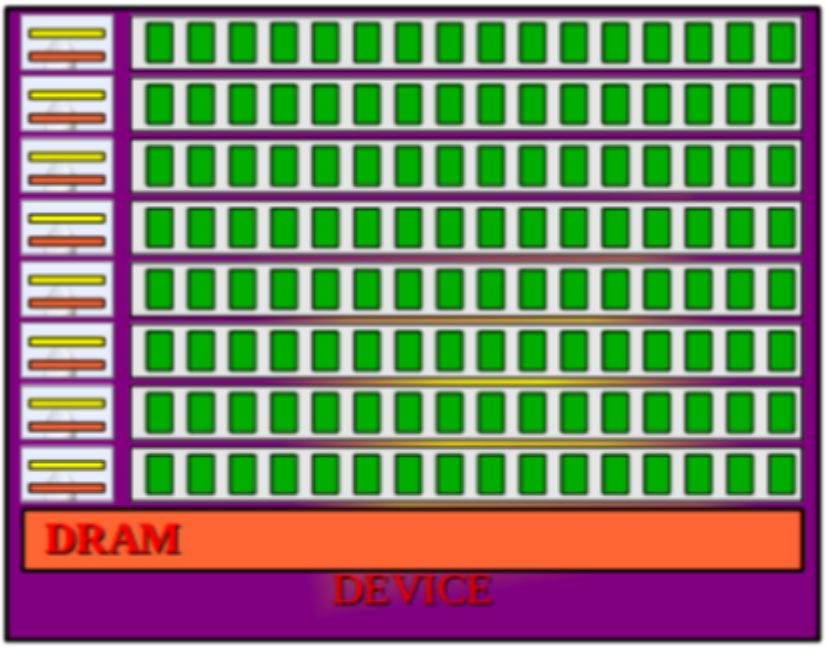


Figure 3-3 Architecture d'un GPU

Il existe sur la carte graphique plusieurs zones mémoires distinctes aux propriétés différentes [16] :

– La mémoire globale est associée à la carte (et non à un multiprocesseur). Elle est accessible en lecture et en écriture, avec un délai de 100 à 400 cycles pour récupérer les données pour chaque lecture.

– La mémoire locale est associée à un thread, elle est accessible très rapidement en lecture et écriture.

– Les registres sont associés à un processeur, ils sont accessibles très rapidement en lecture et écriture, et peuvent être utilisés comme prédicats. Les registres ne sont pas

adressables.

– La mémoire partagée (shared) est associée à un bloc, et existe localement sur un multiprocesseur. Elle permet des temps d'accès instantanés, sauf dans certaines conditions particulières. Elle est adressable en lecture et écriture.

– La mémoire constante est associée à la carte, mais il existe un cache sur chaque multiprocesseur. Le premier accès est coûteux, les suivants sont rapides, tant que le cache ne déborde pas. Cette mémoire est adressable en lecture.

– La mémoire texture est associée à la carte, elle permet de faire du sampling (extraire une valeur à des coordonnées flottantes en faisant de l'interpolation entre cases mémoire). Elle est accessible en lecture, non adressable (sauf par sampling), et dispose d'un petit cache au niveau de chaque microprocesseur.

4. Communication CPU-GPU

La figure ci-dessous montre les différentes étapes associées à l'utilisation du GPU. Pour effectuer un calcul sur l'accélérateur GPU on doit suivre les étapes suivantes :

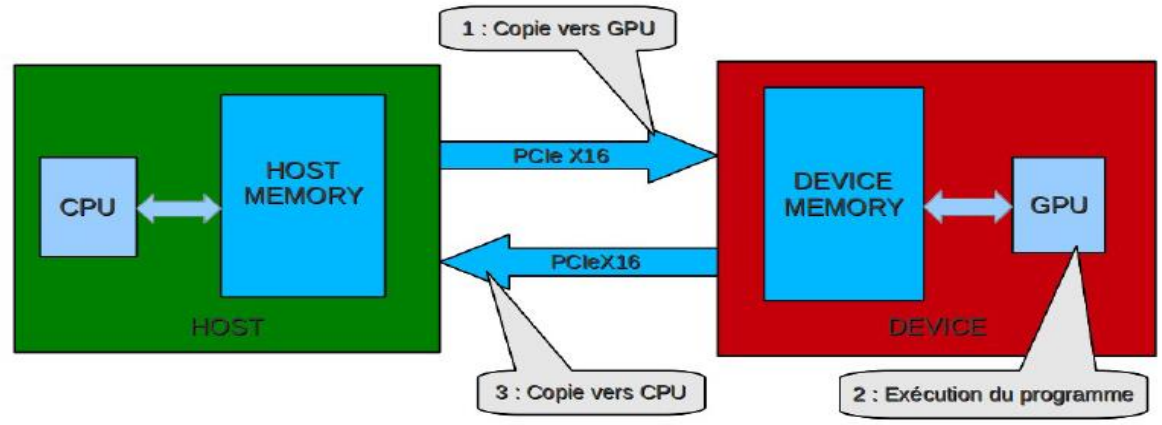


Figure 3-4 Communication CPU-GPU

1. Les données sont transférées de la mémoire du CPU vers la mémoire du GPU.
2. Le GPU traite les données.
3. Les données sont transférées de la mémoire du GPU vers la mémoire du CPU.

5. Programmation sur les GPU.

Nvidia a mis en disposition un langage et un logiciel de haut niveau appelé Compute Unified Device Architecture (CUDA) vu comme une sur-couche au C. Cet outil accompagné d'une documentation détaillée permet d'exploiter les capacités du calcul des GPUs, en se basant sur le principe de programmation parallèle en utilisant des threads. Il se présente comme une API considérée comme une extension du langage et

est composé de trois modules :

1. une interface pour l'application des programmations 'API',
2. un runtime qui joue le rôle intermédiaire entre l'utilisateur et
3. le driver et enfin une librairie pour les outils mathématiques.

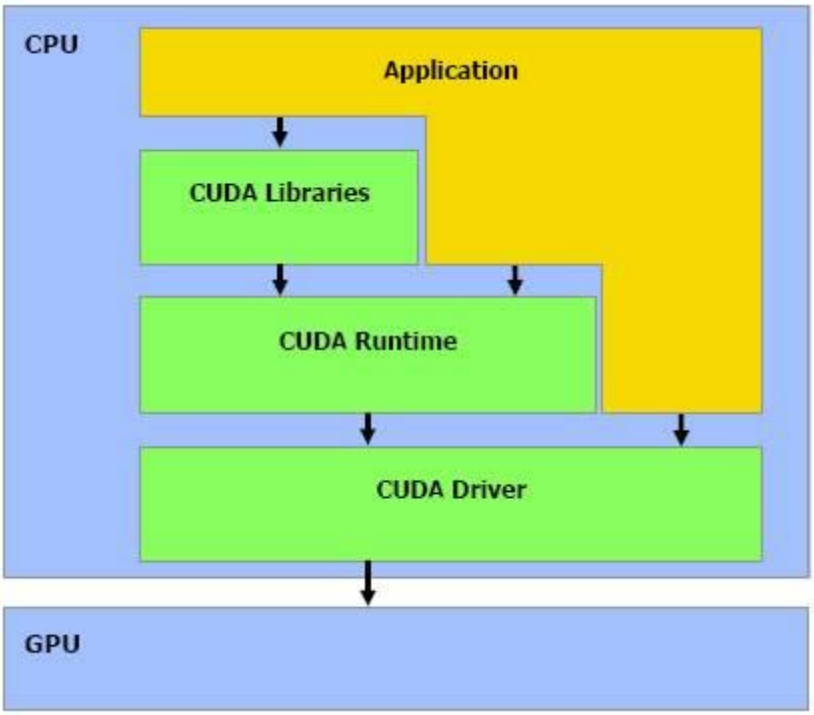


Figure 3-5 L'architecture de CUDA

CUDA propose une boite à outils comprenant un compilateur, des librairies et des outils d'optimisation et de débogage. L'Api de CUDA permet de spécifier les instructions du programme à exécuter sur le GPU. Et se charge automatiquement de créer les threads sur le GPU et de contrôler leurs exécutions.

Porter une application pour l'exécuter sur GPU en utilisant CUDA implique d'utiliser tous ces outils. Il faut initialiser l'environnement CUDA, distinguer les portions de code parallélisables et compiler l'application avec le compilateur CUDA.
Généralement, l'exécution d'un programme CUDA, se traduit par le lancement d'une ou plusieurs fonctions dites « noyau » destinées à s'exécuter sur la carte graphique et non sur les cœurs CPU de la machine. Ces fonctions sont en charge de définir le nombre de threads souhaités pour l'exécution. Dès qu'une fonction noyau est lancée, une organisation de threads se met en place pour satisfaire la demande de calcul. Cette organisation se traduit par une grille contenant souvent des milliers voire des millions de threads.

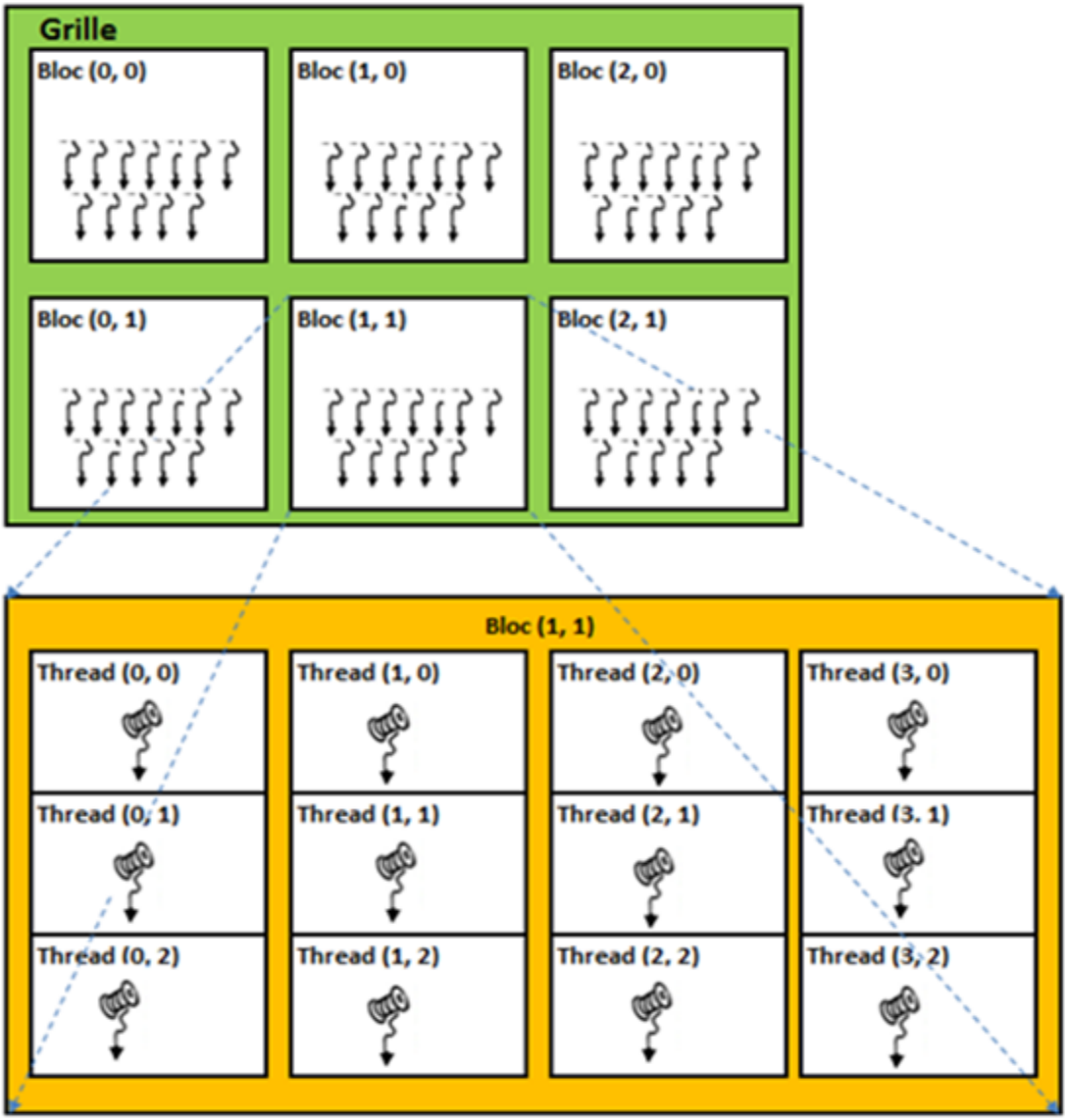


Figure 3-6 Organisation de threads

Une fois qu'un programme qui réalise un calcul sur GPU est lancé, un thread système par défaut est lancé pour démarrer ce programme. Un thread CPU est à la fois nécessaire pour exécuter le programme, mais aussi pour héberger le code noyau CUDA, avant son transfert sur la carte graphique. Ce code noyau sera exécuté par de nombreux threads GPU, qui participeront massivement à la réduction temps de calcul. Lorsque nous parlons de code GPU, nous parlons d'un code qui s'exécute sur l'architecture **CUDA**. En effet la programmation GPU distingue deux modes d'exécution : **host** et **kernel**. Le mode host, est le mode d'exécution par défaut de tout programme, mais la programmation GPU permet d'exécuter votre algorithme sur votre carte graphique. L**a figure suivante** clarifie l'exécution d'un programme **CUDA.**

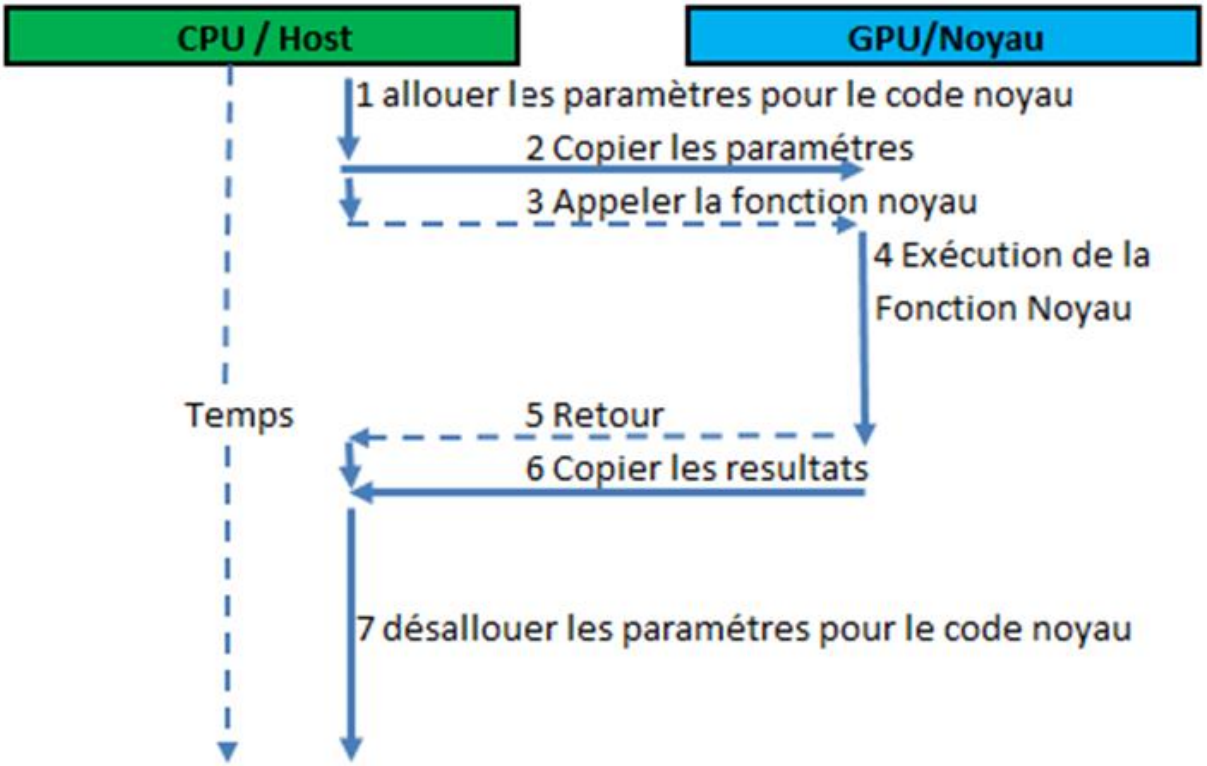
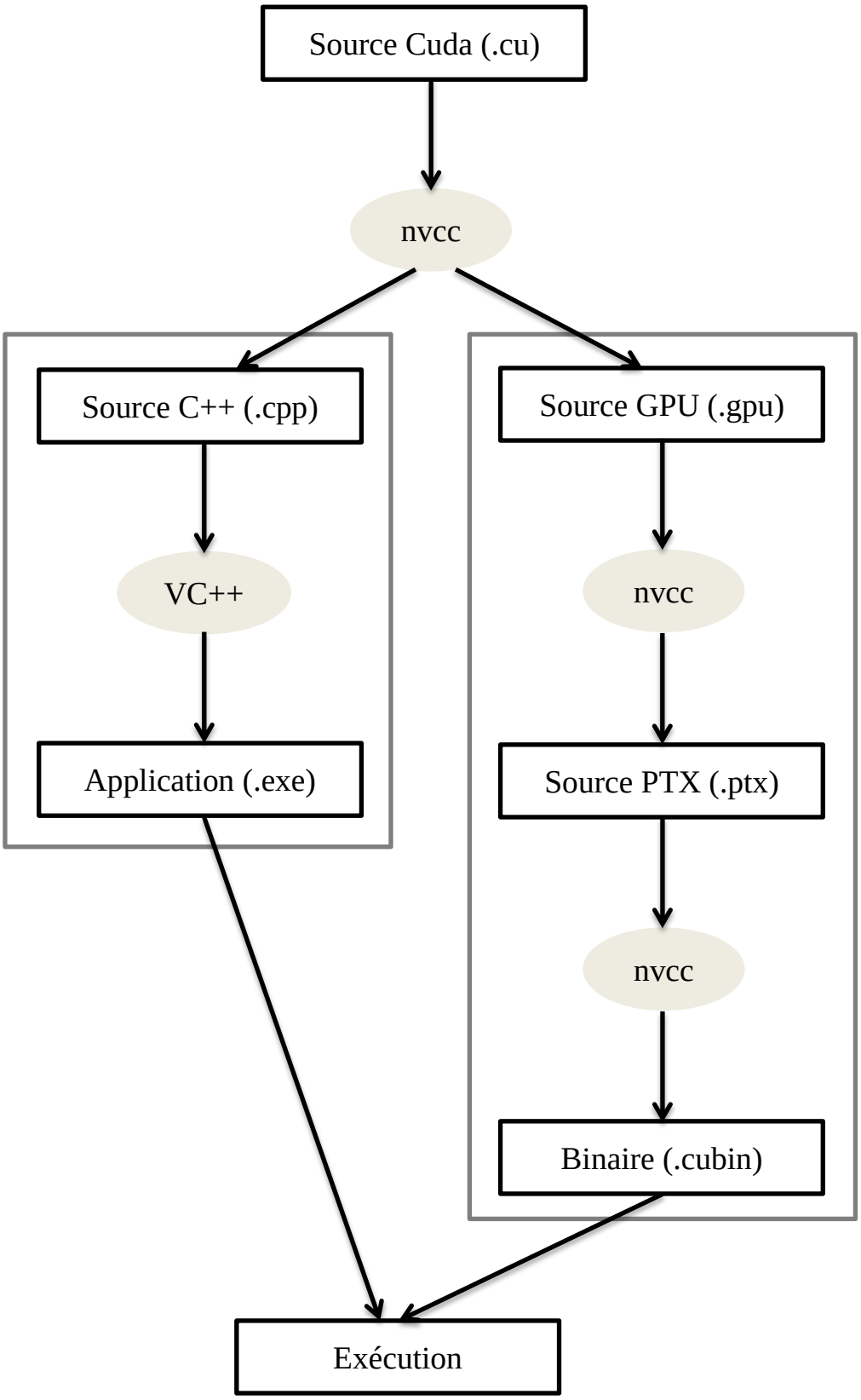


Figure 3-7 Protocole de communication pour un appel noyau, entre le mode CPU/Host et le mode GPU/Noyau

La transmission des paramètres à la fonction noyau, doit passer par des variables dont l'allocation mémoire est issue de la carte graphique.

Le schéma suivent montre le déroulement de la compilation du programme depuis le code source .cu jusqu'à l'exécution



Conclusion

Dans ce chapitre, nous avons présenté un cours survol de la programmation des cartes graphiques et nous avons évoqué la possibilité d'utiliser les cartes graphiques pour des calculs scientifiques parallèles comme nous avons présenté l'environnement CUDA qui permet la programmation de ces cartes, Celle du constructeur NVIDEA en particulier. Il est à noter que les architectures CPU et GPU sont fortement différentes et les conséquences pour les développeurs sont importantes. Car à fin que ces cartes soient exploitées d'une maniéré efficace, l'optimisation des programmes et le découpage en unités de calculs des algorithmes doit être soigneusement pris en considération. Après cette courte introduction à la programmation des GPU, on est en mesure d'entamer la phase d'implémentation de la version parallèle de l'algorithme des plus proches voisins.

Chapitre IV
Implémentation

1. Configuration matériel/logiciel

Nous avons implémenté notre application sous l'environnement de programmation CUDA-C version 7.5 sous Linux Ubuntu 12.5 64bits. Ce choix est motivé par le fait que cet environnement est performent et inclut toutes les fonctionnalités de développement et toutes les technologies liées à la programmation parallèle. Déplus il est supporté par notre carte graphique GTX 610. La configuration de notre machine hôte comporte :

Un CPU intel I3, à 2.40 GHz, 4 Go de RAM et 500Go de disque dur, doté d'une carte graphique GTX 610 à 48 Cœurs et 1 Go De RAM.

2. Données Utiliser

Dans notre travaille on a utilisé la base KDD'99 qui a été créé par le laboratoire Lincoln sous contrat avec DARPA en 1998. Et est devenu une table standard pour l'évaluation et la comparaison des performances des systèmes de détection d'intrusion ; et fut utiliser par la majorité des concepteurs ou chercheurs dans le monde entier. Cette base est divisée en deux table une pour le test et l'autre contenant seulement 10% des enregistrements pour l'apprentissage. Chaque enregistrement correspond une connexion réseau qui peut être soient une connexion normale ou intrusive. Chaque connexion est décrite avec 41 attributs dont 7 sont qualitatifs. Ces attributs sont classés dans trois catégories présentées dans les tableaux : 1, 2 et 3.[20]

nom de l’attribut	la description	type
Duration	longueur (nombre de secondes) de la connexion	continu
protocol_type	type de protocole, par exemple tcp, udp, etc.	discret
Service	service de réseau sur la destination, par exemple, http, telnet, etc.	discret
src_bytes	nombre d'octets de données de la source à la destination	continu
dst_bytes	nombre d'octets de données de la destination à la source	continu
Flag	état normal ou d'une erreur de la connexion	discret
Land	1 si la connexion est de / vers le même hôte / port; 0 sinon	discret
wrong_fragment	nombre de " fragments mal ``	continu
Urgent	nombre de paquets urgents	continu

Tableau 1 Caractéristiques de base de connexions TCP individuels.

nom de l'attribut	*la description*	*type*
hot	nombre de " indicateurs `` hot	continu
num_failed_logins	nombre de tentatives de connexion infructueuses	continu
logged_in	1 si connecté avec succès; 0 sinon	discret
num_compromised	nombre de " conditions `` compromis	continu
root_shell	1 si shell root est obtenue; 0 sinon	discret
su_attempted	1 si `` su root 'commande' tenté; 0 sinon	discret
num_root	nombre de `root " accès`	continu
num_file_creations	nombre d'opérations de création de fichier	continu
num_shells	nombre d'invites shell	continu
num_access_files	nombre d'opérations sur les fichiers de contrôle d'accès	continu
num_outbound_cmds	nombre de commandes sortantes dans une session de ftp	continu
is_hot_login	1 si la connexion appartient à la liste `` hot "; 0 sinon	discret
is_guest_login	1 si la connexion est un `` guest"login; 0 sinon	discret

Tableau 2 caractéristiques de contenu au sein d'une connexion proposée par la connaissance du domaine.

nom de l'attribut	*la description*	*Type*
Count	nombre de connexions à la même hôte que la connexion en cours dans les deux dernières secondes	Continu
serror_rate	% Des connexions qui ont des erreurs `` SYN "	Continu
rerror_rate	% Des connexions qui ont des erreurs `` REJ "	Continu
same_srv_rate	% Des connexions au même service	Continu
diff_srv_rate	% Des connexions à différents services	Continu
srv_count	nombre de connexions au même service que la connexion en cours dans les deux dernières secondes	Continu
srv_serror_rate	% Des connexions qui ont des erreurs `` SYN "	Continu
srv_rerror_rate	% Des connexions qui ont des erreurs `` REJ "	Continu
srv_diff_host_rate	% Des connexions à différents hôtes	Continu

Tableau 3 Trafic caractéristiques calculées à l'aide de deux secondes fenêtres de temps.

Dans le cadre de notre travail, nous n'avons considéré que deux classe de comportement à savoir comportement normal et comportement malveillant. L'ensemble de jeux de données qu'on a utilisé compte 494 021 dont 972 781 correspond au trafic normal. Cette table a été soumise à une phase de prétraitement qui a consisté, d'une part à supprimer les lignes en double et les colonnes nulles et à convertir les attributs non numériques en attributs numériques.

Mesures de Performances

Pour évaluer notre système de classification nous utilisant essentiellement les quatre mesures suivantes (Tableau 6):

- Faux Positif (FP) : Le nombre d'élément attribués à une catégorie inconvenablement. (élément attribués à des mauvaises catégories)
- Faux Négatif (FN) : Le nombre d'élément inconvenablement non attribués. (Qui auraient dû être attribués à une catégorie mais qui ne l'ont pas été).
- Vrai Négatif(VN) : Le nombre d'élément non attribués à une catégorie convenablement (Qui n'ont pas à être attribués à une catégorie, et ne l'ont pas été)

Catégorie Ci		Jugement d'expert	
		Normal	Anomalie
Jugement du classifieur	Normal	VP	FP
	Anomalie	FN	VN

Tableau 4 Matrice de contingence

A partir de ces quatre mesures, on peut calcule d'autres indicateurs tel que la Précision et Rappel qui sont calculés comme suite :

- La précision est la proportion d'élément correctement classés parmi ceux classés par le système dans la classe C_i.

$$Précision(C_i) = \frac{Nombre\_d'élément\ \_bien\_classes\_dans\_C_i}{Nombre\_d'élément\ \_classes\_dans\_C_i}$$

Pour le cas de détection d'intrusion on a

$$Précision(C_i) = \frac{VP}{VP \quad + FP}$$

Le rappel d'une classe C_i est défini comme étant la proportion des vecteurs classés par le système dans la classe C_i sur le nombre d'éléments de la même classe.

$$Rappel(C_i) = \frac{Nombre - d'élément - bien - classes - dans - C_i}{Nombre - d'élément - de - la - classe - C_i}$$

Pour la détection d'intrusion on a

$$Rappel(C_i) = \frac{VP}{VP \quad + FN}$$

En plus de ces facteurs on a aussi le Bruit et le silence. Le bruit est le pourcentage des éléments incorrectement associés à une classe par le système : [29]

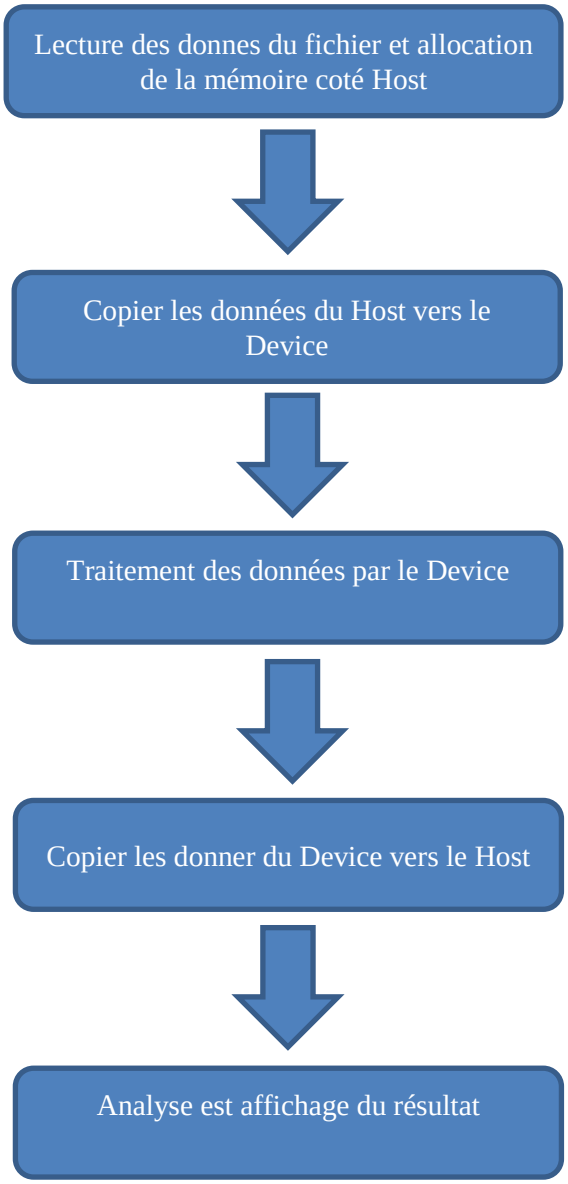
$$Bruit = 1 - Précision(P)$$

La notion de silence est le point de vue opposé du rappel. Le silence est le pourcentage d'élément à associer à une classe incorrectement non classés par le système : [12]

$$Silence(S) \ = 1 - Rappel(R)$$

3. Implémentation de l'algorithme

On rappelle que l'exécution d'un programme par une GPU nécessite cinq phases illustrées par le schéma suivant :



- Initialement notre programme lit les données d'apprentissage et du test à partir deux fichiers text(respectivement apprent.txt et test.txt) et les stock dans deux matrices. Pour faciliter la lecture et puis le traitement de ces données on a stocké les étiquettes de classes (0 pour la classe normale et 1 pour une attaque) dans des vecteurs séparés.

```
for (i = 0; i < nline; i++)
{
 for (j = 0; j < nvar; j++)  fscanf(file, "%f",&mat[i*nvar+j]);
 fscanf(file, "%s",mot);
 if (strcmp(mot,"normal.")==0) k=0; else k=1;
 Classe[i]=k;
}
```

Portion de programme pour la lecture des données d'apprentissage

```
for (j = 0; j < nvar; j++) fscanf(file, "%f",&v[j]);
fscanf(file, "%s",mot);
if (strcmp(mot,"normal.")==0) ki=0; else ki=1;
```

Portion de programme pour la lecture des données de test

A prés avoir lu les données on passe à la phase de transfert de ces matrice vert la mémoire du GPU

Les données envoyées vers le Device sont :

- les vecteurs d'apprentissage.
- les vecteurs de test.
- le nombre de vecteur dans la matrice d'apprentissage.
- le nombre de colonnes de deux matrice qui correspond au nombre d'attributs utilisés.

L'allocation de mémoire dans la GPU se fait avec la fonction : **cudaMalloc**

Et la copie de la mémoire du CPU vers la mémoire du GPU avec la fonction : **cudaMemcpy**

Une fois les données transférées on passe au calcule du nombre de vecteur par bloc comme suite :

$$Nombre\ de\ vecteur\ par\ block = \frac{nombre\ de\ vecteur}{nombre\ de\ cors\ utiliser}$$

Puis on appel notre fonction Kernel, qui calcule le vecteur distance pour chaque donnée de test et la classe dans la classe majoritaire de ces plus proche voisins. Comme illustré dans la portion de code suivante :

```
int n_blocks = 48;
int block_size = nline/n_blocks  + (nline%n_blocks  == 0 ? 0:1);
RowSum<<< n_blocks, block_size >>>(d_B,d_v,d_result,nline,nvar);
```

La distance qu'on a utilisé et la distance euclidienne donnée par :

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(xi - yi)^2}$$

La portion de code suivant représente le corps de la fonction kernel qui sera executée en parallèle sur les 48 cœurs de notre carte graphique

```
__global__ void RowSum(float *c, float *v, float *s, int nrow, int ncol)
{
        int rowIdx = blockIdx.x *blockDim.x+ threadIdx.x;
        int k;
        if (rowIdx < nrow)
                {
                float sum = 0;
                for (k = 0 ; k < ncol ; k++) sum += ( c[rowIdx*ncol+k] -
                v[k] ) * ( c[rowIdx*ncol+k] - v[k]);
                s[rowIdx] = sqrt(sum);
}
}
```

Pour chaque donnée, un vecteur de distance entre cette donnée et les donnée d'apprentissage est calculé et doit être trier à fin d'extraire les indice des voisin les plus proche. Or les algorithme de trie conventionnels consomment beaucoup de temps, c'est pour cela qu'on a choisie d'implémenter une version moins couteuse en temps.

```
For (j = 0 ; j < knn ; j++) idt[j]= dst[j];
   for (j = knn ; j < nline ; j++)
  {
  If (dst[j] < idt[maxi(idt,knn)])
    {
     idt[maxi(idt,knn)]=dst[j];
     index[maxi(idt,knn)]=j;
    }
  }
```

Notre programme a fournis les résultats présentés dans le tableau 5 et 6.

	Pourcentage %
VP	99,957
FP	0,043
VN	99,889
FN	0,111

Tableau 5 les résultats de la détection d'intrusion

Mesure	Rappelle	Précision	Bruit	Silence
Résultat	0.99972459	0.999574847	0.000425153	0.00027541

Tableau 6 les résultats de la détection d'intrusion

Implémentation	Parallèle	Séquentielle
Résultat	34H 55M 03S	30J 20H 02M 05S

Tableau 7 Les résultats des temps d'exécution

Les résultats obtenus montrent l'efficacité de l'algorithme KNN dans la classification et en particulier dans la détection d'intrusion vu qu'il nous a fourni des taux avoisinant 99% de vrai positif et de vrai négatif avec un faible taux de faux alerte. Ce fait a été déjà constaté dans les différents travaux de recherche ayant utilisé l'algorithme des k plus proches voisins dans les problèmes de classification supervisée notamment en détection d'intrusion.

Le but principal de notre projet consistait à paralléliser cet algorithme sur une carte graphique à plusieurs cœurs à fin de lever la principale lacune de cet algorithme qui réside dans sa lenteur et son grand besoin d'espace mémoire.

Dans cette section, nous montrons les performances en termes de temps de calcul de notre algorithme. L'étape la plus coûteuse dans l'algorithme des KPPV est l'étape de calcul des distances. Le reste des calculs est négligeable en termes de temps de calcul. Nous nous sommes orientés vers une implémentation GPU et nous avons délibérément opté d'étudier en détail les différents points d'optimisation permettant d'améliorer le temps de calcul sur GPU notamment en optimisant l'algorithme de tri nécessaire dans la procédure de détection des plus proches voisins. Les comparaisons en terme de temps de calcul sur CPU et sur GPU sont très révélatrices ; puis que notre version parallèle de l'algorithme des plus proches voisin à nécessité que quel que heurs alors que la version séquentielle à demander plusieurs semaines. Nous pensons qu'avec une carte graphique plus puissante, telle que la GTX-750 qui est dotée de 512 cœurs, le temps d'exécution consommé par notre programme serait encore plus court ce qui nous permet de prétendre à une détection d'intrusion en temps réelle

.

4. Conclusion

Dans ce chapitre nous avons présenté l'ossature générale de notre application qui consiste à implémenter une version parallèle de l'algorithme des plus proches voisins que nous avons appliqué à la détection d'intrusion. Les résultats obtenus confirment d'une part, l'efficacité de cet algorithme et l'importante réduction de temps d'exécution due à la parallélisassions sur une modeste carte graphique d'autre part. l'utilisation d'une carte graphique plus professionnelle aurait réduit l' temps d'exécution en quelque secondes.

Conclusion Générale.

Ce Mémoire nous a donné l'opportunité de découvrir le mode de programmation parallèle que nous croyons difficile et nécessite de grand matériel ou plusieurs machines puissantes et couteuses. Mais nous avons découvert qu'une simple carte graphique qu'on avait l'habitude de l'utiliser pour nos jeux vidéo était humblement suffisante pour réduire le temps d'exécution de l'algorithme des plus proche voisins à un facteur très significatif par rapport à une brute mise en œuvre à base de CPU. Cette amélioration peut avoir un impact significatif en fonction du contenu des applications qui utilisent cet algorithme. Aussi, on croit fortement que l'approfondissement de notre maitrise du langage CUDA-C et les différentes API dont il dispose pourrait améliorer davantage le facteur de réduction de temps d'exécution.

Bibliographie

[1] J.P.Anderson. Computer security threat monitoring and surveillance, James P. Anderson Co., Fort Washington, PA ,1980.

[2] DR. Barbara Chapman, Center for Advanced Computing & Data Systems, University of Houston, September 2014

[3] N. Baudoin et Marion Karle , « NT Réseaux : IDS et IPS » ,Ingénieu 2000.2003-2004.

[4] P. Biondi , « Architecture expérimentale pour la détection d'intrusions dans un système informatique » ,Avril-Septembre 2001.

[5] G. BOURKACHE, Un IDS réparti basé sur une société d'agents intelligents, Université M'hamed BOUGARA de BOUMERDES,thèse magistère, 2006.

[6] J. Callut, «Implémentation efficace des Support vector Machines pour la classification» Mémoire présenté en vue de l'‟obtention du grade de Maître en informatique. Université libre de Bruxelles . département informatique, 2003.

[7] M. Cédric, Langage de description d'attaques pour la détection d'intrusion par corrélation d'événement ou d'alerte en environnement réseau hétérogène, thèse de doctorat, université de Rennes, 2003.

[8] A. Cornuéjols, L. Miclet, Y.Kodratoff, « Apprentissage Artificiel, Concepts et algorithmes » ISBN 2-212-11020-0 , 2002.

[9] H.Debar, M.Dacier et A.Wespi , « A revised taxonomy for intrusion detection systems » ,Annales des télécommunications. July-August 2000.

[10] D.E. Denning, An Intrusion-Detection Model, IEEE transactions on software engineering, SE-13(2), p 222-232,1987.

[11] R. O. Duda and P. E. Hart , "Pattern Classification and Scene Analysis". John Wiley & Sons, 1973.

[12] M. Hocine, Classification Automatique de Textes Approche Orientée Agent, Thèse de Magister de l'Université de Aboubekr Belkaid Telemcen, 2011.

[13] J. Holland, Adaptation In Natural And Artificial Systems, University of Michigan Press, 1975

[14] M.Ludovic, « Les machines à vecteurs support pour la classification en imagerie Hyperspectrale , implémentation et mise en oeuvre », rapport d'‟épreuve test du conservatoire national des arts et métiers, Grenoble , 2010.

[15] B. Mukherjee ; California Univ., Davis, CA, USA ; L. T. Heberlein ; K. N. Levitt « Network Intrusion Detection»,IEEE Network, Vol 8, No 3, pp .26-41, 1994.

[16] V. Nicollet, « Pricing de Produits Dérivés Financiers par la Méthode de Monte-Carlo sur des Processeurs Graphiques de Dernière Génération », Rapport de stage du Master Parisien de Recherche en Informatique, 2e année, 20 novembre 2007.

[17] K. Price, « Intrusion Detection Pages »,Purdue University, 1998.

[18] B. Scholkopf and A. Smola, "Leaning with Kernels". MIT Press, 2001.

[19] S. J. Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K. Chan. JAM Project Cost-based Modeling and Evaluation for Data Mining With Application to Fraud and Intrusion Detection. 1999.

[20] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set".

[21] V. Vapnik, "The Nature of Statistical Learning Theory". Springer Verlag, New York, USA, 1995.

[22] P.Vincent, « Modèles à noyaux à structure locale », Thèse de Phd en informatique, Université de Montréal, 2003.

[23] B. White & E. A. Fisch, & U. W. Pooch, « Cooperating Security Managers: A Peer- Based Intrusion Detection System »,IEEE Network Journal, pp. 20-23, January/February 1996.

Résumé

Un **processeur graphique**, ou **GPU** (de l'anglais *Graphics Processing Unit*), est un circuit intégré la plupart du temps présent sur une carte graphique (mais pouvant aussi être intégrée sur une carte-mère ou dans un CPU) et assurant les fonctions de calcul de l'affichage. Un processeur graphique a généralement une structure hautement parallèle qui le rend efficace pour une large palette de tâches graphiques comme le rendu 3D, en Direct3D, en OpenGL, la gestion de la mémoire vidéo, le traitement du signal vidéo, … Dans le cadre de ce mémoire nous avons développé une version parallèle de l'algorithme des plus proches voisins qu'on appliquera au problème de détection d'intrusion. L'exécution de cette version sur une modeste carte graphique la GT-610 qui était suffisante pour réduire le temps d'exécution de cet algorithme à un facteur très significatif par rapport à une brute mise en œuvre à base de CPU.

Abstract

A graphics processing unit (GPU) is an integrated circuit mostly present on a graphics card (but can also be integrated on a motherboard or CPU), and designed to provide functions calculation of the display. A graphics processor usually has a highly parallel structure, which makes it effective for a wide range of graphics tasks such as 3D rendering in Direct3D, OpenGL, and management of video memory… This thesis concerns the development of a parallel version of the nearest neighbor's algorithm which we applied in to the intrusion detection problem.

Running this version on a modest graphics card GT-610 was sufficient to reduce the execution time of this algorithm with a very significant factor.

ملخص

وحدة معالجة الرسومات هو عبارة عن دائرة متكاملة موجودة في الغالب على بطاقة الرسومات (ولكن يمكن أيضا أن تكون مدمجة على اللوحة الأم أو وحدة المعالجة المركزية)، ويساهم في توفير وظائف حساب العرض. معالج الرسومات عادة لديه بنية موازية للغاية مما يؤهله فعالا لمجموعة واسعة من المهام مثل D3 و Direct3D، برنامج OpenGL، وإدارة ذاكرة الفيديو، ومعالجة الإشارات الفيديو، ... في إطار ه هذه الرسالة قمنا بتطوير نسخة متوازية لخوارزمية « الجيران الأقرب" كما قمنا بتطبيق هده النسخة لحل مشكلة كشف التسلل.

إن برمجة هده النسخة و تنفيذها بطاقة الرسومات GT-610 كان كافيا لتقليص المدة الزمنية اللازمة للنسخة التعاقبية بشكل كبير.