

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

polytech.univ-tours.fr

Projet ASR

2017-2018

Configuration d'un OS pour le calcul parallèle

Tuteur académique

Patrick MARTINEAU

Étudiants

Benjamin CALDAS (DI5)

Logan VERECQUE (DI5)

14 janvier 2018



Liste des intervenants

Nom	Email	Qualité
Benjamin CALDAS	benjamin.caldas@etu.univ-tours.fr	Étudiant DI5
Logan VERECQUE	logan.verecque@etu.univ-tours.fr	Étudiant DI5
Patrick MARTINEAU	patrick.martineau@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Benjamin Caldas et Logan Verecque susnommés les auteurs.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Patrick Martineau susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Benjamin Caldas et Logan Verecque, *Configuration d'un OS pour le calcul parallèle*, Projet ASR, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={Caldas, Benjamin and Verecque, Logan},
  title={Configuration d'un OS pour le calcul parallèle},
  type={Projet ASR},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2017-2018}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
I Introduction	1
1 Contexte.....	1
2 Sujet	1
3 Objectifs	2
II Définition du sujet	3
4 Système d'exploitation	3
5 Clé USB bootable	4
6 OpenMP	4
7 Cuda.....	4
8 MPI.....	5
III Solution envisagée	6
9 Principe.....	6
10 Choix du système d'exploitation.....	6

10.1	Recherche	6
10.2	Lubuntu	7
IV	Étapes de réalisation	8
11	Création de la clé bootable	8
11.1	Création de l'environnement	8
11.2	Partitionnement de la clé USB et Installation du SE	8
12	Gcc	9
12.1	Installation	9
12.2	Validation OpenMP	9
13	Cuda	11
13.1	Vérifications	11
13.2	Installation Cuda	12
13.3	Installation nvcc	12
13.4	Validation	12
14	MPI	13
14.1	Installation	13
14.2	Validation mono-poste	14
14.3	Validation sur réseau	15
V	Gestion de projet	16
15	Organisation	16
16	Planning	16
17	Gestion de versions	17
VI	Problèmes et difficultés rencontrés	18
18	Connexion réseau de Polytech	18
19	Temps des installations	18
20	Secure Boot	18
21	Mode Live et mode persistant	19
22	Limite de stockage	19
23	Fragilité	19
VII	Outils utilisés	20
24	Linux Live Creator	20
25	USB Image Tool	21
26	Etcher	22
27	Diskpart	23

VIII	Guide d'utilisation	24
IX	Guide d'installation	26
X	Guide du développeur	29
XI	Conclusion	30

Table des figures

Table des figures

1	Énoncé exact du sujet	1
2	Le système d'exploitation dans l'architecture d'un ordinateur.....	3
3	Logo OpenMP	4
4	Logo NVidia Cuda	5
5	Application utilisant MPI.....	5
6	Liste des SE compatible avec Cuda	7
7	OpenMP - Code de validation, compilation et exécution.....	10
8	OpenMP - Code 2 de validation, compilation et exécution.....	11
9	Cuda - Code SAXPY	12
10	C - Code SAXPY.....	13
11	Cuda vs C - Compilation et exécution codes SAXPY.....	13
12	MPI - Code de validation mono-poste, compilation et exécution	14
13	Tâches identifiées au début du projet	16
14	Planning du projet.....	17
15	Versions des images	17
16	Linux live USB creator.....	21
17	USB image tool	22
18	Etcher	22
19	Diskpart - Exécution et liste des commandes.....	23
20	Vérification de la carte graphique - Code	24
21	Vérification de la carte graphique - OK	25
22	Vérification de la carte graphique - KO	25
23	Sauvegarde de l'image disque avec USB Image Tools.....	26
24	Re-partitionnement de la clé USB avec Diskpart.....	27

25	Gravure de l'image disque avec Etcher	28
----	---	----

Première partie

Introduction

1 Contexte

Dans le cadre de notre cursus nous avons l'occasion de nous spécialiser en choisissant la voie des Systèmes d'Informations ou la voie d'Architecture, Système et Réseaux. Ce document présente le projet d'Architecture, Système et Réseaux réalisé par le binôme Benjamin Caldas / Logan Verecque finalisant la spécialisation ASR au sein de la 5ème année du diplôme d'Ingénieur en Informatique de l'école Polytechnique de Tours.

2 Sujet

La figure suivante contient l'énoncé exact de notre sujet.

Sujet ASR #1 : Mise en place d'une infrastructure de calcul parallèle - P. Martineau

On propose d'intégrer les différents outils de calcul parallèle dans un OS minimal qui pourrait être porté sur une clef USB. Cuda permet de profiter du parallélisme sur les GPU OpenMP permet de profiter du parallélisme entre les core d'un CPU MPI permet de profiter du parallélisme entre ordinateurs au sein d'un réseau.

Les trois outils sont complémentaires. On souhaite donc :

- prendre en main chaque outil
 - intégrer leur utilisation et leur complémentarité (intégration) au sein d'un unique OS.
 - prévoir de faciliter le déploiement de cet OS sur un ensemble de machines équipées ou non du matériel nécessaire pour mettre en œuvre chaque niveau de parallélisme
- > le but est de rendre indépendant le développement de l'exécution des processus. Cette exécution bénéficiant ou non de la parallélisation.

Figure 1 – Énoncé exact du sujet

3 Objectifs

L'objectif de notre projet était de mettre en place un système d'exploitation Unix contenant tous les outils permettant de réaliser du développement suivant les trois grands axes de parallélisations :

- multi-processeurs (ou CPU) avec OpenMP
- sur processeurs graphiques (ou GPU) avec Nvidia Cuda
- sur différents postes d'un réseau avec MPI

Ce système devait également être portable et bootable depuis n'importe quel clé usb. Cette contrainte indique également que le système doit être léger, simple d'utilisation et d'installation.

Deuxième partie

Définition du sujet

4 Système d'exploitation

Un système d'exploitation (SE), ou Operating System (OS) en anglais est un ensemble de programmes qui pilote tous les composants de l'ordinateur. Il dirige l'utilisation des ressources par les applications. Les ressources de stockage, de calcul et de communication sont donc gérées par le SE et distribuées aux différents logiciels. Le SE assure donc le lien entre le matériel informatique et les applications utilisées ou non par les utilisateurs. Ces différentes interactions sont illustrées dans la figure suivante :

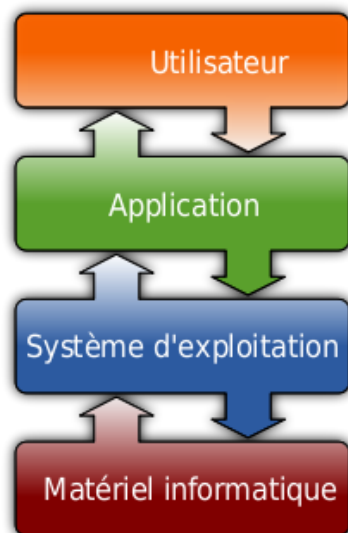


Figure 2 – Le système d'exploitation dans l'architecture d'un ordinateur

Les principaux systèmes d'exploitations sont Windows, Mac OS et les distributions de type Linux. Nous opterons pour une distribution de ce dernier car leur principal avantage est que tout est paramétrable facilement et rapidement. Ceci nous sera très utile dans le cadre de notre projet d'architecture systèmes et réseaux.

5 Clé USB bootable

Une clé USB bootable est une clé USB qui intègre un système capable de faire démarrer un système d'exploitation sans faire appel à un autre élément de stockage. Cela permet notamment d'installer un SE sur un ordinateur sans avoir le CD d'installation ou de pouvoir utiliser une distribution particulière depuis n'importe quelle machine.

Le GRUB ou (GRand Unified Bootloader) est un logiciel permettant de charger un système d'exploitation. Il permet d'amorcer les systèmes de la norme POSIX dont Linux. Ce logiciel joue donc un rôle important dans la création d'une clé USB bootable embarquant un SE de type Linux.

6 OpenMP

OpenMP (Open Multi-Processing) est une interface de programmation pour le calcul parallèle sur architecture à mémoire partagée. Cette API est multiplateforme (GNU/Linux, OS X et Windows) pour les langages de programmation C, C++ et Fortran. OpenMP se présente sous la forme d'un ensemble de directives, d'une bibliothèque logicielle et de variables d'environnement. De plus, OpenMP est portable et dimensionnable.

OpenMP permet donc de développer rapidement des applications parallèles à petite granularité en restant proche du code séquentiel.

Par ailleurs, il est également possible de réaliser de la programmation parallèle hybride en utilisant à la fois OpenMP et MPI.



Figure 3 – Logo OpenMP

7 Cuda

CUDA (Compute Unified Device Architecture) est une technologie en GPGPU, c'est-à-dire utilisant un processeur graphique (GPU) pour exécuter des calculs à la place du processeur (CPU). L'idée est donc de permettre de « soulager » le processeur en utilisant le processeur graphique pour effectuer un maximum de calcul en parallèle de celui-ci.

CUDA permet de programmer des GPU en C. Elle a été développée par NVidia pour ses cartes graphiques et garde donc l'exclusivité autour de cette marque.



Figure 4 – Logo NVidia Cuda

8 MPI

MPI (Message Passing Interface) est une bibliothèque de fonctions, conçue en 1993, permettant le calcul parallèle en C, C++ et en Fortran. Elle exploite les ordinateurs multiprocesseur, donc quasiment toutes les machines actuelles, qu'elles soient en réseau ou non. Elle s'appuie sur cette communication en réseau pour distribuer les calculs à effectuer entre toutes les machines du réseau par passage de messages.

Lorsqu'une application sera exécutée avec MPI, elle sera clonée suivant le nombre de processus spécifié et envoyée sur les différentes ressources de stockage et ceci par l'intermédiaire du réseau, comme le montre la figure suivante :

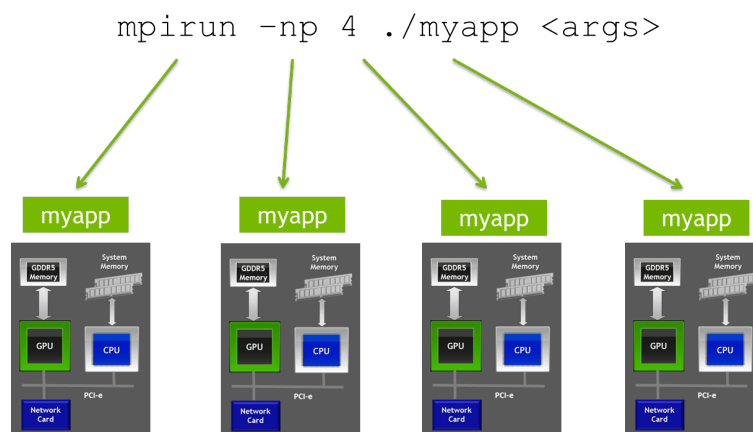


Figure 5 – Application utilisant MPI

MPI utilise le concept de communiqueurs. Un communiqueur est un ensemble de processus pouvant communiquer ensemble et cette communication sera possible seulement si ces processus sont dans le même communiqueur.

Troisième partie

Solution envisagée

9 Principe

L'idée est de s'appuyer sur un OS le plus léger possible pour être portable et bootable directement depuis une clé USB. Néanmoins, il nous faut un système capable de supporter les trois outils de parallélisations. Seulement, si OpenMP et MPI sont disponible et utilisable sur l'intégralité du monde Unix ce n'est pas le cas de Cuda qui se limite seulement à quelques système d'exploitation (cf. 10. Choix du système d'exploitation).

Une fois notre clé disponible et le système correctement installé il nous faut mettre en place et installer sur celle-ci nos trois outils de développement. Nous commencerons tout d'abord par OpenMP puis par CUDA et enfin par MPI. A noter que chaque phase d'installation fera intervenir une phase de tests afin de valider le bon fonctionnement de notre outil avant de passer à l'installation de la suivante.

Enfin, l'utilisation des trois outils est effective dans la limite des conditions suivantes :

- OpenMP : Toujours
- CUDA : Si une carte graphique NVIDIA est présente et disponible
- MPI : Utilisation multi-machines si un réseau de machines existe.

10 Choix du système d'exploitation

Cette section présente la démarche de sélection du système d'exploitation.

10.1 Recherche

Afin de choisir notre système d'exploitation nous avons cherché à trouver un système d'exploitation très léger capable de respecter les contraintes fixées pour ce projet. Parmi ces contraintes, la plus importante était évidemment d'être capable de supporter CUDA. Pour cela, NVIDIA a mis à disposition sur son site officiel la liste des systèmes capables de l'utiliser. Cette liste est la suivante :

Table 1. Native Linux Distribution Support in CUDA 9.1							
Distribution	Kernel	GCC	GLIBC	ICC	PGI	XLC	CLANG
x86_64							
RHEL 7.x	3.10	4.8.5	2.17	17.0	17.x	NO	4.0.0
RHEL 6.x	2.6.32	4.4.7	2.12				
CentOS 7.x	3.10	4.8.5	2.17				
CentOS 6.x	2.6.32	4.4.7	2.12				
Fedora 25	4.8.8	6.2.1	2.24-3				
OpenSUSE Leap 42.3	4.4.68	4.8.5	2.22				
SLES 12 SP3	4.4.21	4.8.5	2.22				
Ubuntu 17.04	4.9.0	6.3.0	2.24-3				
Ubuntu 16.04	4.4	5.3.1	2.23				
POWER8(*)							
RHEL 7.x	3.10	4.8.5	2.17	NO	17.x	13.1.6	4.0.0
Ubuntu 16.04	4.4	5.3.1	2.23	NO	17.x	13.1.6	4.0.0
POWER9(**)							
Ubuntu 16.04	4.4	5.3.1	2.23	NO	17.x	13.1.6	4.0.0
RHEL 7.4 IBM Power LE	4.11	4.8.5	2.17	NO	17.x	13.1.6	4.0.0

(*) Only the Tesla GP100 GPU is supported for CUDA 9.1 on POWER8.

(**) Only the Tesla GV100 GPU is supported for CUDA 9.1 on POWER9.

Figure 6 – Liste des SE compatible avec Cuda

Nous avons ensuite regardé de plus près chacun des systèmes et nous avons pu observer que le plus léger était Ubuntu. Néanmoins, il était encore possible d'aller encore plus loin et de s'intéresser à la famille de système d'exploitation Ubuntu elle-même.

10.2 Lubuntu

Notre choix s'est donc finalement porté sur Lubuntu pour plusieurs raisons importantes. Lubuntu est un projet de distribution GNU/Linux et une version dérivée d'Ubuntu. L'objectif de Lubuntu est d'être une version plus légère, plus économe en ressources matérielles et moins consommatrice en énergie qu'Ubuntu.

On notera que Lubuntu utilise pour cela l'environnement de bureau LXDE et le compilateur GCC n'est pas installé par défaut.

Quatrième partie

Étapes de réalisation

Dans cette partie, nous verrons quelles étaient les différentes étapes de réalisation du projet.

11 Création de la clé bootable

Il a fallut, dans un premier temps, créer la clé bootable. Ceci s'est fait en deux étapes, premièrement la création de clé bootable depuis laquelle on allait lancer l'installation et ensuite le partitionnement de la clé USB et l'installation du SE.

11.1 Création de l'environnement

Pour partitionner la clé USB et y installer un SE, il faut s'appuyer sur une distribution Linux. C'est pour ceci qu'il a fallut créer une clé Live Lubuntu. On a utilisé une clé USB de faible capacité ici, 4 Go suffisent. Nous avons utilisé l'outil Linux live USB creator en spécifiant un `.iso` Lubuntu. La création de la clé USB bootable en mode live a prit environ 15 minutes. Nous avons ensuite démarré notre machine sur cette clé USB live.

11.2 Partitionnement de la clé USB et Installation du SE

Après avoir booté sur la clé USB live tout juste créé, nous avons lancer l'utilitaire d'installation du SE Lubuntu. Nous avons précisé que nous souhaitions une installation personnalisée. Pour installer Lubuntu en version persistante dans le temps, il faut prévoir une clé de 16 Go minimum. Nous avons ensuite choisis le lecteur sur lequel effectué l'installation : notre clé USB puis nous avons spécifié de donner 4 Go pour le swap et le reste (environ 10 Go) pour les données en ext2. Nous avons choisis cette même clé USB pour l'installation du programme de démarrage, aussi appelé GRUB. L'installation a ensuite duré environ 2 heures. Après ceci, notre clé USB était fin prête.

12 Gcc

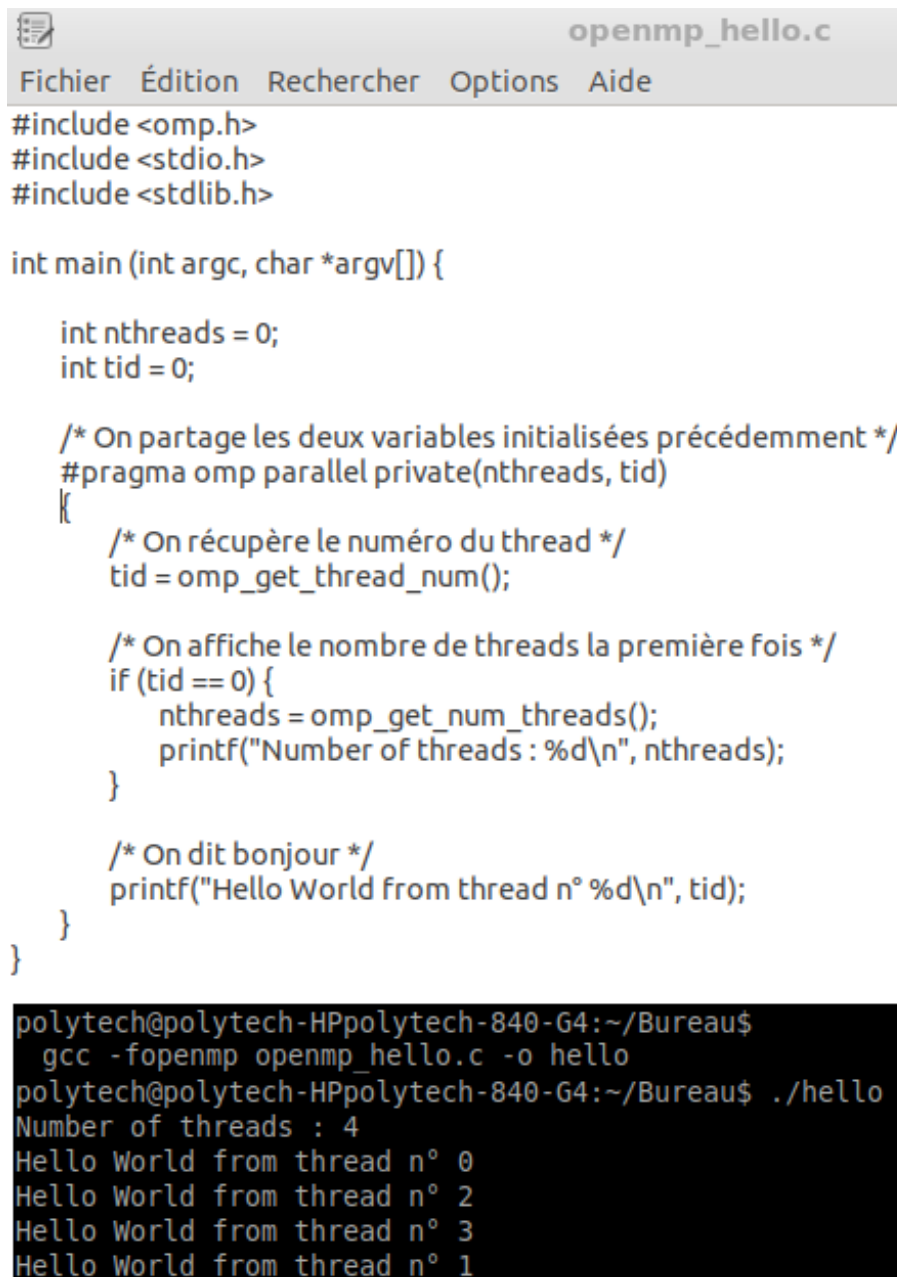
12.1 Installation

Le paquet gcc n'est pas présent nativement lorsque l'on installe Lubuntu sans les outils recommandés, il a donc fallu l'installer. Dans ce but, nous avons effectué une mise à jour des services *apt-get* puis nous avons installé le paquet gcc. L'installation de ce paquet d'environ 80 Mo prend un peu plus d'1 heure. Afin de tester si le paquet a bien été installé, on effectue une commande permettant de connaître la version de gcc. Si une version est affichée dans le terminal, le paquet a bien été installé.

12.2 Validation OpenMP

Lorsque gcc est installé, OpenMP est utilisable. On a donc codé un programme simple permettant de voir si la programmation multi-cœurs est bien effective.

La figure ci-dessous illustre le code que nous avons mis en œuvre afin de tester OpenMP. Celui-ci se contente de dire bonjour à l'utilisateur par le biais des différents threads qu'il a à sa disposition.



The image shows a code editor window titled "openmp_hello.c". The menu bar includes "Fichier", "Édition", "Rechercher", "Options", and "Aide". The code is as follows:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {

    int nthreads = 0;
    int tid = 0;

    /* On partage les deux variables initialisées précédemment */
    #pragma omp parallel private(nthreads, tid)
    {
        /* On récupère le numéro du thread */
        tid = omp_get_thread_num();

        /* On affiche le nombre de threads la première fois */
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads : %d\n", nthreads);
        }

        /* On dit bonjour */
        printf("Hello World from thread n° %d\n", tid);
    }
}
```

Below the code, a terminal window shows the compilation and execution:

```
polytech@polytech-HPpolytech-840-G4:~/Bureau$
gcc -fopenmp openmp_hello.c -o hello
polytech@polytech-HPpolytech-840-G4:~/Bureau$ ./hello
Number of threads : 4
Hello World from thread n° 0
Hello World from thread n° 2
Hello World from thread n° 3
Hello World from thread n° 1
```

Figure 7 – OpenMP - Code de validation, compilation et exécution

On peut remarquer que les threads ne mettent pas le même temps à exécuter le code et c'est pour cela que les affichages ne suivent pas l'ordre croissant des numéros de thread.

La figure ci-après illustre un exemple de code où l'on effectue un grand nombre d'opérations dans une boucle for partagée entre les différents threads.

```

openmp_test.c
Fichier  Édition  Recherche  Options  Aide
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

int main (int argc, char const *argv[]){

    /* On récupère le nombre de threads*/
    int nbThreads = 0;
    #pragma omp parallel
    {
        nbThreads = omp_get_num_threads();
    }
    printf("Nb threads : %d \n", nbThreads);

    /* On initialise le tableau des résultats */
    int nb_op_thread[nbThreads];
    for (int i=0;i<nbThreads;i++) nb_op_thread[i] = 0;

    /* On effectue 10 000 opérations */
    #pragma omp parallel for
        for (int n=0;n<9999;n++) {
            for (int i=0; i<nbThreads;i++)
                if (omp_get_thread_num() == i)
                    nb_op_thread[i]++;
            // do something ...
        }

    /* On s'assure que le travail a bien été divisé*/
    printf("Nombre d'opérations par thread :\n");
    for (int i=0;i<nbThreads;i++)
        printf("Thread n°%d : %d\n", i, nb_op_thread[i]);
}

```

```

polytech@polytech-HPpolytech-840-G4:~/Bureau$
gcc -fopenmp openmp_test.c -o test
polytech@polytech-HPpolytech-840-G4:~/Bureau$
./test
Nb threads : 4
Nombre d'opérations par thread :
Thread n°0 : 2500
Thread n°1 : 2500
Thread n°2 : 2500
Thread n°3 : 2500

```

Figure 8 – OpenMP - Code 2 de validation, compilation et exécution

On remarque que les 10 000 opérations sont bien équitablement répartis entre les 4 threads disponibles.

13 Cuda

13.1 Vérifications

Avant d'installer les paquets nécessaires au fonctionnement de Cuda, il y a plusieurs vérifications à effectuer dans le but de savoir si le système est correctement paramétré pour exécuter des programmes mettant en oeuvre la programmation parallèle sur GPU.

Il a fallu dans un premier temps vérifier que la distribution Linux était compatible. Vu que nous avons choisi le SE en fonction de ce critère de compatibilité, il n'y avait pas de soucis de ce côté. Ensuite, il fallait vérifier que l'ordinateur possède une carte graphique de marque NVidia. En revanche, dans notre cas, nous paramétrons une distribution embarquée sur clé USB, cette vérification sera donc à effectuer sur les postes qui utiliseront notre clé USB bootable. Enfin, Cuda s'appuie sur le paquet gcc que nous avons installé puis vérifié précédemment.

13.2 Installation Cuda

Nous avons ensuite pu installer les paquets nécessaires à Cuda en suivant les recommandations du site <https://developer.nvidia.com/>. Cette installation requiert environ 3 Go d'espace disque et prend approximativement 3h. Après avoir installé Cuda, on va se rendre compte qu'il n'est pas encore utilisable.

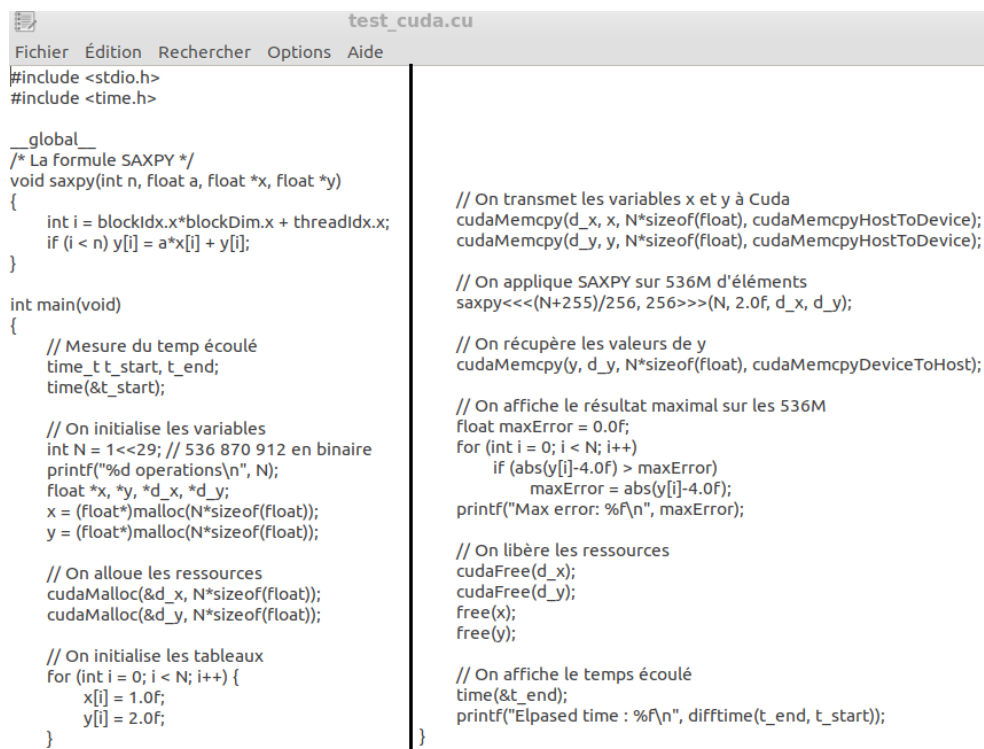
13.3 Installation nvcc

En effet, afin de réaliser des programmes parallèles en Cuda, on a besoin du compilateur nvcc. Pour cela, on va installer le paquet *nvidia-cuda-toolkit* qui intégrera le compilateur et les autres outils nécessaires (bibliothèques, python). Ce paquet requiert environ 2 Go d'espace disque et nécessite environ 2h d'installation.

13.4 Validation

Afin de valider l'installation de Cuda ainsi que de tous les composants nécessaires pour effectuer de la programmation parallèle s'appuyant sur le GPU, nous avons implémenté du code en *.cu*, l'extension propre aux programmes en Cuda. Ce programme implémente le SAXPY, pour Single Precision of $A \times X + Y$, une fonction d'algèbre linéaire qui consiste à calculer l'opération $z = ax + y$ où x, y et z sont des vecteurs et a un scalaire. Ce programme est considéré comme le *Hello world* de Cuda.

La figure ci-dessous illustre le code mis en œuvre dans le but de tester la programmation multi-GPU avec Cuda.



```
test_cuda.cu
Fichier  Édition  Recherche  Options  Aide
#include <stdio.h>
#include <time.h>

__global__
/* La formule SAXPY */
void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

int main(void)
{
    // Mesure du temps écoulé
    time_t t_start, t_end;
    time(&t_start);

    // On initialise les variables
    int N = 1<<29; // 536 870 912 en binaire
    printf("%d operations\n", N);
    float *x, *y, *d_x, *d_y;
    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));

    // On alloue les ressources
    cudaMalloc(&d_x, N*sizeof(float));
    cudaMalloc(&d_y, N*sizeof(float));

    // On initialise les tableaux
    for (int i = 0; i < N; i++) {
        x[i] = 1.0f;
        y[i] = 2.0f;
    }

    // On transmet les variables x et y à Cuda
    cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

    // On applique SAXPY sur 536M d'éléments
    saxpy<<<(N+255)/256>>>(N, 2.0f, d_x, d_y);

    // On récupère les valeurs de y
    cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

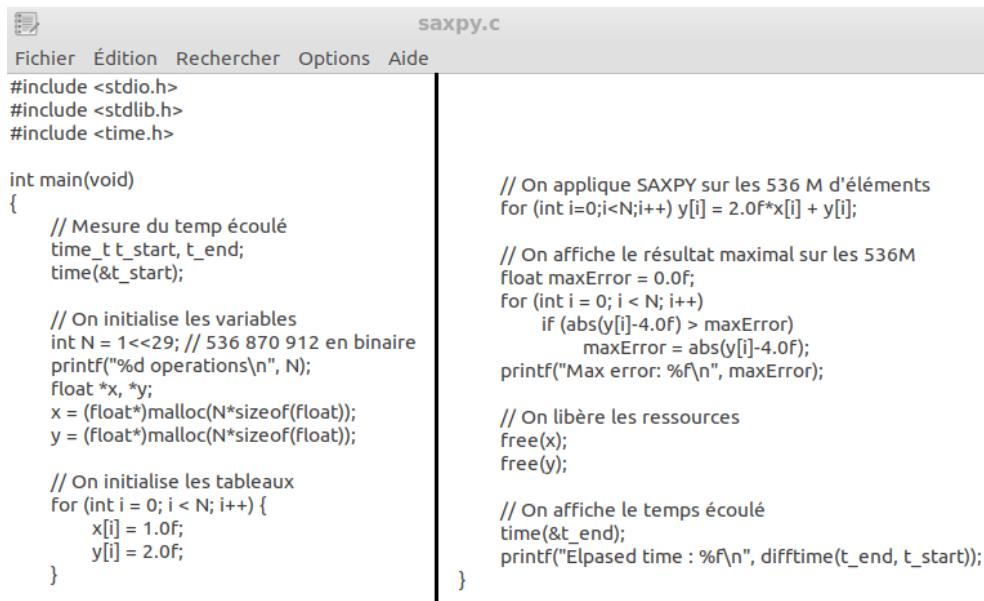
    // On affiche le résultat maximal sur les 536M
    float maxError = 0.0f;
    for (int i = 0; i < N; i++)
        if (abs(y[i]-4.0f) > maxError)
            maxError = abs(y[i]-4.0f);
    printf("Max error: %f\n", maxError);

    // On libère les ressources
    cudaFree(d_x);
    cudaFree(d_y);
    free(x);
    free(y);

    // On affiche le temps écoulé
    time(&t_end);
    printf("Elapsed time : %f\n", difftime(t_end, t_start));
}
```

Figure 9 – Cuda - Code SAXPY

Nous avons implémenté le même code en c classique afin de comparer les résultats. Il est illustré dans la figure suivante :



```
saxpy.c
Fichier Édition Rechercher Options Aide

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    // Mesure du temps écoulé
    time_t t_start, t_end;
    time(&t_start);

    // On initialise les variables
    int N = 1<<29; // 536 870 912 en binaire
    printf("%d operations\n", N);
    float *x, *y;
    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));

    // On initialise les tableaux
    for (int i = 0; i < N; i++) {
        x[i] = 1.0f;
        y[i] = 2.0f;
    }

    // On applique SAXPY sur les 536 M d'éléments
    for (int i=0;i<N;i++) y[i] = 2.0f*x[i] + y[i];

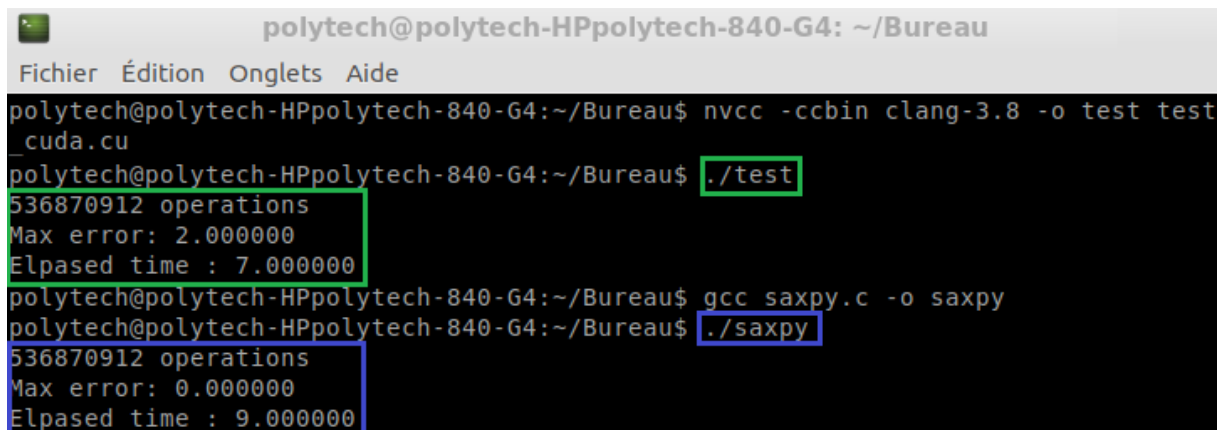
    // On affiche le résultat maximal sur les 536M
    float maxError = 0.0f;
    for (int i = 0; i < N; i++)
        if (abs(y[i]-4.0f) > maxError)
            maxError = abs(y[i]-4.0f);
    printf("Max error: %f\n", maxError);

    // On libère les ressources
    free(x);
    free(y);

    // On affiche le temps écoulé
    time(&t_end);
    printf("Elapsed time : %f\n", difftime(t_end, t_start));
}
```

Figure 10 – C - Code SAXPY

On observe les résultats suivants :



```
polytech@polytech-HPpolytech-840-G4: ~/Bureau
Fichier Édition Onglets Aide

polytech@polytech-HPpolytech-840-G4:~/Bureau$ nvcc -ccbin clang-3.8 -o test test_cuda.cu
polytech@polytech-HPpolytech-840-G4:~/Bureau$ ./test
536870912 operations
Max error: 2.000000
Elapsed time : 7.000000

polytech@polytech-HPpolytech-840-G4:~/Bureau$ gcc saxpy.c -o saxpy
polytech@polytech-HPpolytech-840-G4:~/Bureau$ ./saxpy
536870912 operations
Max error: 0.000000
Elapsed time : 9.000000
```

Figure 11 – Cuda vs C - Compilation et exécution codes SAXPY

On peut remarquer que le code Cuda (en vert) mets 2 secondes de moins que le code classique en c (en bleu). La programmation utilisant le GPU fait donc gagner 2 précieuses secondes.

14 MPI

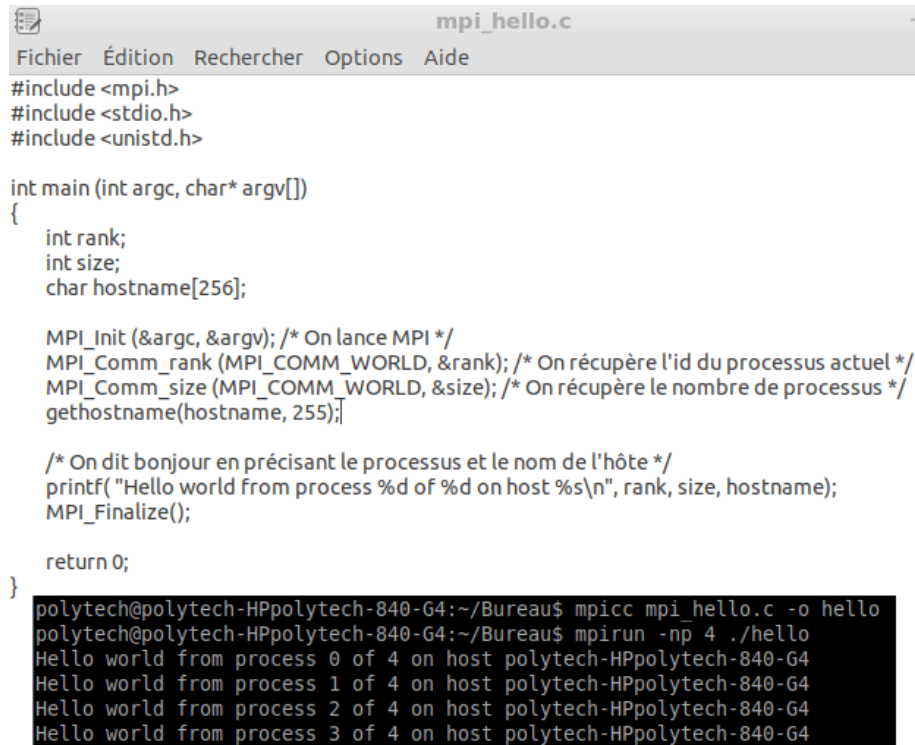
14.1 Installation

L'installation de MPI est la plus simple et la plus rapide. Il suffit d'installer le paquet *mpich*, sa documentation et les librairies dont il a besoin. Les paquets occupent 50 Mo d'espace disque et ne mets pas plus de 15 min à s'installer.

14.2 Validation mono-poste

Après avoir installé MPI, nous avons implémenté un programme simple qui récupère tous les processus à disposition et qui dit bonjour de chacun d'eux.

La figure ci-dessous illustre le code mis en oeuvre dans le but de valider la programmation MPI sur un seul poste.



```
mpi_hello.c
Fichier  Édition  Rechercher  Options  Aide
#include <mpi.h>
#include <stdio.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
    int rank;
    int size;
    char hostname[256];

    MPI_Init (&argc, &argv); /* On lance MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* On récupère l'id du processus actuel */
    MPI_Comm_size (MPI_COMM_WORLD, &size); /* On récupère le nombre de processus */
    gethostname(hostname, 255);

    /* On dit bonjour en précisant le processus et le nom de l'hôte */
    printf( "Hello world from process %d of %d on host %s\n", rank, size, hostname);
    MPI_Finalize();

    return 0;
}

polytech@polytech-HPpolytech-840-G4:~/Bureau$ mpicc mpi_hello.c -o hello
polytech@polytech-HPpolytech-840-G4:~/Bureau$ mpirun -np 4 ./hello
Hello world from process 0 of 4 on host polytech-HPpolytech-840-G4
Hello world from process 1 of 4 on host polytech-HPpolytech-840-G4
Hello world from process 2 of 4 on host polytech-HPpolytech-840-G4
Hello world from process 3 of 4 on host polytech-HPpolytech-840-G4
```

Figure 12 – MPI - Code de validation mono-poste, compilation et exécution

14.3 Validation sur réseau

TO DO

Cinquième partie

Gestion de projet

15 Organisation

Les séances dédiés à la réalisation des projets ASR étaient répartis à hauteur de 2 par semaine. Il était très difficile de se diviser les tâches car notre projet consistait en une suite d'installations suivies de validations et de sauvegardes. Il était impossible d'assurer deux installations en même temps car celles-ci devaient s'effectuer l'une à la suite des autres. Pendant qu'une personne effectuait les installations, la seconde personne était souvent sur un travail de recherches d'informations afin de préparer au mieux la prochaine installation.

16 Planning

Lors de l'analyse du projet, nous avons identifié les grandes tâches à effectuer. Celles-ci sont illustrées dans la figure suivante :



Figure 13 – Tâches identifiées au début du projet

La réalisation de projet s’est déroulée sur 4 mois. Les différentes tâches et leurs répartitions temporelles sont illustrées dans le diagramme de Gantt suivant :

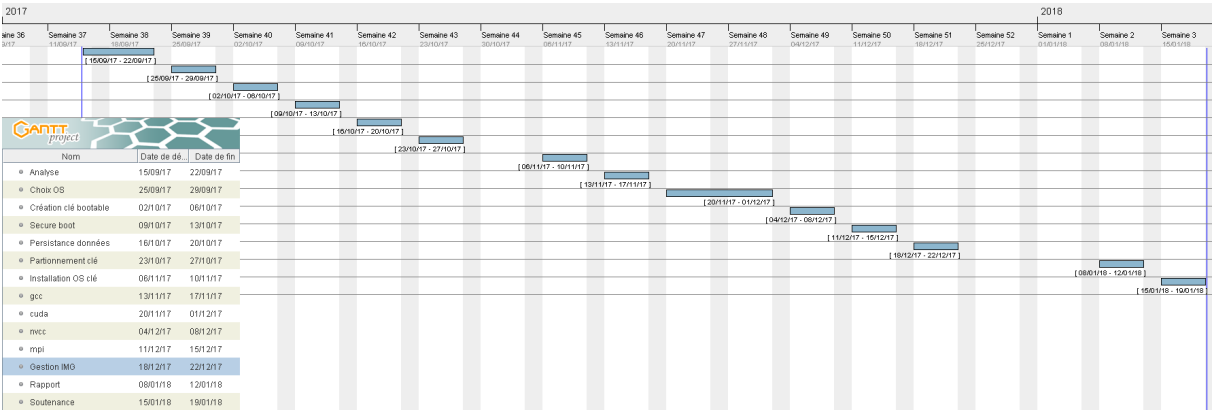


Figure 14 – Planning du projet

17 Gestion de versions

Dans le but de gérer les différentes versions de l’avancement de nos installations sur la clé USB, nous avons généré plusieurs fichiers *.img* dans le but de ne pas perdre notre avancement et de revenir en version antérieure en cas e problème majeur. Les différentes versions sont illustrées dans la figure suivante :

 os.img	Fichier IMG	15 138 816 Ko
 gcc.img	Fichier IMG	15 138 816 Ko
 cuda_nvcc.img	Fichier IMG	15 138 816 Ko
 mpi.img	Fichier IMG	15 138 816 Ko

Figure 15 – Versions des images

Sixième partie

Problèmes et difficultés rencontrés

18 Connexion réseau de Polytech

L'une des deux difficultés majeures rencontrées lors de ce projet concerne la connexion wifi délicate de Polytech. Selon l'heure de la journée, sur nos machines personnelles respectives il était possible que la vitesse de téléchargement descende jusqu'à 30 Ko/s. Cette situation n'était donc pas simple à gérer lorsque nous avions par exemple un outil de plus d'un Go à récupérer. En conséquence, plus nous avons avancé vers la fin du projet et plus les différents téléchargements ont été effectués soit en parallèle des créneaux de projet ASR mais chez nous soit pendant notre temps libre.

19 Temps des installations

La deuxième difficulté qui nous aura contraint à revoir notre façon de travailler est le temps d'installation des différents outils. En effet, si on considère le débit Internet difficile de Polytech en wifi sur nos machines personnels le temps d'installation des trois outils a pu s'élever jusqu'à 6h au total. C'est l'installation de CUDA qui est la plus difficile au vu de la taille assez conséquente de l'outil. Evidemment, ce genre de difficulté a posé de sérieux soucis pendant les créneaux de seulement 2h dans l'emploi du temps. Il a donc fallu gérer les installations en dehors de Polytech et des différents créneaux quand la situation l'exigeait.

20 Secure Boot

Un autre problème est également survenu assez rapidement dans notre projet. Cela concerne la façon de booter sur la clé USB au démarrage. En effet, selon la marque et le modèle de l'ordinateur il nous était difficile de booter sur notre clé. La faute à cause de l'outil de « Secure Boot » par exemple sur les modèles d'ASUS qui bloque par défaut les démarrages de système d'exploitations par des périphériques extérieurs à l'ordinateur.

De plus, il faut veiller à démarrer sur la clé USB en utilisant le mode *Legacy* plutôt que le mode *UEFI* (Unified Extensible Firmware Interface ou interface micrologiciel extensible unifiée en français). Ce dernier va bloquer un certain nombre de fonctionnalités et le boot risque de pas aboutir alors que le mode Legacy n'a pas le même paramétrage et permettra de booter sur la clé USB sans soucis.

Il est important de noter que ce problème ne concerne pas les PC présents au sein de l'école. Le démarrage par clé USB se fait sans difficulté à partir du menu de sélection de démarrage.

21 Mode Live et mode persistant

Il existe trois façons de booter sur une clé USB :

- En mode live : on utilise le système d'exploitation mais aucune donnée ne sera sauvegardée. Le système sera remis à zéro au prochain démarrage de celui-ci. Ce mode est présent pour permettre l'installation complète du système sur la clé ou à un autre endroit.
- En mode persistant : on utilise le système d'exploitation avec une capacité de stockage alloué par défaut à 4Go maximum.
- OS installé : l'OS est définitivement installé sur la clé USB avec une capacité définie par l'utilisateur lors de l'utilisation.

Une idée au départ était d'utiliser le « Mode Persistant » pour installer nos outils sans avoir un installé de manière complète le système d'exploitation sur notre clé. Malheureusement, cela n'est pas possible car la taille de nos outils au total dépasse largement 4Go. Nous avons donc dû prendre en compte cela et nous avons décidé d'installer de manière complète le système sur la clé.

22 Limite de stockage

Comme dit précédemment, la taille de nos outils au total est assez élevée, presque 16Go au final. En effet, presque l'intégralité des 16Go disponible par la clé USB permet l'installation des différents outils, ce qui laisse très peu de place pour le développement en lui-même.

23 Fragilité

Il faut absolument manier la clé USB avec le plus grand soin. Comme tout périphérique extérieur contenant un système d'exploitation celui-ci est fragile. Il est important de ne pas être brutal avec celui-ci ni de retirer la clé en cours d'exécution du système ou si celui-ci n'est pas encore totalement terminé. Dans le cas contraire, on peut s'exposer à une destruction ou une corruption pur et simple du système d'exploitation. Dans cet optique, il est important d'utiliser des sauvegardes de notre OS pour éviter une réinstallation complète de celui-ci.

Septième partie

Outils utilisés

Dans cette partie, nous récapitulerons tous les outils utilisés, nous détaillerons leurs fonctionnalités et nous indiquerons où et comment se les procurer.

24 Linux Live Creator

LinuxLive USB Creator est un logiciel gratuit et open source pour Windows qui permet de créer une clé USB bootable avec une distribution Linux dessus. Ce logiciel se veut très simple d'utilisation avec une interface ergonomique et intuitive. Chaque étape nécessaire à la création de la clé USB est affublé d'un feu de circulation permettant d'indiquer à l'utilisateur si l'étape a bien été remplie.

Cet outil est disponible sur le site officiel du logiciel : <http://www.linuxliveusb.com/fr/>



Figure 16 – Linux live USB creator

25 USB Image Tool

USB image tool est un logiciel gratuit pour Windows qui permet créer facilement des copies d'images disques à partir de lecteurs amovibles. L'avantage de ce logiciel est qu'il n'a pas besoin d'installation, il est prêt à l'emploi après téléchargement. Il permet de générer des images au format *.img* et *.ima*. Il permet également la restauration d'images disques en cas de perte de données.

Cet outil est disponible sur le site officiel du logiciel : <https://usb-image-tool.fr.uptodown.com/windows>

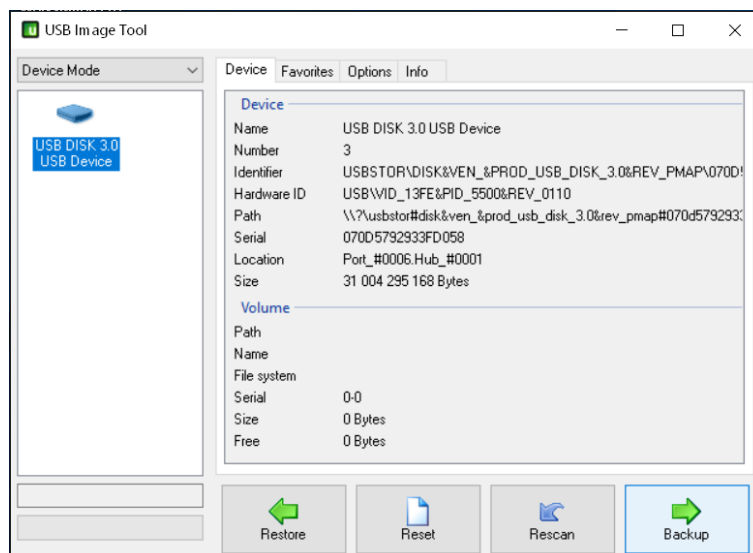


Figure 17 – USB image tool

26 Etcher

Etcher est un logiciel gratuit pour Windows qui permet de graver des images disques sur des lecteurs amovibles comme les clés USB. Il est très intuitif et facile d'utilisation. Il comporte trois étapes qui doivent être réalisées les unes à la suite des autres. Il faut d'abord sélectionner l'image disque, au format *.img*, *.iso* ou *.zip*, avant de sélectionner le lecteur de destination qui peut être une clé USB ou une carte SD. Il suffit ensuite d'appuyer sur *Flash!* et d'attendre que la gravure se fasse.

Cet outil est disponible sur le site officiel du logiciel : <https://etcher.io/>

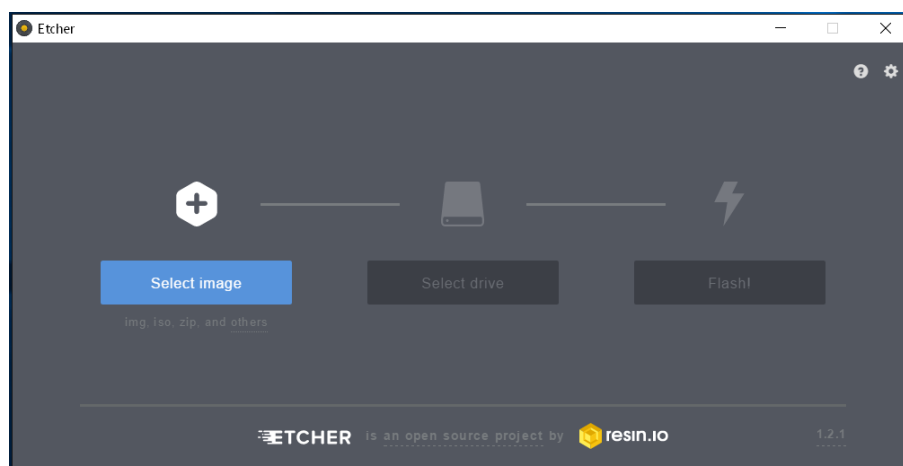
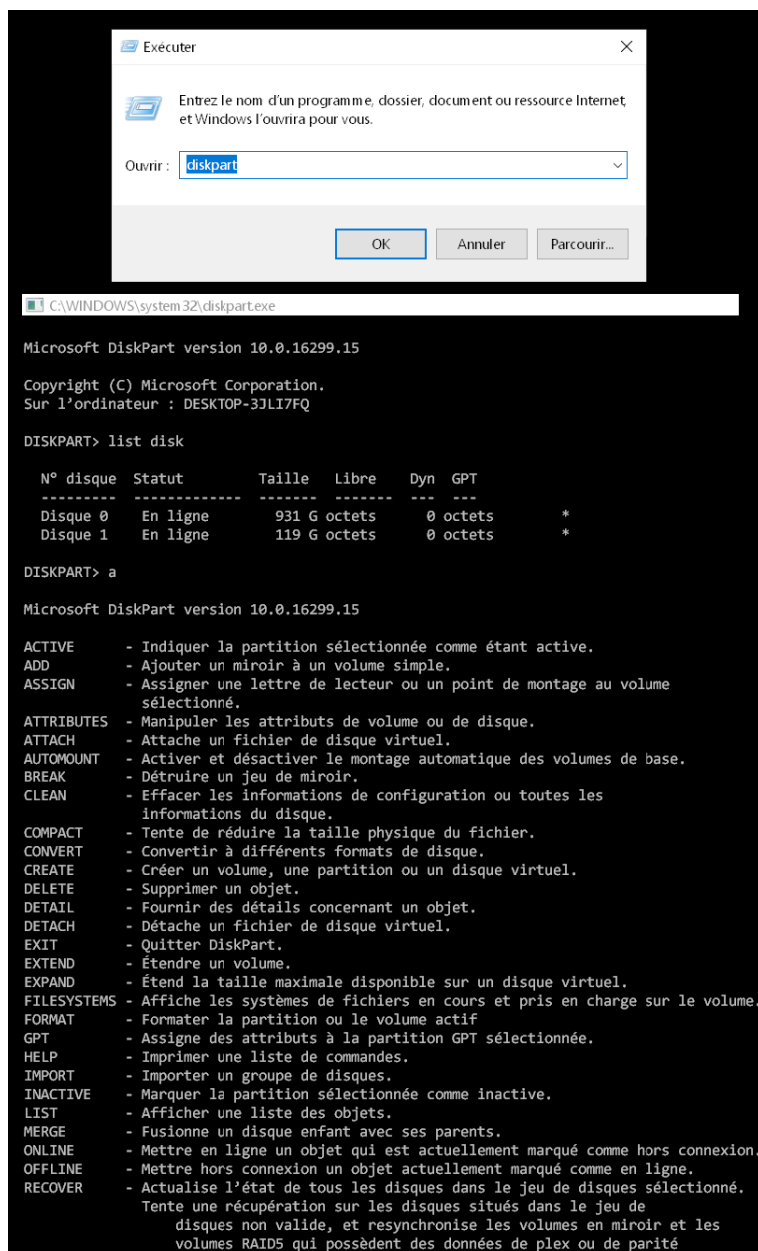


Figure 18 – Etcher

27 Diskpart

Diskpart est un outil logiciel de partitionnement des supports de stockage. Il est nativement intégré sous Windows. Diskpart permet de découper un disque dur ou autre périphérique de stockage externe en plusieurs partitions, par exemple dans le but d'assurer la cohabitation de plusieurs systèmes d'exploitation sur une même machine. Diskpart permet également de réparer les supports de stockage en leur attribuant une seule partition primaire.



```
Microsoft DiskPart version 10.0.16299.15

Copyright (C) Microsoft Corporation.
Sur l'ordinateur : DESKTOP-3JLI7FQ

DISKPART> list disk

   N° disque   Statut   Taille   Libre   Dyn   GPT
   -----   -
Disque 0      En ligne 931 G octets  0 octets  *
Disque 1      En ligne 119 G octets  0 octets  *
```

```
Microsoft DiskPart version 10.0.16299.15

ACTIVE          - Indiquer la partition sélectionnée comme étant active.
ADD             - Ajouter un miroir à un volume simple.
ASSIGN          - Assigner une lettre de lecteur ou un point de montage au volume
                  sélectionné.
ATTRIBUTES      - Manipuler les attributs de volume ou de disque.
ATTACH          - Attache un fichier de disque virtuel.
AUTOMOUNT       - Activer et désactiver le montage automatique des volumes de base.
BREAK          - Détruire un jeu de miroir.
CLEAN           - Effacer les informations de configuration ou toutes les
                  informations du disque.
COMPACT         - Tente de réduire la taille physique du fichier.
CONVERT         - Convertir à différents formats de disque.
CREATE          - Créer un volume, une partition ou un disque virtuel.
DELETE         - Supprimer un objet.
DETAIL         - Fournir des détails concernant un objet.
DETACH         - Détache un fichier de disque virtuel.
EXIT           - Quitter DiskPart.
EXTEND         - Étendre un volume.
EXPAND         - Étend la taille maximale disponible sur un disque virtuel.
FILESYSTEMS     - Affiche les systèmes de fichiers en cours et pris en charge sur le volume.
FORMAT         - Formater la partition ou le volume actif
GPT            - Assigne des attributs à la partition GPT sélectionnée.
HELP           - Imprimer une liste de commandes.
IMPORT         - Importer un groupe de disques.
INACTIVE       - Marquer la partition sélectionnée comme inactive.
LIST           - Afficher une liste des objets.
MERGE          - Fusionne un disque enfant avec ses parents.
ONLINE         - Mettre en ligne un objet qui est actuellement marqué comme hors connexion.
OFFLINE        - Mettre hors connexion un objet actuellement marqué comme en ligne.
RECOVER        - Actualise l'état de tous les disques dans le jeu de disques sélectionné.
                  Tente une récupération sur les disques situés dans le jeu de
                  disques non valide, et resynchronise les volumes en miroir et les
                  volumes RAID5 qui possèdent des données de plex ou de parité
```

Figure 19 – Diskpart - Exécution et liste des commandes

Huitième partie

Guide d'utilisation

Dans le but d'aider les utilisateurs à utiliser notre clé USB bootable intégrant un SE capable d'effectuer de la parallélisation avec OpenMP, Cuda et MPI, nous avons réalisé un guide d'utilisation.

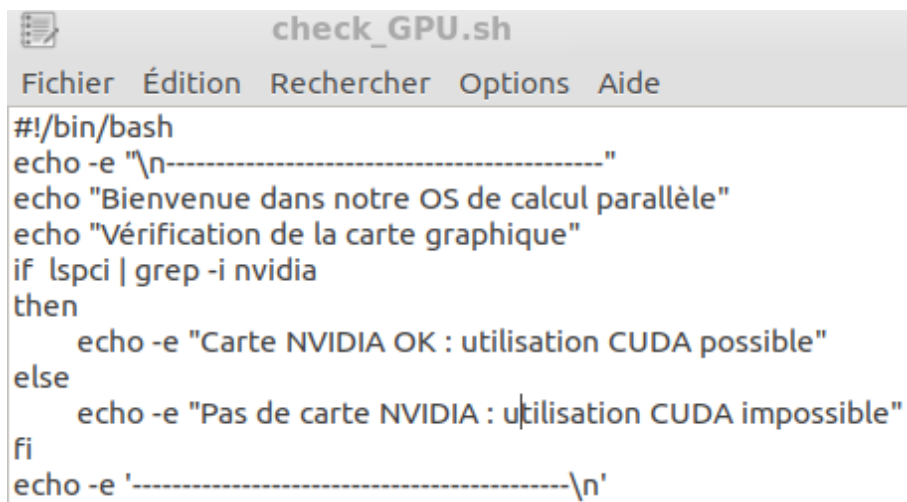
Brancher la clé USB.

Ouverture de session "password" pour le compte polytech

Dire où sont les exemples

Commande OpenMP

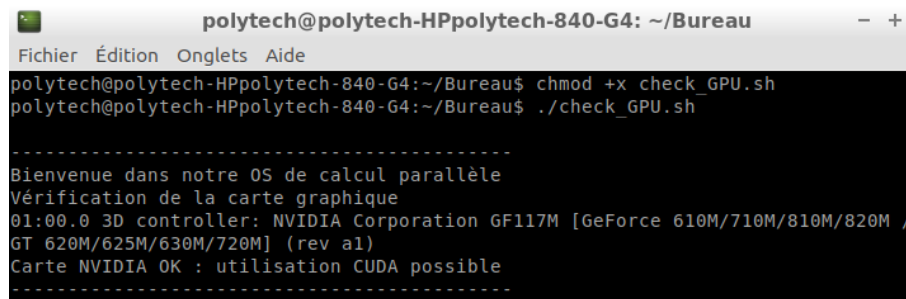
Nous avons réalisé un petit script `.sh` afin de vérifier si une carte graphique de marque NVIDIA est présente afin de programmer parallèlement en utilisant le GPU. Le code est illustré dans la figure suivante :

A screenshot of a terminal window titled 'check_GPU.sh'. The window has a menu bar with 'Fichier', 'Édition', 'Rechercher', 'Options', and 'Aide'. The code displayed is a bash script that checks for an NVIDIA GPU. It starts with a shebang, prints a separator line, a welcome message, and a verification message. It then uses 'lspci' to check for 'nvidia' in the hardware list. If found, it prints 'Carte NVIDIA OK : utilisation CUDA possible'; otherwise, it prints 'Pas de carte NVIDIA : utilisation CUDA impossible'. Finally, it prints another separator line.

```
#!/bin/bash
echo -e "\n-----"
echo "Bienvenue dans notre OS de calcul parallèle"
echo "Vérification de la carte graphique"
if lspci | grep -i nvidia
then
    echo -e "Carte NVIDIA OK : utilisation CUDA possible"
else
    echo -e "Pas de carte NVIDIA : utilisation CUDA impossible"
fi
echo -e '-----\n'
```

Figure 20 – Vérification de la carte graphique - Code

Nous avons dans la figure suivante un exemple de machine possédant une carte graphique NVIDIA. La programmation en Cuda sera donc possible.

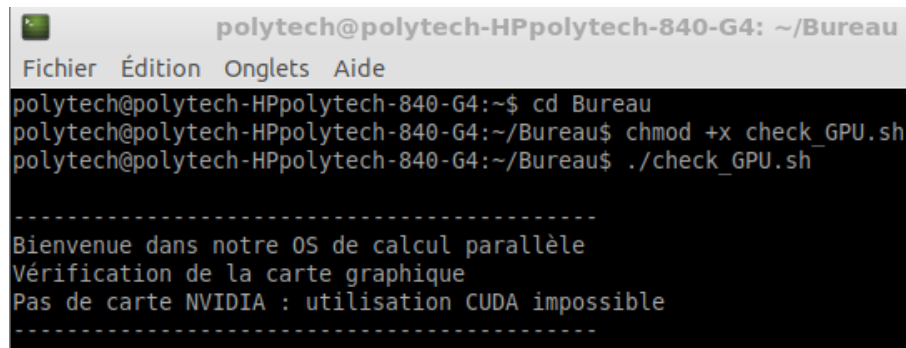
A terminal window titled 'polytech@polytech-HPpolytech-840-G4: ~/Bureau'. The user runs 'chmod +x check_GPU.sh' and then './check_GPU.sh'. The script outputs a welcome message, verifies the GPU, and reports 'Carte NVIDIA OK : utilisation CUDA possible'.

```
polytech@polytech-HPpolytech-840-G4: ~/Bureau
Fichier  Édition  Onglets  Aide
polytech@polytech-HPpolytech-840-G4:~/Bureau$ chmod +x check_GPU.sh
polytech@polytech-HPpolytech-840-G4:~/Bureau$ ./check_GPU.sh

-----
Bienvenue dans notre OS de calcul parallèle
Vérification de la carte graphique
01:00.0 3D controller: NVIDIA Corporation GF117M [GeForce 610M/710M/810M/820M /
GT 620M/625M/630M/720M] (rev a1)
Carte NVIDIA OK : utilisation CUDA possible
-----
```

Figure 21 – Vérification de la carte graphique - OK

Nous avons dans la figure suivante un exemple de machine ne possédant pas de carte graphique NVIDIA. La programmation en Cuda ne sera donc pas possible.

A terminal window titled 'polytech@polytech-HPpolytech-840-G4: ~/Bureau'. The user runs 'cd Bureau', 'chmod +x check_GPU.sh', and './check_GPU.sh'. The script outputs a welcome message, verifies the GPU, and reports 'Pas de carte NVIDIA : utilisation CUDA impossible'.

```
polytech@polytech-HPpolytech-840-G4:~/Bureau$ cd Bureau
polytech@polytech-HPpolytech-840-G4:~/Bureau$ chmod +x check_GPU.sh
polytech@polytech-HPpolytech-840-G4:~/Bureau$ ./check_GPU.sh

-----
Bienvenue dans notre OS de calcul parallèle
Vérification de la carte graphique
Pas de carte NVIDIA : utilisation CUDA impossible
-----
```

Figure 22 – Vérification de la carte graphique - KO

Commande Cuda

Commande MPI

Neuvième partie

Guide d'installation

Dans le but d'aider un utilisateur à installer l'image de notre SE capable d'effectuer de la parallélisation avec OpenMP, Cuda et MPI, nous avons réalisé un guide d'installation. Ce guide permettra aussi à l'utilisateur de générer sa propre image après avoir fait ce qu'il voulait sur le SE. On lui indiquera également comment re-partitionner sa clé USB et graver à nouveau une image disque sur sa clé.

1) Sauvegarde .IMG avec USB Image Tool

Pour lancer le programme USB image tool, il suffit de double-cliquer sur l'exécutable dans le dossier qui a été téléchargé. Ensuite, il faut identifier la clé USB de laquelle on souhaite effectuer une sauvegarde la sélectionner. Ensuite, il faut cliquer sur le bouton *Backup* et spécifier le nom de fichier de l'image disque qui va être générée. L'opération ne devrait pas prendre plus de 15 minutes. La figure ci-après résume les étapes à effectuer :

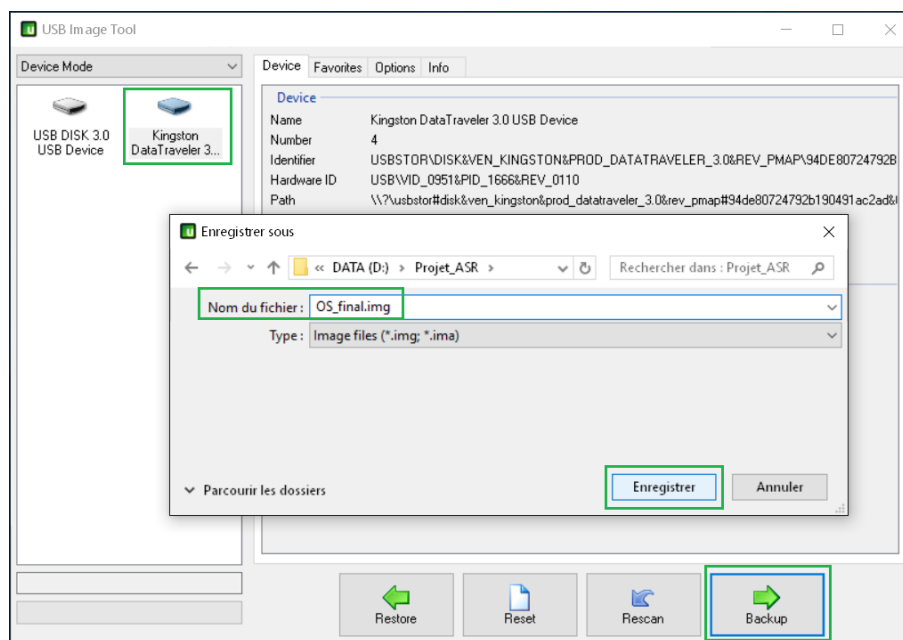


Figure 23 – Sauvegarde de l'image disque avec USB Image Tools

2) Partitionnement avec Diskpart

Pour lancer le programme Diskpart sous Windows, il suffit d'aller dans *Exécuter* et de saisir *diskpart* dans la saisie du programme à ouvrir.

Ensuite, il faudra sélectionner le disque à partitionner, effacer les partitions actuelles et créer une nouvelle partition primaire. Ces actions à effectuer sont synthétisées dans la figure suivante :

```
C:\WINDOWS\system32\diskpart.exe
Copyright (C) Microsoft Corporation.
Sur l'ordinateur : DESKTOP-3JLI7FQ

DISKPART> list disk

   N° disque  Statut      Taille  Libre  Dyn  GPT
   -----
   Disque 0    En ligne    931 G  0 octets  0 octets  *
   Disque 1    En ligne    119 G  0 octets  0 octets  *
   Disque 2    En ligne    28 G  0 octets  0 octets

DISKPART> select disk=2

Le disque 2 est maintenant le disque sélectionné.

DISKPART> detail disk

USB DISK 3.0 USB Device
ID du disque : 51FD4E08
Type : USB
État : En ligne
Chemin : 0
Cible : 0
ID LUN : 0
Chemin d'accès de l'emplacement : UNAVAILABLE
État en lecture seule actuel : Non
Lecture seule : Non
Disque de démarrage : Non
Disque de fichiers d'échange : Non
Disque de fichiers de mise en veille prolongée : Non
Disque de fichiers de vidage sur incident : Non
Disque en cluster : Non

   N° volume  Ltr  Nom      Fs      Type      Taille  Statut  Info
   -----
   Volume 5   G    FAT32    Amovible  6032 M  Sain
   Volume 6   F    FAT32    Amovible  22 G   Sain

DISKPART> clean

DiskPart a réussi à nettoyer le disque.

DISKPART> convert mbr

DiskPart a correctement converti le disque sélectionné au format MBR.

DISKPART> create partition primary

DiskPart a réussi à créer la partition spécifiée.
```

Figure 24 – Re-partitionnement de la clé USB avec Diskpart

3) Gravure .IMG avec Etcher

Il suffit de lancer le programme Etcher, de sélectionner l'image disque à graver, de sélectionner la clé USB de destination puis de lancer la gravure en appuyant sur le bouton *Flash!*. La figure ci-après illustre les actions à effectuer :

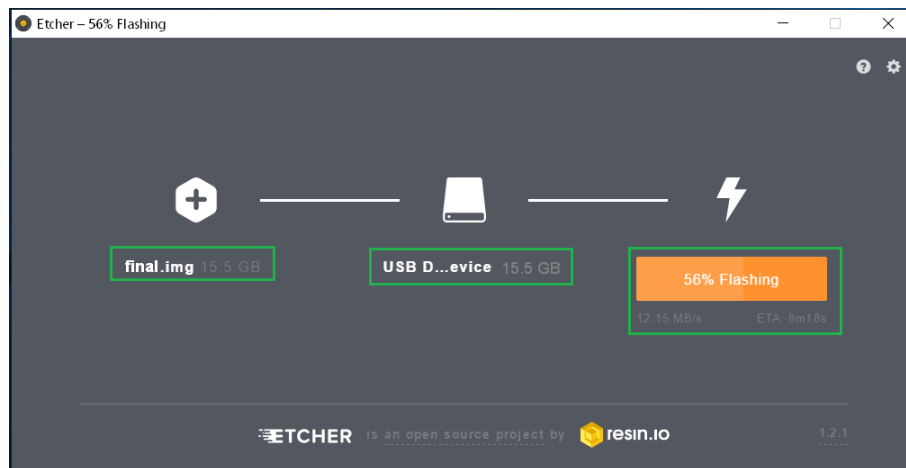


Figure 25 – Gravure de l'image disque avec Etcher

Il y aura une phase de gravure puis une phase de validation. Le tout ne devrait pas dépasser les 30 minutes.

Dixième partie

Guide du développeur

Dans le but d'aider un futur développeur qui se lancerait dans la même tâche que celle qui nous a été confiée, nous avons créé un guide destiné à une personne ayant des compétences assez avancées en système dans le but de lui donner la marche à suivre dans la réalisation du clé USB bootable intégrant un SE capable d'effectuer de la parallélisation avec OpenMP, Cuda et MPI.

Onzième partie

Conclusion

Ce projet aura été pour nous l'occasion de mettre en pratique tout un panel de connaissances acquises tout au long de notre formation en Architecture, Systèmes et Réseaux. Bien que le sujet en lui-même soit relativement court, il n'aura pas été exempt de toute difficulté comme nous avons pu le constater et l'expérimenter par nous-même. Néanmoins, c'est à travers celles-ci et en utilisant nos connaissances en Unix et en système que nous avons pu mener ce projet à bout. Cela nous a permis d'enrichir nos connaissances et nos acquis qui nous seront utiles aussi bien dans notre vie personnelle que dans notre vie professionnelle.

Nous rendons avec ce rapport les 2 clés USB qui nous avaient été confiées avec le SE configuré pour les 3 modes de parallélisation gravé à l'intérieur. Nous rendons également l'image disque au format *.img* de ce SE. Un guide d'installation et un guide d'utilisation, respectivement pour l'image disque et les 2 clés USB rendues, sont disponibles dans ce rapport.

Configuration d'un OS pour le calcul parallèle

Résumé

Projet ASR de 5ème année à l'école Polytechnique de Tours.

L'objectif est la mise en place d'un système d'exploitation contenant tous les outils nécessaires au développement suivant les trois grands axes de parallélisation : OpenMP, Nvidia GPU CUDA et MPI.

Cette réalisation passe par la création de la clé USB bootable puis par l'ajout des éléments permettant les trois axes de parallélisation et les validations qui viennent avec.

Mots-clés

Système d'exploitation, OpenMP, Cuda, MPI

Abstract

ASN project of the 5th year at the Polytechnic School of Tours.

The goal is to set up an operating system containing all the necessary tools for development along the three main axes of parallelization : OpenMP, Nvidia GPU CUDA and MPI.

This realisation need firstly to create the bootable USB key device, secondly in adding every element that each of these three parallelization axes needs, and finally in ensuring the validations that they need.

Keywords

Operating system, OpenMP, Cuda, MPI

Tuteur académique
Patrick MARTINEAU

Étudiants
Benjamin CALDAS (DI5)
Logan VERECQUE (DI5)