POLYTECH TOURS
64 avenue Jean Portalis
37200 TOURS, FRANCE
Tél +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Rapport de projet ASR 2022 - 2023

Projet Station TV (Capture JPEG)

Tuteur Académique :
Mathieu DELALANDRE

Etudiant :
PERVEYRIE Hugo
DURET Romuald

Table des matières

Introduction

Étudiants à Polytech Tours en Informatique option Architecture des Systèmes et Réseau en 3ème année de cycle ingénieur, dans le cadre du projet d'Architecture des Systèmes et Réseaux, nous avons amélioré la solution de Hugo PERVEYRIE (programme de capture sur la Station TV) afin que celle-ci fonctionne avec une architecture biprocesseur. L'objectif du projet était donc de réaliser un état de l'art des solutions possibles, puis de choisir et d'intégrer une de ses solutions dans le programme. En parallèle, l'installation de 2 disques dur sur la station de travail T640 a été réalisée.

Dans ce rapport on vous présentera d'abord le LIFAT, les différents projets en cours et le projet station TV. On verra ensuite l'état de l'art sur les différentes solutions existante et l'implémentation d'une de ses solutions dans le programme de capture. On abordera ensuite l'amélioration matérielle et logiciel de la station de travail puis les différentes difficultés rencontrées.

On finalisera ce rapport avec des conclusions et perspectives concernant ce projet ASR.

Présentation du cadre du stage

Présentation du LIFAT

Le LIFAT (Laboratoire d'Informatique Fondamentale & Appliquée de Tours) est composé de 48 membres de la faculté (Professeurs et Professeurs Assistant), 25 doctorants et 7 docteurs.

Les préoccupations scientifiques du LIFAT sont nombreuses. Le LIFAT souhaite concevoir et développé des modèles, des méthodes et des algorithmes ainsi que fournir des ressources et des logiciels pour extraire des informations et récupérer des connaissances à partir de données en intégrant l'interaction homme-machine. Le LIFAT souhaite aussi résoudre des problèmes d'optimisation combinatoire avec la volonté d'obtenir de bons résultats en un bon temps de calcul.

Le laboratoire est actuellement organisé en 3 groupes de recherches :

- Base de données et Traitement des langues naturelles (BdTln)
- Recherche opérationnelle, Ordonnancement et Transport (ROOT)
- Reconnaissance de Forme et Analyse d'Image (RFAI)

Depuis le 1er janvier 2012, l'équipe « Operations research, scheduling, and transportation" » est associé avec le CNRS en tant que « Equipe Recherche Labellisée du CNRS ».

Trois champs d'application principaux fédèrent les activités du laboratoire :

- Santé et Handicap
- Big Data et Calcul Haute Performance
- Humanités numériques

Le LIFAT a de l'expérience dans les collaborations académiques (au niveau national et international) et les partenariats industriels. Les nombreuses opportunités de transfert de technologie (au monde social et économique) de recherche menée par le laboratoire ont abouti à la mise en place d'un centre de transfert associé avec le laboratoire ILIAD.

Le LIFAT et le Laboratoire d'Informatique Fondamentale d'Orléans (LIFO) font partie de la fédération de recherche Informatique Centre Val de Loire (ICVL)

Plus que jamais, l'accent est mis sur la qualité de la production scientifique dans les journaux internationaux, sur le suivi des doctorants pour une qualité de travail maximum, et sur le maintien et l'enrichissement de leurs relations internationales.

Présentation du projet Station TV

Le projet Station TV est un projet mené par le LIFAT depuis 2019. Le but du projet est de récupérer des données provenant des chaînes télévisées françaises grâce à deux stations de travail dédié et de les utiliser pour faire des analyse TV, de la télévision sociale et du journalisme de données.

Le projet station TV fait l'objet de plusieurs projets étudiants ayant pour but le déploiement d'applications de traitement d'images et d'intelligence artificielle.

Les domaines d'application de ce projet étant extrêmement variés et évoluant au fil des années, il est compliqué de tous les décrire.

Amélioration du programme de capture et de la station de travail

Concernant la partie du projet Station TV sur laquelle nous avons été amené à travailler, nous devions améliorer un programme de capture existant préalablement réalisé par Hugo PERVEYRIE dans le cadre de son stage de fin de 4ème année. L'objectif était de permettre au programme d'utiliser toutes les ressources matérielles disponible à savoir l'utilisation des 2 processeurs en simultané.

Afin de choisir une solution adéquat, on a réalisé un état de l'art des différentes solutions existantes qui nous permettrait de résoudre notre problème. La solution qui s'est démarqué des autres est Open MP par sa simplicité et notre connaissance commune sur cet API. Open MP à lui seul ne nous permettait pas de résoudre ce problème. Nous avons manipulé les groupes de processeurs Windows pour que Visual Studio utilise bien les 2 processeurs comme souhaité.

Nous avons ensuite installé les deux disques dur préalablement commandés pour que la station de travail dispose d'un volume de stockage suffisant pour faire fonctionner le programme de capture sur le long terme.

Pour terminer nous devions effectuer des tests de charge du programme de capture après redéploiement des cartes d'acquisition mais cette partie n'a pas pu être réalisé par manque de temps.

Toute cette partie du projet s'est déroulé sur la station de travail Dell PowerEdge T640. Cette partie du projet étant assez technique et traitant beaucoup l'aspect matériel de la station de travail, il est important de décrire ses composants.

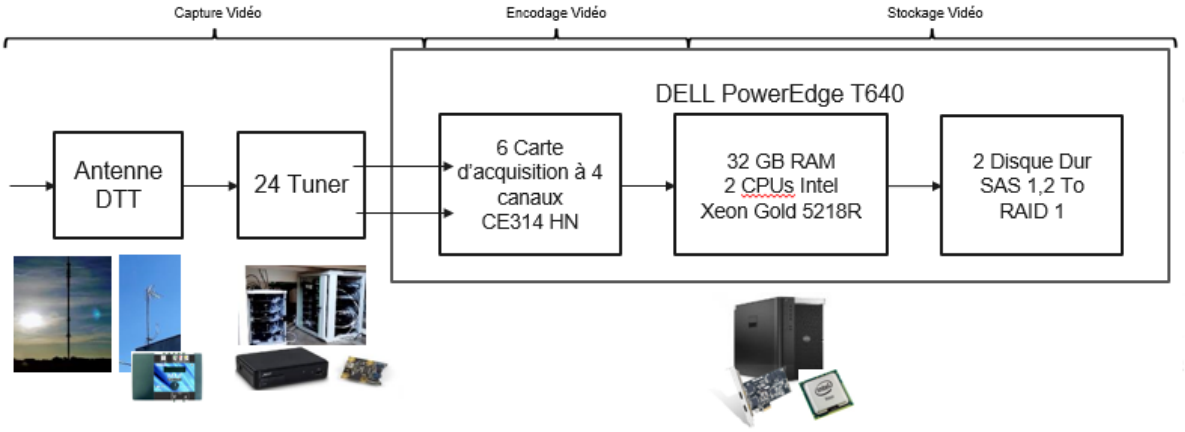
Les deux stations de travail

Les deux stations de travail utilisés par le LIFAT sont bien différentes. La Dell Précision T7600 est utilisée pour de la capture vidéo simple tandis que la Dell PowerEdge T640 est utilisée pour du traitement d'image, qui est beaucoup plus gourmand en ressource que de la capture vidéo.

Dell PowerEdge T640

La station de travail PowerEdge T640 est la station sur laquelle nous avons travaillé pendant toute la durée du projet. Les composants de cette station étant une grosse composante de notre projet, vous pourrez retrouver dans les sous chapitres suivants des détails concernant cette station.

Pour bien comprendre les prochaines sous parties voici un schéma présentant le cheminement de la capture d'image, allant de la capture vidéo au stockage des images :



Composants

La station de travail DELL PowerEdge T640 est composé de 2 processeurs Intel Xeon Gold 5218R (2 x 20 cœurs cadencé à 2,10 GHz) et de 32Gb de RAM. Elle possède 5 cartes d'acquisition AVerMedia CE314-HN et 1 carte d'acquisition AVerMedia plus récente. L'architecture disque étant assez complexe, elle sera décrite dans le chapitre « L'architecture disque ».

Architecture biprocesseur

Une architecture biprocesseur pour un ordinateur indique le fait que l'ordinateur est équipé de 2 processeurs. C'est une forme d'architecture parallèle puisque ces 2 processeurs vont travailler ensemble pour apporter plus de puissance de calcul aux programmes.

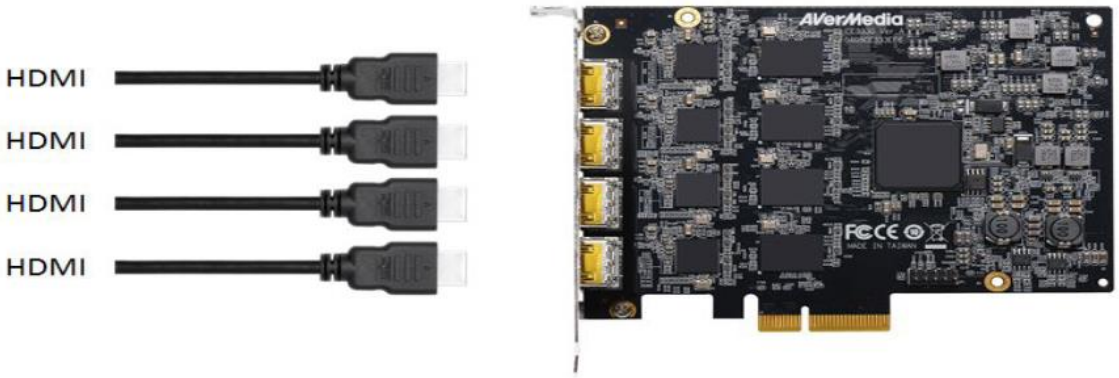
Il faut savoir que Windows Server gère nativement ce genre d'architecture. La station de travail est donc équipée de Windows Server 2019.

Le problème des architectures multiprocesseur est que tous les programmes ne sont pas compatibles et n'utiliseront donc qu'un seul processeur sur les deux. Il faut donc utiliser ou programmer des logiciels capables d'utiliser une telle architecture. C'est pour cela que ce type d'architecture n'est utilisé que sur des serveurs et non sur des machines personnelles car peu de programmes sont compatibles. Notre projet a donc pour but d'améliorer le programme de capture pour répondre à cette contrainte.

Il y'a énormément d'intérêt à utiliser une telle architecture sur un serveur mais globalement c'est la performance qui est recherché. Sur la station de travail, cette architecture va doubler notre nombre de cœur et donc notre nombre de thread indépendant disponible.

Carte de capture PCIe Full HD HDMI 4 canaux

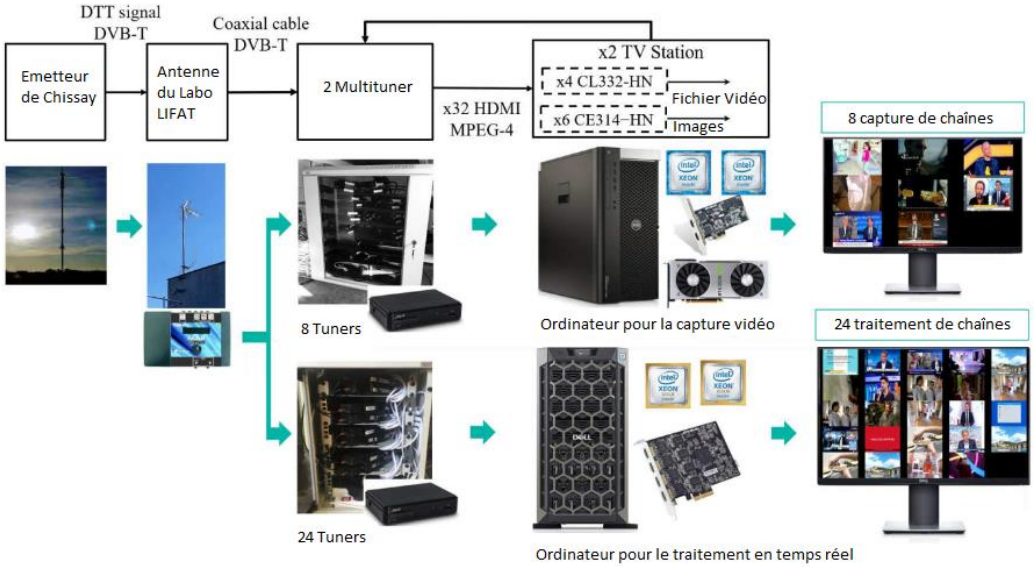
Le modèle de la carte de capture est une AVerMedia CE314-HN. C'est une carte de capture PCIe Gen1x4 (1 Go/s) équipée de 4 ports HDMI avec une entrée audio intégrée. Avec son entrée de source vidéo HDMI, il est capable de prendre en charge une capture de vidéo en temps réel allant jusqu'à la résolution Full HD en 60 images par seconde (FPS). Lorsque l'on utilise les 4 ports, on peut capturer en Full HD en 30 FPS. Les images capturées sont nativement en 8 bit par pixel et en format de pixels YUY2. Ce modèle supporte le multi carte et nous permet donc d'en embarquer 5 dans la station de travail afin de capturer 4x6 flux soit 24 flux.



La station de travail est aussi équipée d'une 6ème carte d'acquisition qui est un modèle plus récent que celle vu précédemment.

Différence entre les 2 stations de travail

Tout d'abord, voici un schéma afin de mieux comprendre le cheminement entre la réception du signal TNT et la réception puis traitement par les 2 stations TV :



La différence entre les deux stations TV est assez simple. La station de capture vidéo va récupérer des fichiers audio/vidéo des 8 chaînes qui seront traités plus tard par d'autre applications et la station de traitement en temps réel va récupérer les 24 chaînes et générer un nombre d'image choisis pour ces 24 chaînes. Cette génération d'image est beaucoup plus couteuse en ressource qu'une simple capture vidéo.

La connexion Tuner / Chaine / Carte

Le LIFAT capture le signal TNT et le démodule en différent flux vidéo à l'aide des 24 tuners. Nous avons donc 24 chaines capturable sur les 24 tuners. Pour cela on va utiliser les 6 cartes d'acquisition AVerMedia CE 314 HN pour récupérer les 24 flux vidéo des tuners. Une carte CE 314 HN peut capturer 4 flux en simultané.

L'architecture disque

La station de travail Dell PowerEdge T640 est composé initialement de 2 disque dur Dell 1,2 TB cadencé à 10 000 tours par minutes et de 3.5 pouces en SAS

Il existe actuellement 3 types de disque dur :

- Les disques SATA (Serial Advanced Technology Attachment) étant les plus commun car beaucoup utilisés sur les PC mais très rarement sur les serveurs en raison de leur faible performance en lecture/écriture.
- Les disques SAS (Serial Attached SCSI) qui sont la plupart du temps utilisé sur les serveurs et les stations de travail car ils sont 2 fois plus performant que les disques SATA, possède la même volumétrie et restent très largement abordable.
- Les disques SSD (Solid State Drive) qui sont des disques très rapide en lecture/écriture mais possédant une volumétrie moindre. Ils sont généralement utilisés pour héberger un système d'exploitation ou une base de données en raison de leur gros besoin en lecture/écriture.

C'est donc tout naturellement que le choix de Dell pour les disques intégrés de base à la station de travail a été porté sur les disques SAS.

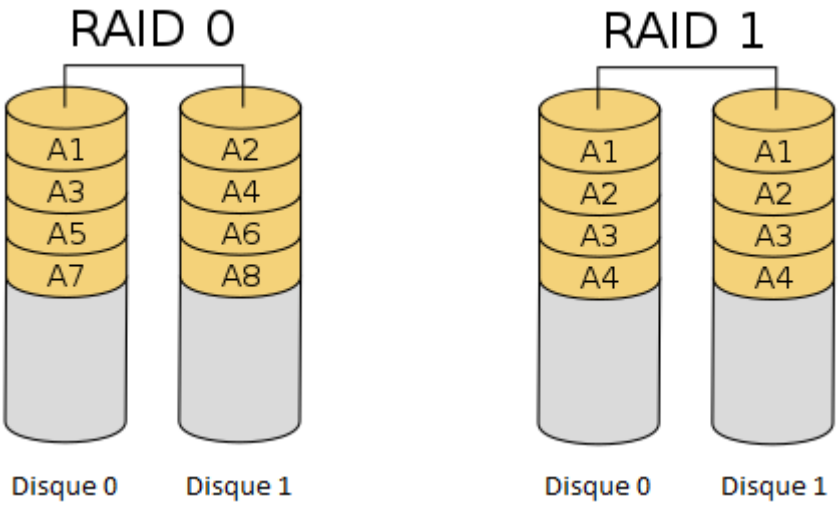
La station de travail T640 est équipée d'un contrôleur de réseau de disque Dell PERC H330. Un contrôleur de réseau de disques est un périphérique qui gère les disques physiques et les présente à l'ordinateur sous formes d'unité logique. Il implémente presque toujours RAID ce qui le rend plus généralement connu sous le nom de contrôleur RAID.

Le RAID est un ensemble de techniques de virtualisation du stockage qui permet de répartir les données sur plusieurs disque durs améliorant ainsi les performances, la sécurité ou la tolérance aux pannes. Il existe différents types d'architecture RAID numéroté à partir de 0.

Les deux interfaces qui nous intéressent sont les suivantes :

- RAID 0 (volume agrégé par bandes) est une configuration RAID permettant de faire travailler n disques dur (n>=2) en parallèle afin d'augmenter significativement les performances.
- RAID 1 (disques en miroir) est une configuration RAID utilisant n disques redondants (avec n>=2), chaque disque de la grappe contenant exactement les mêmes données à tout moment. Cette solution offre un excellent niveau de protection des données. La capacité utile reste inchangée et est égale à la capacité du plus petit disque dur de la grappe.

Les architectures RAID peuvent se combiner entre elles (RAID 0+1)



Sur notre station de travail, les 2 disques dur SAS sont en RAID 1 afin de permettre au serveur de continuer à s'exécuter en cas de panne matériel de l'un des disques SAS. Cette tolérance aux pannes est nécessaire sur une station de travail. Il faut bien comprendre que cette architecture RAID 1 utilise nos 2 disques physiques pour afficher seulement 1 seul disque logique et qu'on a donc une capacité utilisable de seulement 1,2 To sur la machine.

Un des objectifs du stage est d'améliorer l'architecture disque afin que notre programme de capture puisse tourner sur plusieurs mois sans que cela ne pose de problème concernant la volumétrie disque disponible. Nous disposons pour cela de 2 disques dur SAS de 12 To à installer sur la station de travail.

Etat de l'art

Dans ce projet, la notion de parallélisation est prépondérante étant donné que nous avons pour objectif de faire en sorte que le programme de capture profite de l'architecture biprocesseur de la machine, ce qui n'était pas le cas jusqu'à maintenant. Pour y parvenir, il nous faut arriver à paralléliser le programme afin qu'il puisse être distribué équitablement sur les 2 processeurs. Ainsi, à travers le contexte du projet, matériel et logiciel, des outils sont à notre disposition afin de paralléliser un programme en C++ sur Windows Server. Nous détaillerons ici certains d'entre eux, qui, malgré le fait qu'ils n'ont pas forcément été retenus pour notre projet, restent néanmoins des outils robustes de parallélisation.

OpenMP

OpenMP (Open Multi-Processing) sera détaillé plus tard dans le rapport mais il est important de noter que c'est l'outil qui a été retenu pour notre projet. C'est une interface de programmation pour le calcul parallèle. Un des principaux avantages d'OpenMP est sa disponibilité, en effet, l'outil est pris en charge sur de nombreuses plateformes telles que GNU/Linux, OS X mais encore Windows pour de nombreux langages comme C, C++ et Fortran. Voici ci-dessous un exemple basique d'une utilisation d'OpenMP

```cpp
// omp_atomic.cpp
// compile with: /openmp
#include <stdio.h>
#include <omp.h>

#define MAX 10

int main() {
   int count = 0;
   #pragma omp parallel num_threads(MAX)
   {
      #pragma omp atomic
      count++;
   }
   printf_s("Number of threads: %d\n", count);
}
```

A travers cet exemple nous pouvons également noter une des forces de l'outil. En effet, on peut remarquer la présence du mot clé « #pragma » indiquant une directive. Les directives définissent au programma quel type de séquence de code doit être exécuté (« for », « parallel »,…) mais également comment et qui doit l'exécuter (« master », « barrier » ,…).

Globalement, OpenMP est l'un des meilleurs outils disponibles pour la parallélisation. Enfin, les directives sont une bonne façon d'organiser correctement notre code surtout dans notre cas où de nombreuses librairies se mélangent car l'outil reste malgré tout assez séquentiel dans son écriture.

Parallel Patern Library (PPL)

La bibliothèque de modèles parallèles permet d'utiliser un modèle de programmation différent que celui fournit par la bibliothèque de base de C++ (std). Cette bibliothèque « éloigne » le développeur de la parallélisation car il ne gère pas totalement la création ni l'exécution du code parallélisé car la bibliothèque automatise le tout d'elle-même.

La bibliothèque PPL permet notamment de :

- Paralléliser des tâches
- Utilisation et développement d'algorithmes parallèles
- Utilisation de conteneurs et objets parallèles

Comme dit précédemment, la bibliothèque remplace certains outils fournis de base par la bibliothèque std de C++ comme l'exemple ci-dessous :

```
// Use the for_each algorithm to compute the results serially.
elapsed = time_call([&]
{
   for_each (begin(a), end(a), [&](int n) {
      results1.push_back(make_tuple(n, fibonacci(n)));
   });
});
wcout << L"serial time: " << elapsed << L" ms" << endl;

// Use the parallel_for_each algorithm to perform the same task.
elapsed = time_call([&]
{
   parallel_for_each (begin(a), end(a), [&](int n) {
      results2.push_back(make_tuple(n, fibonacci(n)));
   });
```

On peut remarquer que 2 exécutions de la même tâche sont réalisées. La première utilise l'outil «for_each » fournit par la bibliothèque de base std de C++ tandis que la deuxième utilise la version parallèle fournit elle par la bibliothèque PPL « parallel_for_each ». Au niveau de l'optimisation, étant donné que la bibliothèque gère d'elle-même la parallélisation les threads sont donc optimisés.

Cette bibliothèque est un excellent outil de parallélisation, néanmoins nous avons décidé de ne pas l'utiliser à cause du fait que l'utilisation de la bibliothèque automatise beaucoup de choses sur lesquelles nous préfèrerions avoir la main dessus et également car nous avons déjà utilisé un outil de parallélisation dans le passé (OpenMP).

Thread

Thread provient de la bibliothèque std de base de C++ et permet d'utiliser des threads basiques. Cet outil est pratique pour apprendre la parallélisation ou encore pour une parallélisation simple. Cette raison nous a poussé à écarter la possibilité d'utiliser Thread à cause du contexte de réalisation du projet qui n'est pas adapté dans notre cas. En effet, les outils présentés plus haut permettent de plus facilement gérer les boucles, ce qui nous évite de devoir réécrire le code sur lequel nous travaillons et d'une manière plus optimisée. Ci-dessous, un exemple basique de l'utilisation de Thread.

```cpp
// thread example
#include <iostream>       // std::cout
#include <thread>         // std::thread

void foo()
{
  // do stuff...
}

void bar(int x)
{
  // do stuff...
}

int main()
{
  std::thread first (foo);     // spawn new thread that calls foo()
  std::thread second (bar,0);  // spawn new thread that calls bar(0)

  std::cout << "main, foo and bar now execute concurrently...\n";

  // synchronize threads:
  first.join();                // pauses until first finishes
  second.join();               // pauses until second finishes

  std::cout << "foo and bar completed.\n";

  return 0;
}
```

Amélioration du programme de capture

Présentation du SDK de AVerMedia

L'utilisation du SDK (Kit de Développement Logiciel) de Avermedia nous a été imposé en raison de l'utilisation des cartes Avermedia CE 314 HN pour acquérir les flux des chaînes.

Le SDK de Avermedia est une boîte à outils de développement riche en fonctionnalités qui offre un ensemble complet de modules fonctionnels pour développer de manière flexible et efficace des applications verticales. Le matériel AVerMedia est accessible à l'aide de l'API logicielle pour optimiser les performances et ajouter de la valeur aux solutions fournies.

Le SDK est basé sur une conception de module fonctionnel, fournissant l'initialisation et la configuration des paramètres vidéo et audio, la capture, l'encodage/décodage, l'enregistrement, l'édition, la diffusion en continu, l'accélération matérielle, le rendu et la lecture. Il prend également en charge divers langages de programmation.

Le SDK de Avermedia est effectivement utilisable avec de nombreux langages de programmation mais l'utilisation de C++ nous a été « imposé ». Hugo PERVEYRIE ayant développé le programme de capture en C++ pour différentes contraintes, nous avons améliorer le programme dans la même continuité. Romuald DURET n'ayant pas travaillé auparavant sur ce programme, il a tout d'abord fallu qu'il comprenne globalement le fonctionnement de celui-ci pour que nous commencions à travailler dessus.

Nous nous sommes aidés de la documentation complète du SDK de AVerMedia en anglais sur ce lien : http://ftp2.avermedia.com/SDK/Linux_HDCapSDK_EN.pdf

Cette documentation est assez complexe puisque les noms de fonctions ne correspondent parfois pas à leur version en C++ et certains paramètres varient entre la documentation et l'API C++. Il est important aussi de notifier que l'ensemble des fonctions utilisés dans l'API ne sont pas modifiables et font donc du SDK de AVerMedia une solution qui peut être contraignante pour gérer la capture d'image.

Présentation de Open MP

Open MP (Open Multi-Processing) est une API pour le calcul parallèle. Celle-ci est utilisable avec les langages de programmation C, C++ et Fortran.

L'API s'utilise sous la forme d'un ensemble de directive, d'une bibliothèque logicielle traditionnelle et aussi de variable d'environnement configurable.

L'intérêt d'Open MP est qu'elle permet de développer des applications parallèles tout en restant proche du code séquentiel.

Solution Multi Processeur

Groupes de processeurs

Le système d'exploitation, par défaut, va faire faire 1 groupe de processeur logique pour chaque processeur physique. Dans notre cas on aura donc 2 groupe avec 40 processeurs physiques chacun.

Une application Windows peut nativement accéder qu'a un seul groupe de processeur ce qui correspond à un seul processeur physique. Afin de changer cela, on doit être en mesure de manipuler les groupes de processeurs avec C++. Il existe pour cela des fonctions et structures Windows permettant de manipuler les affinités de groupe de chaque thread.

La structure « GROUP_AFFINITY » permet de manipuler l'affinité de groupe d'un processeur logique. Les fonctions « GetThreadGroupAffinity() » et « SetThreadGroupAffinity() » permettent de récupérer et de définir le groupe d'affinité d'un processeur logique.

Pour en savoir plus, vous pouvez vous rendre sur ce lien qui documente toute cette partie :

https://learn.microsoft.com/fr-fr/windows/win32/procthread/processor-groups

Blocage 64 Thread

Après renseignement sur beaucoup de forum, il semblerait qu'il y ait un blocage sur le nombre de thread par Microsoft. En effet, Open MP permet de créer autant de thread que l'on souhaite mais Visual Studio bloque ce nombre à 64 et ne prend pas en compte les threads supplémentaires. Vous pouvez retrouver quelques informations sur ce problème à cette adresse (réponse officielle de Microsoft) :

https://social.msdn.microsoft.com/Forums/vstudio/en-US/772c339d-27d5-4992-bcff-8a30f5077edf/couldnt-create-more-than-64-openmp-threads-in-a-test-application?forum=parallelcppnative

Programme de test

En utilisant toutes les structures et fonction de la partie groupe de processeurs, nous avons réussi à manipuler les affinités de groupe afin de changer leur appartenance et donc utiliser des thread du processeur physique inutilisé.

Une répartition égale des threads par rapport au processeur physique est faite afin de ne pas surcharger un processeur plus qu'un autre.

Vous pouvez retrouver le code complet du programme de test en annexe.

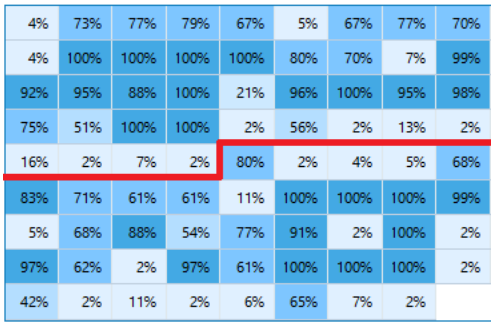
Pour tester cette affection de thread, nous avons développé une fonction de surcharge qui occupe totalement un thread pendant une vingtaine de seconde, ce qui nous laisse le temps d'observer les résultats.

Vous pouvez retrouver cette fonction ci-dessous :

```
void testSurcharge(int N) {
    int i = 0;
    while (i < N) {
        int* a = new int;
        i++;
    }
}
```

Résultats

Dans le gestionnaire des tâches, on peut accéder à l'utilisation de chacun des processeurs logiques de nos 2 processeurs physiques. On peut voir sur l'image ci-dessous le pourcentage d'utilisation des 40 processeurs logiques de notre premier processeur physique puis celui de nos 40 processeurs logiques du deuxième processeur physique.



L'image nous montre bien que les deux processeurs sont utilisés et donc que notre programme produit les résultats attendus. Il faut donc intégrer cette solution de gestion de groupe dans le programme de capture.

Intégration

Après avoir vérifié le fonctionnement de la solution multiprocesseur, nous avons débuté l'intégration. Celle-ci s'est déroulé assez simplement étant donné que Hugo PERVEYRIE est l'auteur du programme de capture.

Globalement, la fonction s'est bien « insérée » dans le code mis à part pour la gestion des objets. Nous avons dû recréer des listes pour l'objet de capture et le numéro de chaîne car OpenMP a des conditions très spécifique sur les boucles for qu'il peut paralléliser.

```
int sizeDWORD = 0;
DWORD dwordList[24];
int sizeHANDLE = 0;
HANDLE handleList[24];

for (map<DWORD, HANDLE>::iterator it = g_mhObject.begin(); it !=
g_mhObject.end(); ++it) {
    dwordList[sizeDWORD] = it->first;
    handleList[sizeHANDLE] = it->second;
    sizeDWORD++;
    sizeHANDLE++;
}
```

Le programme est similaire en tout point à notre programme de test mais appelle la fonction de capture à la place de celle de surcharge. Cependant cette fonction pose un problème.

En effet, le problème est que notre solution ne fonctionne plus dès lors que le programme utilise la fonction « AvecCaptureImageStart ». Celle-ci redéfinit les affinités de groupes et ne nous permet d'utiliser les deux processeurs.

Afin de prouver que cette fonction est bien problématique nous avons appelé la fonction de surcharge à sa place et avons récupérer ce rendement des processeurs :

10%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	20%	2%	7%	4%	4%	5%
3%	5%	48%	5%	4%	6%	5%	6%	3%
2%	4%	3%	43%	2%	2%	2%	2%	2%
2%	3%	3%	2%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	100%
100%	100%	100%	100%	100%	100%	100%	100%	

On peut constater qu'en dehors de l'appel de cette fonction, les 2 processeurs physique sont bel et bien utilisés.

La réécriture de cette fonction sort du cadre de ce projet et fait partie intégrante du PRD de Hugo PERVEYRIE. Nous n'avons donc pas cherché à résoudre ce problème majeur qui sera résolu ultérieurement à travers la phase conception du PRD de Hugo PERVEYRIE.

Amélioration de la station de travail

Extension de l'architecture disque

En plus des deux disques SAS de 1,2 To présent sur la machine, le LIFAT avait précédemment fait un devis pour ajouter deux autre disques SAS de 12 To chacun. L'installation de ses disques a été réalisés par des techniciens étant donné qu'il fallait ouvrir la station de travail ce qui nous est interdit. Un compte rendu nous a été retransmis concernant la configuration de ses 2 disques dur par rapport aux contrôleur Dell Perc H330. En effet il y avait 2 configurations possible pour l'installation des disques dur :

- RAID 0 : Les données s'écrivent sur les 2 disques en même temps. Cela présente un gros gain de performance puisque la bande passante est mutualisé mais en cas de perte d'un disque dur, c'est toutes les données qui sont perdues.
- JBOD (Just a Bunch of Disks) : Les données s'écrivent sur 1 des 2 disques. Pas de gain de performance.

En raison de la réduction massive d'écriture de données du programme de capture grâce au PRD de Hugo PERVEYRIE, les gains de performance offert par la solution RAID 0 ne nous sont pas utile et on va préférer la solution JBOD qui sera plus simple et moins risqué.

Étude sur les drivers des cartes AVerMedia

En lien avec ce projet ASR, un problème est apparu et cette partie aura pour but de mettre les choses à plat afin de retranscrire au mieux la situation qui pose un problème. En effet, les cartes AVerMedia avaient besoin d'installation de drivers, certains sont signés, d'autres non. L'installation de drivers sur des machines officielles de l'école nécessite un ticket afin que la procédure voulant être réalisée soit validée par la DSI. Cette installation nécessite des droits spéciaux pouvant valider l'installation de ces drivers.

Ainsi, des comptes ont été créés pouvant normalement gérer l'installation. Malheureusement, l'installation n'a pas fonctionné dans la globalité. L'erreur proviendrait du fait que Windows bloquerait l'installation des drivers car étant signés pour Windows 10 d'après les drivers fournis par le constructeur (la machine fonctionne elle sur Windows Server 2019). Le problème a été mis de côté quelques temps car les nouveaux disques étaient en installation. Mais maintenant que la machine est fonctionnelle il est important de réfléchir à de potentielles solutions à ce problème.

De notre côté, résoudre le problème dans sa totalité semble compliqué surtout avec le peu de temps que nous avons. Néanmoins à travers certains articles nous pouvons proposer des possibles solutions. Elles ne remplaceront pas l'installation correcte des drivers mais peuvent aider potentiellement les

personnes qui devront s'occuper de ce problème. En effet, d'après l'article : http://www.kms-quebec.com/fr/SignatureDePilotes2016.pdf, il y aurait la possibilité de désactiver la vérification des signatures des drivers lors du chargement. Cette solution n'est néanmoins clairement pas conventionnelle et sera encore moins acceptée comme une version correcte de la machine.

Difficultés rencontrées

Redéploiement des cartes d'acquisition

En raison des règles du LIFAT concernant la station de travail, nous ne pouvions redéployer par nous même les cartes d'acquisition de la station de travail. La disposition actuelle fait que 4 cartes envoient leurs données vers 1 processeur et les 2 restantes vers l'autre processeur. L'objectif était de déplacer une des cartes pour que la disposition soit équivalente sur les 2 processeurs afin de ne pas surcharger un des bus de données.

Malheureusement, pour effectuer ce redéploiement nous avions obligatoirement besoin d'un technicien et celui-ci n'était pas disponible durant la durée de notre projet.

Test pleine charge 480 images / seconde

En raison du problème précédent, les tests de charge n'ont pas été réalisé car un des bus de données est surchargés à cause de la présence de 4 des cartes d'acquisition. Les tests de charges seront donc réalisés ultérieurement lorsque le redéploiement aura eu lieu.

Conclusion

Ce projet nous a apporté de nouvelles connaissances sur la programmation multi processeur. Cette expérience peu commune sur l'Architecture des Système aurait été compliqué sans avoir accès à une machine aussi puissante. De plus, l'étude des diverses améliorations tel que l'installation des disques et une réinstallation des drivers nous a aussi apporté de nouvelles expériences, par exemple avec les interfaces SAS qui sont peu communes en dehors des entreprises.

Nous avons rencontré aussi quelques difficultés que nous avons expliqués et qui nous ont empêché de réaliser les tests pleine charges à temps. Ces difficultés font malheureusement parti des projets et nous feront plus attention lors de nos prochains projets aux contraintes de temps que peuvent poser un rendez-vous avec un technicien.

Annexe

```
int main() {

    GROUP_AFFINITY GroupAffinity1;
    GROUP_AFFINITY GroupAffinity2;

    #pragma omp parallel num_threads( 24 )
    {

        HANDLE thread = GetCurrentThread();
        GROUP_AFFINITY previousAffinity;
        GetThreadGroupAffinity(thread, &previousAffinity);

        if (omp_get_thread_num() > 12) {
            GroupAffinity1.Reserved[0] = 0;
            GroupAffinity1.Reserved[1] = 0;
            GroupAffinity1.Reserved[2] = 0;
            GroupAffinity1.Group = 0;
            GroupAffinity1.Mask = 1 << (omp_get_thread_num() % 12);
            if (SetThreadGroupAffinity(thread, &GroupAffinity1,
&previousAffinity)) {
                printf_s("Thread set to group 0: %d\n", omp_get_thread_num());
            }
        }
        else {
            GroupAffinity2.Reserved[0] = 0;
            GroupAffinity2.Reserved[1] = 0;
            GroupAffinity2.Reserved[2] = 0;
            GroupAffinity2.Group = 1;
            GroupAffinity2.Mask = 1 << (omp_get_thread_num() % 12);
            if (SetThreadGroupAffinity(thread, &GroupAffinity2,
&previousAffinity)) {
                printf_s("Thread set to group 1: %d\n", omp_get_thread_num());
            }
        }

        #pragma omp for
        for (int i = 0; i < 24; i++) {
            testSurcharge(10000000);
        }
    }
    return 0;
}
```

```
si.initStreamImage(dwNumChannel);
g_mhObject = si.getAllHandleObject();

int nbThreads = 0;

GROUP_AFFINITY GroupAffinity1;
GROUP_AFFINITY GroupAffinity2;

int sizeDWORD = 0;
DWORD dwordList[24];
int sizeHANDLE = 0;
HANDLE handleList[24];

for (map<DWORD, HANDLE>::iterator it = g_mhObject.begin(); it !=
g_mhObject.end(); ++it) {
    dwordList[sizeDWORD] = it->first;
    handleList[sizeHANDLE] = it->second;
    sizeDWORD++;
    sizeHANDLE++;
}

int nbRepart = NB_THREADS / 2;
cout << nbRepart << "ici" << endl;

#pragma omp parallel num_threads( 24 )
{

    HANDLE thread = GetCurrentThread();
    GROUP_AFFINITY previousAffinity;
    GetThreadGroupAffinity(thread, &previousAffinity);

    if (omp_get_thread_num() > 12)
    {
        GroupAffinity1.Reserved[0] = 0;
        GroupAffinity1.Reserved[1] = 0;
        GroupAffinity1.Reserved[2] = 0;
        GroupAffinity1.Group = 0;
        GroupAffinity1.Mask = 1 << (omp_get_thread_num() % 12);
        if (SetThreadGroupAffinity(thread, &GroupAffinity1,
&previousAffinity))
        {
            printf_s("Thread set to group 0: %d\n", omp_get_thread_num());
        }
    }
    else
    {
        GroupAffinity2.Reserved[0] = 0;
        GroupAffinity2.Reserved[1] = 0;
```

```
        GroupAffinity2.Reserved[2] = 0;
        GroupAffinity2.Group = 1;
        GroupAffinity2.Mask = 1 << (omp_get_thread_num() % 12);
        if (SetThreadGroupAffinity(thread, &GroupAffinity2,
&previousAffinity))
        {
            printf_s("Thread set to group 1: %d\n", omp_get_thread_num());
        }
    }

    #pragma omp master
    nbThreads = omp_get_num_threads();

    #pragma omp for
    for (int i = 0; i < 24; i++) {
        cout << "Lancement du thread pour channel " << dwordList[i] << " par
le thread " << omp_get_thread_num() << endl;
        captureChannel(ci, dwordList[i], dwResolution, dwSecondDuration,
dwFPS, handleList[i]);
    }
}
```