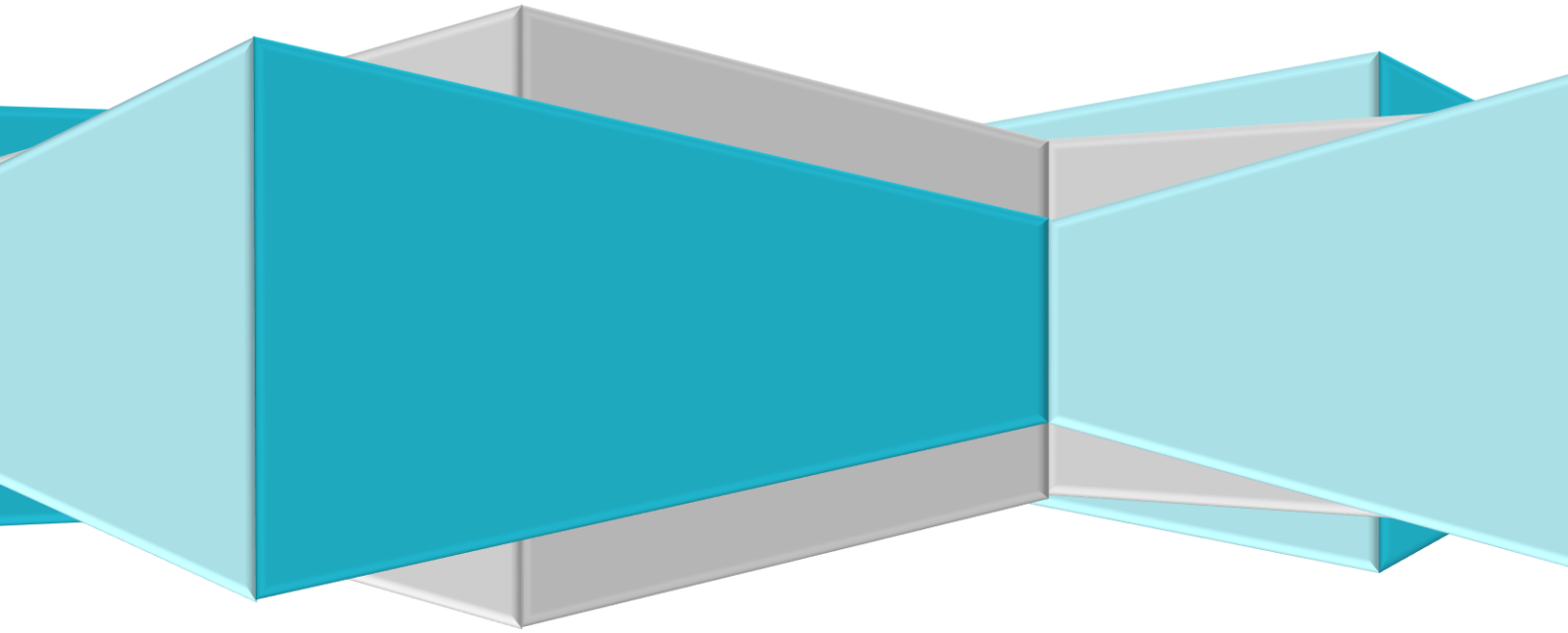
Université François Rabelais de Tours
Polytech Tours
Département informatique industrielle

Sélection de template

Projet collectif



Chef de projet : Guillaume COUE
Thibault ARTUS
Alexandre BILLAY
Johan KARPINSKY
Emilie LEMEE

Encadrant :
Mathieu DELALANDRE

Sommaire

Introduction

Dans le cadre de notre formation à Polytech Tours en quatrième année, nous avons à mettre en œuvre un projet en groupe de cinq à six personnes. Le sujet de notre projet, qui a été choisi par l'ensemble du groupe, concerne la sélection de « Template » sur une image en noir et blanc de type manga.

Nous allons donc d'abord commencer par vous présenter le projet qui nous a été confié afin de resituer le contexte de ce projet, puis le cahier des charges qui a été validé par l'enseignant. Nous vous présenterons l'organisation que nous avons faite pour le projet. Nous vous présenterons ensuite le travail que nous avons effectué divisé en trois différentes parties, définies dans le cahier des charges que nous vous avions fourni en début de projet. La première partie concerne la transformation du fichier d'entrée, la deuxième partie concerne la création de processus (« threads ») parallèles, enfin, la troisième partie se base sur un traitement que l'on vectorise. Nous vous présenterons ensuite la conduite de projet que nous avons mise en œuvre.

Ce dossier présentera le travail que nous avons effectué pour ce projet.

Présentation du projet

Découverte du projet

Le projet de sélection de « Template » concerne l'étude d'une image en noir et blanc (deux niveaux de couleur) afin d'en déterminer la zone la plus caractéristique. Cette étude est effectuée informatiquement, et nécessite beaucoup de ressources pour les calculs qui mènent au résultat.

Les calculs, très gourmands en ressources, ne peuvent être effectués classiquement. Nous mettrons donc en œuvre une architecture de travail parallèle pour le processeur. Nous vectoriserons aussi les instructions pour gagner du temps. Des tests et des comparaisons nous permettrons de déterminer le fonctionnement optimal de notre système.

L'application de cette sélection de template sur l'image concerne les images de manga et pourra être utilisée, dans un temps ultérieur, pour la mise en œuvre de recherche de « scans » de manga (mangas scannés et diffusés illégalement sur internet). Nous travaillerons sur l'image suivante :

Image 1 : Image de travail

La première étape de ce projet se décompose en trois parties majeures. La première partie concerne la lecture du fichier d'entrée et la mise en mémoire afin de pouvoir travailler sur le fichier. La deuxième partie, axée sur la parallélisassions en threads de notre tâche, consistera à étudier le comportement du processeur lorsqu'on lui donne une tache décomposée en plusieurs threads. L'étude nous permettra d'obtenir la configuration optimale de travail du processeur. La troisième partie concerne l'utilisation d'instructions vectorisées. Nous chercherons à montrer l'avantage d'utiliser des instructions vectorisées par rapport aux instructions non-vectorisées.

La structure globale de notre projet sera la suivante :

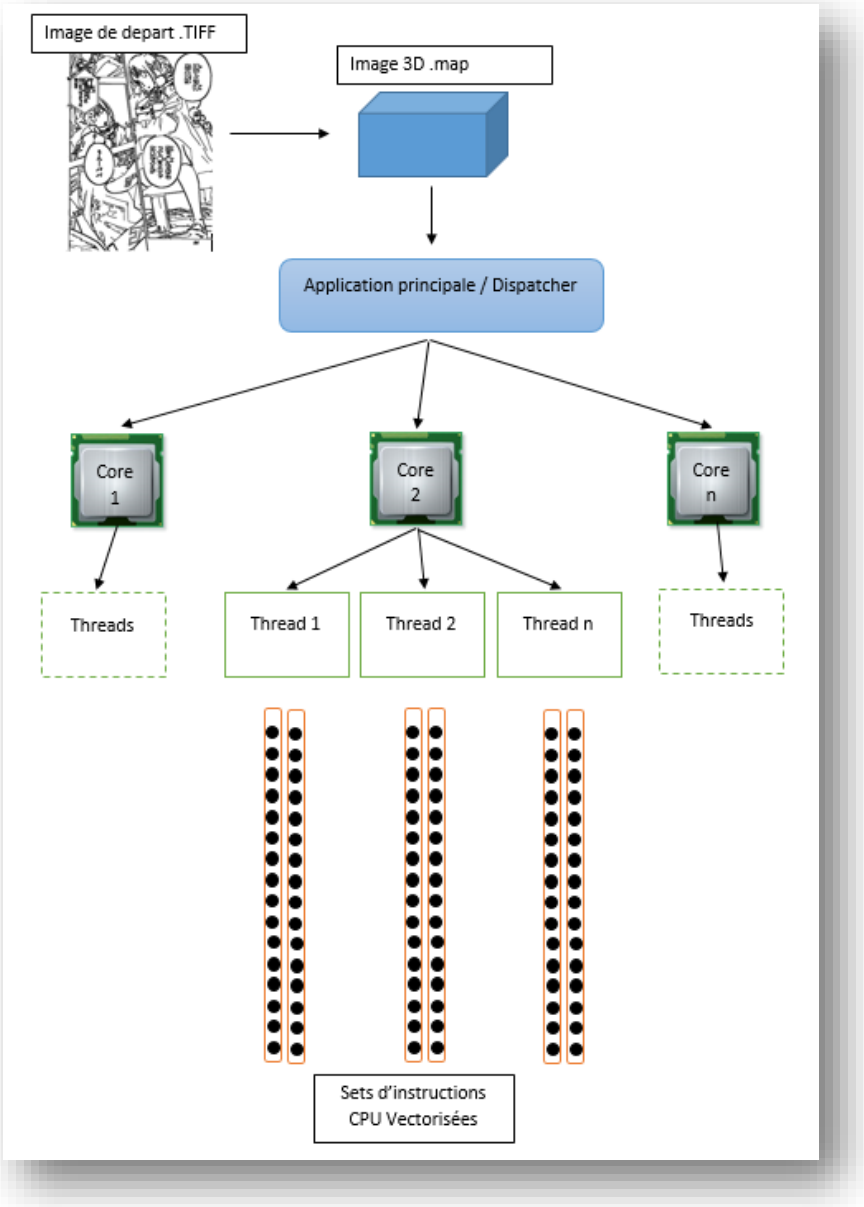


Image 2 : Structure globale

Cahier des charges

Contexte et définition

Le projet consiste à réaliser une sélection de Template sur une image en noir et blanc. C'est-àdire à déterminer, à l'aide d'un traitement informatique, la zone la plus représentative de l'image.

Dans notre cas, nous appliquerons ce traitement à des pages de manga numérisé, la zone sera représentée par un rectangle.

Les techniques mathématiques à mettre en place pour effectuer la sélection sont l'autocorrélation et le calcul de rapport signal sur bruit. Appliqué à notre image, ces calculs requièrent une complexité algorithmique de O (n^2).

Si on prend en fichier de départ une image de 1300x900 et qu'on applique à chaque pixel un traitement de 0.01s approximativement, on se retrouve alors avec un temps total de $((1300*900)*0.01)^2$ = 1584 jours de calcul, ce qui est assez problématique.

On veut donc réduire au maximum le temps de calcul de ces Template. Un des aspects majeurs de ce projet est de mettre en place une structure de calcul parallèle en combinant différentes méthodes pour optimiser au mieux les ressources disponibles.

Outils utilisés

Aucune contrainte ne nous est imposée sur le matériel et les logiciels à utiliser mais, d'après les recherches préliminaires que nous avons effectuées, deux solutions s'offrent à nous :

- Travailler sous Linux avec le compilateur GCC, qui permet la vectorisation ;

- Travailler sous Windows avec l'IDE Visual Studio et le Compilateur Intel ® C++ Compiler, qui permet aussi la vectorisation.

Pour le moment, nous partons sur Windows, car des licences gratuites nous sont proposées sur les produits Microsoft et Intel en tant qu'étudiant et ces outils se prêtent mieux à l'aspect projet et au travail en groupe. De plus, toutes les personnes dans le groupe ont une machine personnelle Windows comportant un processeur Intel d'architecture x64.

Livrables

Des livrables seront délivrés à plusieurs étapes du projet :

- Les premiers correspondent aux 3 premières parties du projet, rendu W18 :

Partie «Fichier d'entrée » :

Un code permettant de lire le fichier .map, d'instancier une image en mémoire, et d'en afficher une partie. Ainsi que les méthodes pour supprimer, ajouter, et accéder aux données.

Partie « Threads » :

Un code permettant la génération contrôlée (nombre, début, fin) de threads effectuant une tâche quelconque.

Un rapport succinct montrant le nombre de threads nécessaire pour un temps d'occupation CPU optimal.

Partie « Vectorisation » :

Un code permettant d'effectuer un traitement (comparaison binaire XOR bit à bit) de façon vectorisée, par registres s'instruction 128 bits (cf. SSE2 Intel) Un rapport succinct comparant la vitesse de traitement d'une même tâche avec et sans la vectorisation.

- Les seconds correspondent au produit fini, en W25, regroupant les 3 premières parties ainsi que le traitement de l'image, et un rapport.

Partie «Algorithme de traitement » :

Algorithme d'autocorrélation.

Partie « Dispatcher de taches entre les threads » :

Capable de répartir les tâches et de récolter les résultats.

Partie « Multi Cœur » :

Extension du traitement actuel, de 1 a tous les cœurs du CPU.

Le contenu détaillé des 3 dernières parties seront développées à la suite de la validation des 3 premières parties.

Organisation du travail

Décomposition du projet

Après avoir vu l'ampleur du projet à réaliser, nous avons décidé de découper le projet en plusieurs parties selon les différents éléments sur lesquels travailler. De plus nous avons décidé avec notre encadrant de choisir un jalon intermédiaire pour faire le point sur l'avancement et revoir les objectifs du projet.

Le premier jalon se décompose en trois parties principales : lire le fichier d'entrée au format « .map » et le mettre en mémoire, faire une étude sur le nombre de thread optimal et faire une étude sur la vectorisation. Une fois que cette première partie sera réalisée, nous effectuerons la deuxième partie qui est la mise en commun du travail effectué en première partie par chacun des groupes, et le développement du traitement de l'image par instructions vectorisées.

La décomposition du travail a été effectuée de la manière suivante :

- Guillaume COUE : gestion de projet, documentation et support aux autres groupes,
- Alexandre BILLAY et Emilie LEMEE : lecture du fichier d'entrée,
- Johan KARPINSKY : Etude des Threads,
- Thibault ARTUS : Vectorisation d'instructions.

Lecture du fichier d'entrée

Présentation du sujet

La première partie concerne le traitement appliqué au fichier d'entrée. Nous travaillons sur une image de type manga constituée de deux niveaux de couleur (noir et blanc). Afin de simplifier le travail à effectuer sur l'image, il nous est proposé d'utiliser un fichier transformé de l'image. Nous ne travaillons donc pas sur l'image au format .tiff, mais sur un fichier au format .map.

Ce format de fichier peut être représenté par une superposition d'images 2D décalées à chaque fois. Le résultat est un fichier « 3D ». Si on choisit un pixel de l'image en « face avant » de ce fichier, on peut obtenir les 256 pixels au-dessus de celui-ci. Ce qui fait que lorsqu'on sélectionne une ligne sur le .map, on obtient un rectangle de 256xlongueur de la ligne, ce qui correspond a un Template.

Notre rôle va donc être de lire ce fichier d'entrée au format « .map » ainsi que de l'instancier en mémoire sous forme de tableau 3D sur lequel nous irons ensuite travailler.

Premier jalon

Code effectué

Cette partie du programme a été réalisée en C++ grâce au logiciel QtCreator.

On aura un fichier main.cpp ainsi que deux classes « cmap » et « arrayvector ». Nous verrons l'architecture de notre programme.

- Main

Ce fichier contient les références à nos deux autres fichiers ainsi que la boucle principale contenant la création d'un objet de type cmap ainsi que l'appel à la fonction « open » et « extractionbinaire ».

- Cmap

C'est dans le fichier cmap.H que nous définissons la classe cmap.

```
class cmap
{
public:
    cmap();
    ~cmap();
    cmap(int width, int heigth);
    void open();
    int getWidth();
    int getHeigth();
    arrayvector* getMt();

private:
    int var_width;
    int var_heigth;
    arrayvector *mt;
};
```

- Arrayvector

Nous allons stocker notre fichier .map dans un tableau 3D. Pour cela, nous utilisons la classe « arrayvector ».

Nous le définissons par le type de données qu'il contient (ici des « double ») ainsi que sa taille en hauteur et largeur. Puis nous définissons les différentes méthodes d'accès à ce tableau.

```
void cmap::open()
{

  int i=0;
  ifstream fichier("/Users/Thunder/Github/ProjetCoTemplate/000.map", ios::in);
        if(fichier)
        {
            cout << "Ouverture du fichier \n" << endl;
            char buffer[ sizeof(long double) ];
            while(fichier)
            {

            for(int x=0;x<this->getHeigth(); x++)
            {
                for (int z=0;z<this->getWidth(); z++)
                {
                    for (int k=0;k<4; k++)
                    {
                        fichier.read((char*)&i, sizeof buffer);
                        this->mt->ecriture(i,x,z,k);
```

Figure 1 méthode de mise en mémoire d'un .map

```
private:
    std::vector<std::vector<std::vector<double> > > array3D;
    int Heigth;
    int Width;
```

Figure 2 Structure de stockage

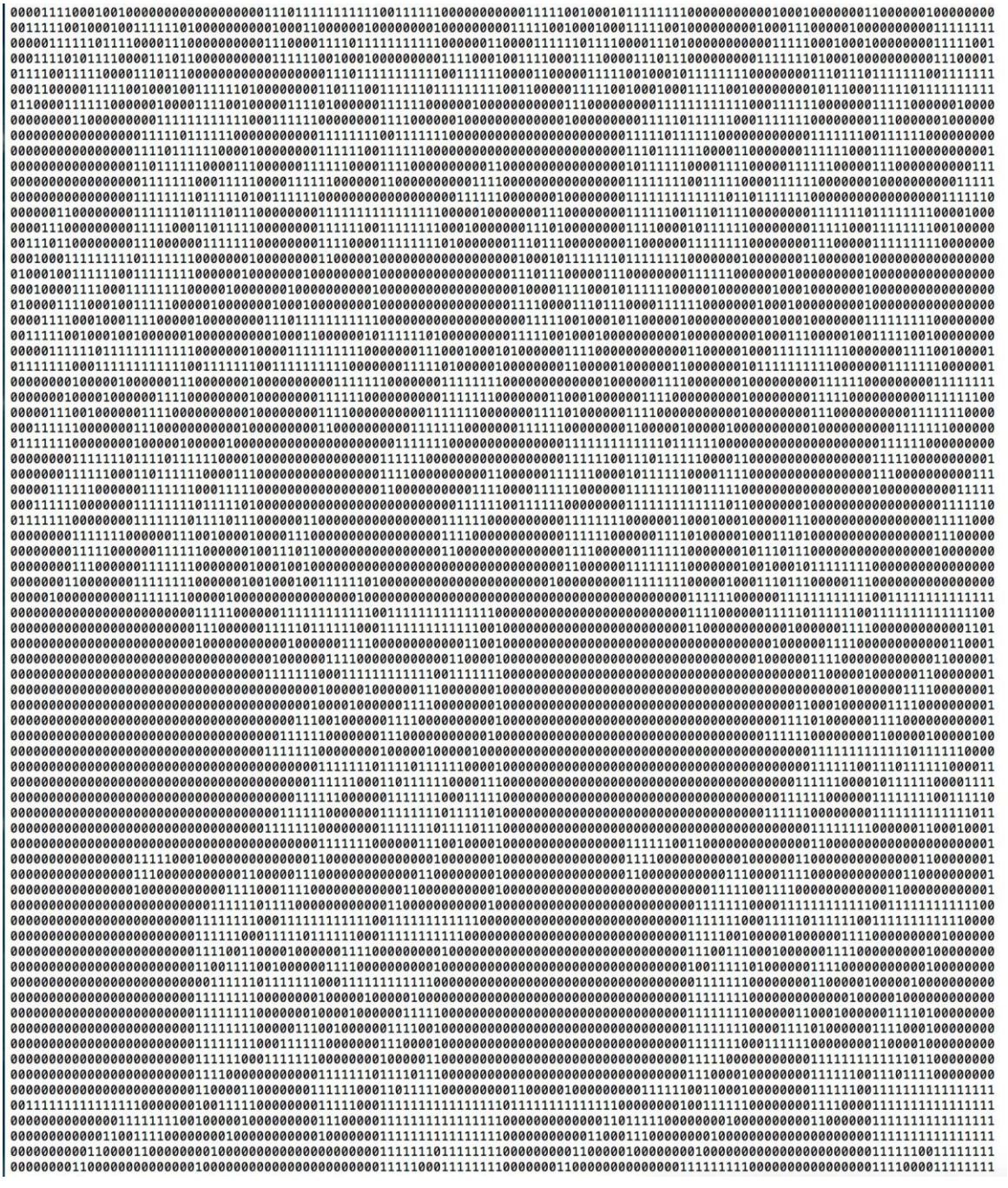


Figure 3 Exemple d'affichage de Template

Deuxième jalon

Nous pensions avoir obtenu un système fonctionnel puisque nous obtenions bien une image contenant des 0 et des 1. Cependant après avoir vu avec l'encadrant, nous avons découvert que notre image rendue ne correspondait pas vraiment à l'image auquel le fichier correspond. Nous avons donc souhaité revoir le code afin que notre système puisse lire correctement le fichier.

› Curseur de fichier

Nous avons découvert qu'il y avait un motif qui se répétait dans notre image rendue. Nous sommes donc partis sur la piste que le système lisait en permanence la même information. Nous avons fait des recherches et avons décidé d'utiliser les curseurs avec seekg et tellg de la bibliothèque fstream.

```
// Recherche de la taille du fichier en position le pointeur de lecture à la fin de celui-ci
fichier.seekg (0, fichier.end);
int length = fichier.tellg();

// Remise du pointeur de lecture au debut du fichier
fichier.seekg (0, fichier.beg);

// Nombre de données presentes dans le fichier
std::cout << "Lecture de : " << length << " char\n";

// Tant que le fichier comporte des données non lues
while(fichier)
{
    for(int x=0;x<this->getHeigth(); x++)
    {
        for (int z=0;z<this->getWidth(); z++)
        {
            for (int k=0;k<4; k++)
            {
                fichier.read((char*)&i, sizeof buffer);
                this->getMt()->ecriture(i,x,z,k);
            };
        };
    };

    // Décalage du pointeur de lecture dans le fichier
    pos += sizeof buffer;

    // While(fichier) lira le prochain chunk a partir de cette position
    fichier.seekg(pos);
```

Nous allons d'abord mettre le curseur au début de notre fichier puis, à chaque fois que nous allons mettre lire un élément, nous décalons le curseur de la taille de cet élément. Nous sommes maintenant sûre de lire des éléments différents.

› Taille de la variable lue

Nous lisons la variable dans un fichier map sous forme d'un « long double ». Nous nous sommes demandé si cette la taille de cette variable n'était pas le problème. Après avoir fait différents tests et comparé avec le programme java, nous nous sommes rendu compte que seule la taille « long double » nous permettait d'obtenir les résultats.

› Affichage de l'image

Un souci important se situait dans la restitution des données une fois écrites dans notre arrayVector. Ces données étant des integer, elles devaient subir une modification pour être affichées dans la console de façon lisible. C'est pourquoi nous avons décidé de les convertir en binaire. Mais un nouveau problème est donc apparu, l'ancienne méthode bien que fonctionnelle ne gardait pas les zéros au début du binaire afficher, ce qui provoquait des décalages dans l'affichage et une visualisation impossible. Nous avons donc mis en place un formatage de la taille de la sortie à 64 bits, et un remplissage des vides provoquer avant le binaire avec des zéros. Le code de la fonction se présente donc maintenant de la façon suivante :

```
void arrayvector::affichageBinaire(int posX, int posY, int depth)
{
    int val = this->array3D[posX][posY][depth];
    string result;

    // Convertisseur int to binary
    while(val!=0)
    {
        if(val%2==0)
        {
            result = "0"+result;
        }
        else
        {
            result = "1"+result;
        }
        val/=2;
    }

    // On renvoi la valeur binaire, en ajoutant les "leading zeros" au debut du bin
    cout << setw(64) << setfill('0') << result;
}
```

Parallélisassions de processus

Présentation du sujet

Afin de gagner le plus de temps sur les calculs possibles, nous souhaitons mettre en œuvre un système de parallélisation de threads. En effet, afin d'améliorer la vitesse d'exécution de notre tache, nous souhaitons la découper en plusieurs processus qui effectueront chacun une part de la tâche. Nous devons aussi déterminer un nombre optimal de threads simultanés afin d'avoir un fonctionnement optimal de notre système.

Le résultat obtenu est qu'un thread « infini » utilise 25% du CPU. Nous émis l'hypothèse que visual studio utilisait un cœur pour traiter un thread vue que le processeur utilisé était un quadri-core. Mais après un autre test sur un processeur 2 cœur nous avons observé les mêmes résultats.

Ces résultats sont valable sous visual studio avec la bibliothèque std ::thread.

Premier jalon

› Code effectué

Lancement des threads

```
int _tmain(int argc, _TCHAR* argv[])
{
    int acc = 0, i = 0;

    int parts = 8;
    std::thread tt[num_threads];

    //Lancement d'un groupe de threads
    for (int i = 0; i < num_threads; ++i) {
        tt[i] = std::thread(call_from_thread,i);
    }

    for (int i = 0; i < num_threads; ++i) {
        tt[i].join();
    }
```

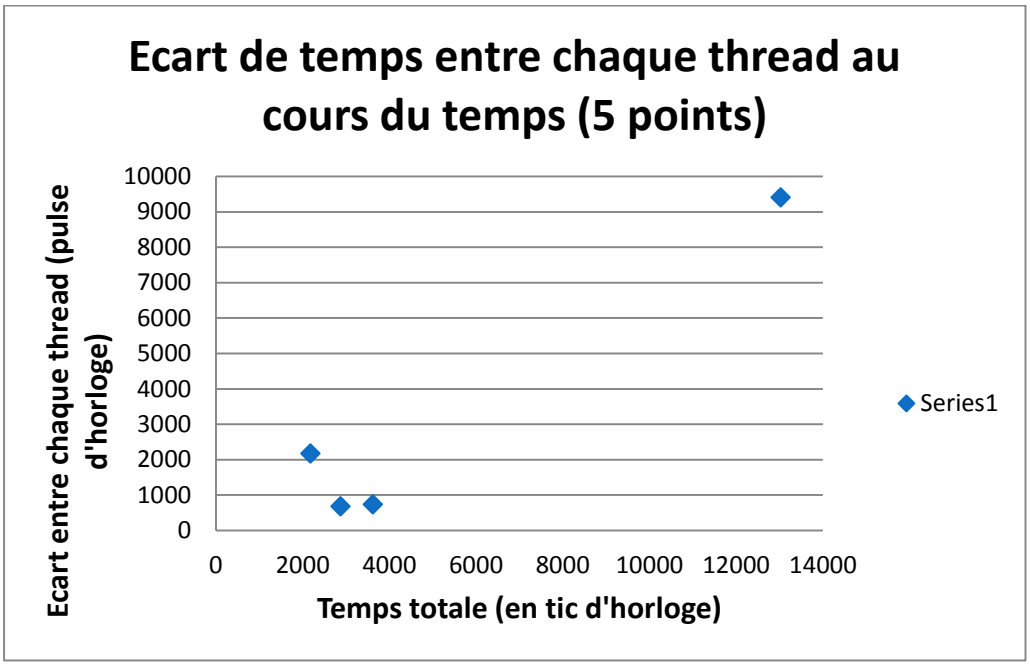
› Tests effectués

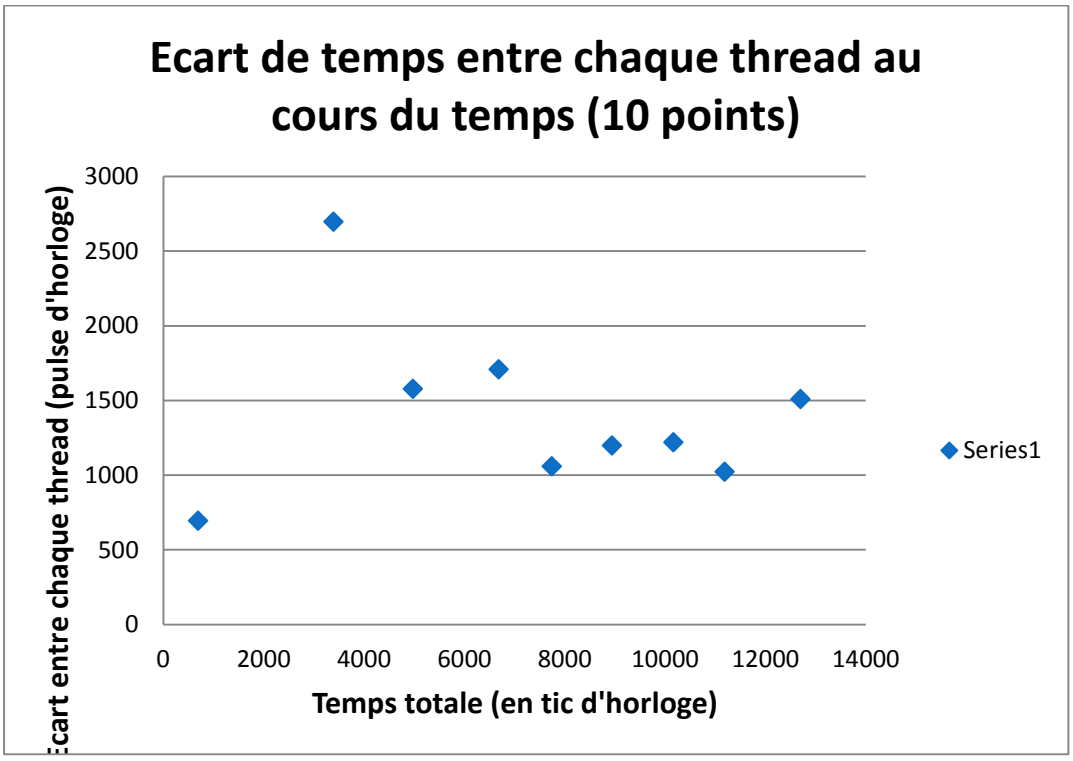
Le temps passé dans le thread était calculé grâce à la fonction « QueryPerformanceFrequency() » qui permet de déterminer combien de coup d'horloge on eux lieux ainsi que la fréquence de cet horloge.

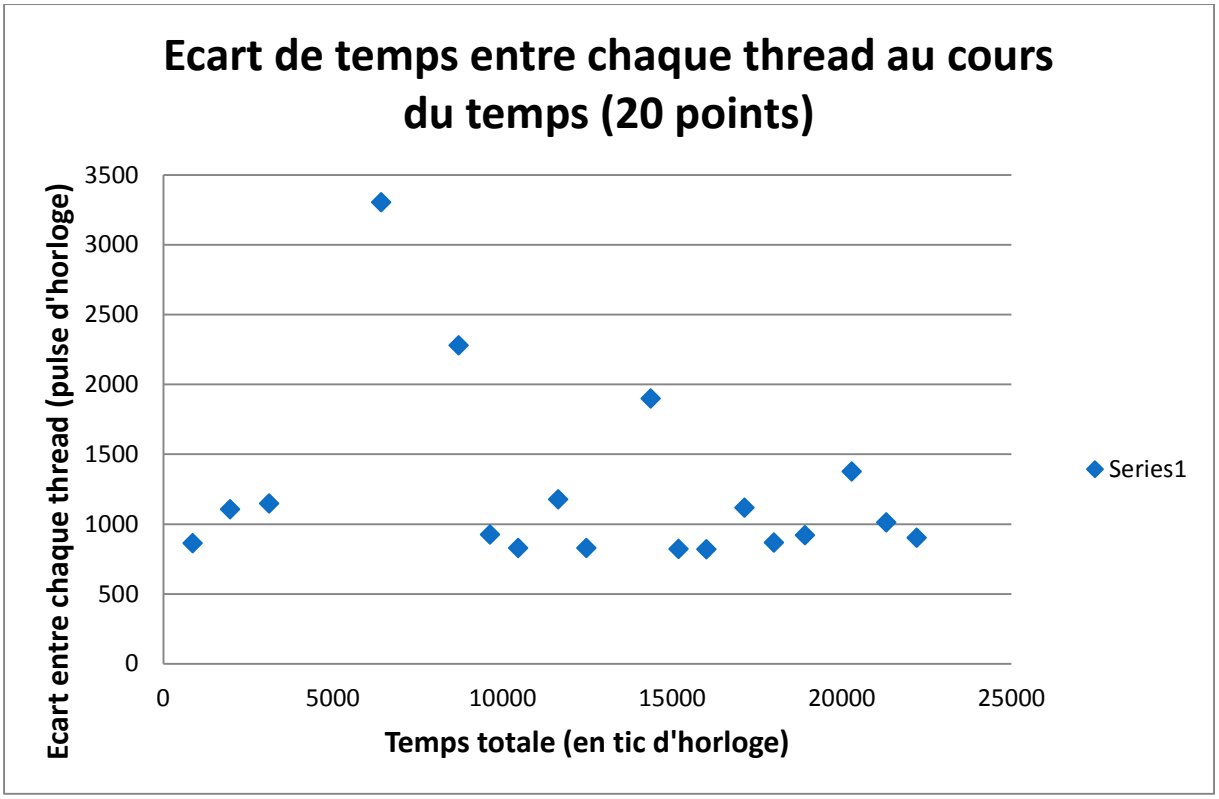
Le premier résultat était que la différence de temps d'exécution entre les premiers threads crée et les autres étaient quasiment nuls.

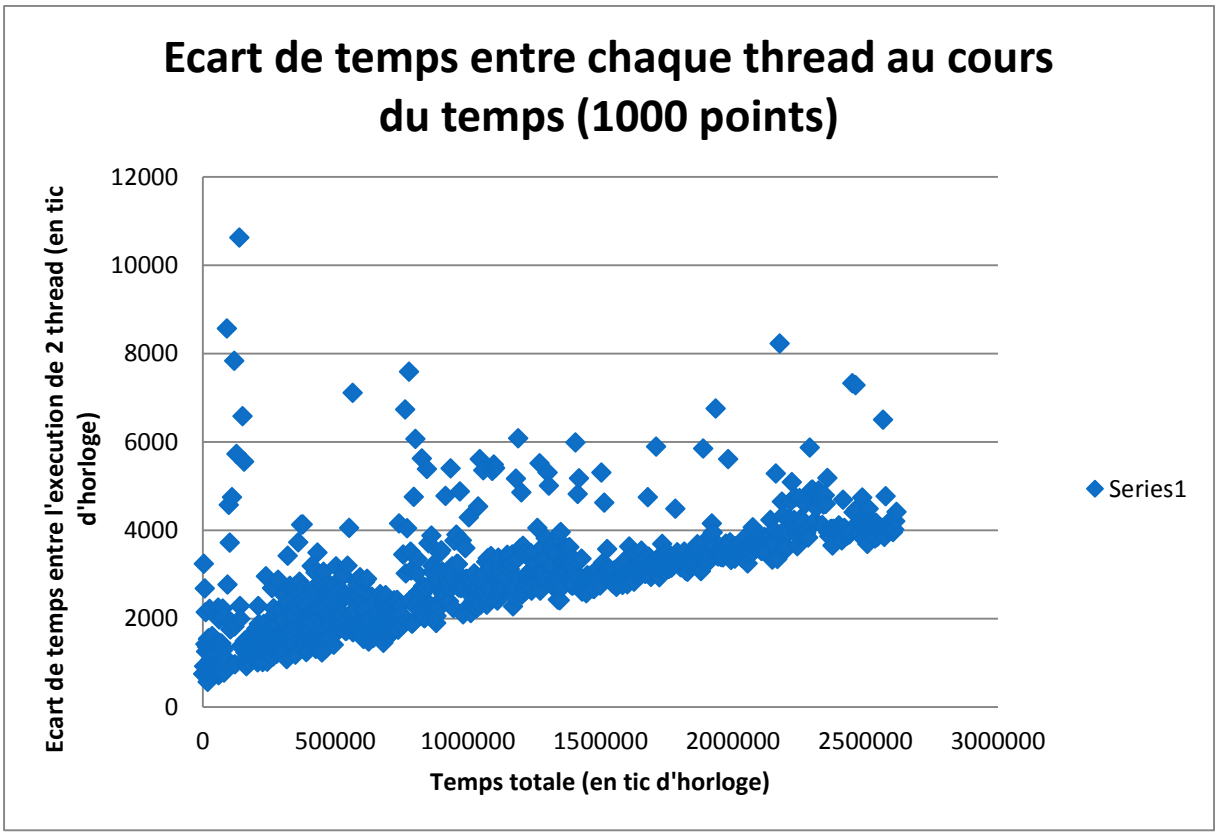
Nous avons alors souhaité connaitre le nombre de thread traité en un temps donné. Pour cela nous avons remplacé le calcul effectué dans le thread par une écriture dans un fichier de la date auquel le thread c'est effectué.

Les test sont effectué pour 5,10,20 et 1000 threads






Ecart de temps entre chaque thread au cours du temps (20 points)
Ecart entre chaque thread (pulse d'horloge)
3500
3000
2500
2000
1500
1000
500
0
0
5000
10000
15000
20000
25000
Temps totale (en tic d'horloge)
Series1


Ecart de temps entre chaque thread au cours du temps (1000 points)
Ecart de temps entre l'execution de 2 thread (en tic d'horloge)
12000
10000
8000
6000
4000
2000
0
0
500000
1000000
1500000
2000000
2500000
3000000
Temps totale (en tic d'horloge)
Series1

Deuxième jalon

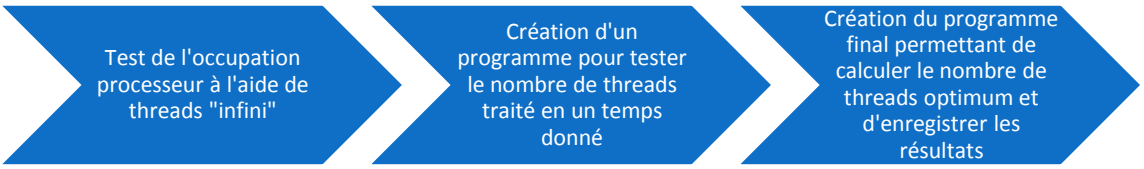
Introduction

L'objectif de cette partie est de déterminer le nombre de threads optimum à utiliser pour réaliser les différents calculs de manière parallèle. Pour cela, le programme à créer devra :

- Gérer le nombre de thread créés
- Leur affecter un calcul
- Enregistrer les résultats

Déroulement de la programmation et des tests

Lors de cette partie nous sommes passés par différentes étapes avant de développer le programme final. Les principales étapes du développement ont été les suivantes :



Fonctionnement du programme

Le programme a pour but de tester le nombre de threads optimum. Pour cela, il fonctionne de la manière suivante :

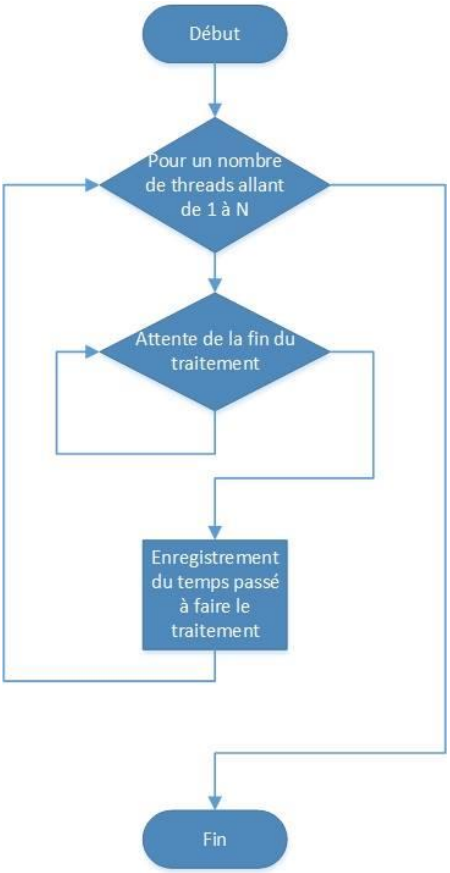
Une première boucle permet de gérer le nombre de threads à tester. Dans cette boucle nous commençons par créer le nombre de threads souhaités pour ensuite leur affecter un calcul.

Ensuite, on vérifie que tous les threads ont fini leur traitement à l'aide de la fonction « join ». Enfin, les résultats obtenus sont moyennés pour avoir une courbe de résultats plus fiable.

Finalement, le programme enregistre les résultats dans un fichier texte où il crée deux colonnes :

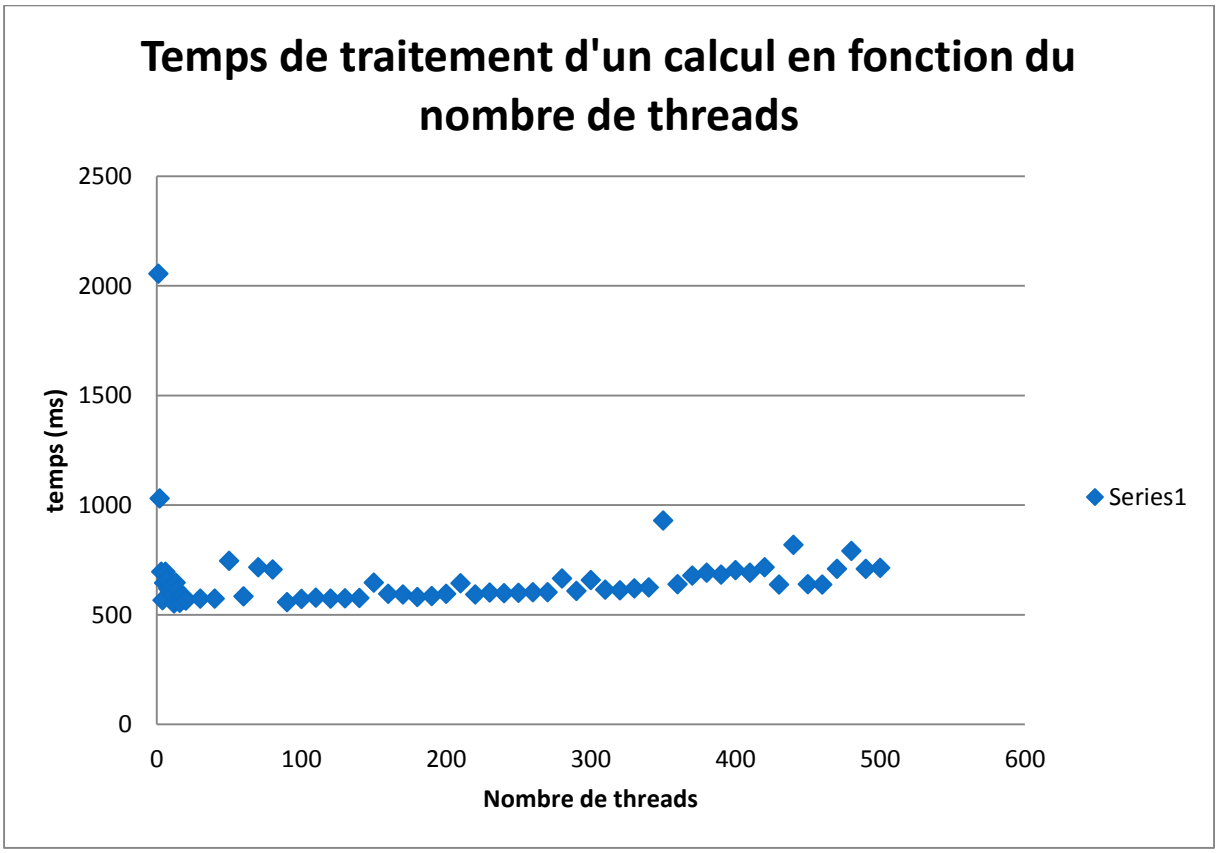
- Une première pour le nombre de threads testés.
- Une seconde pour le temps de traitement.

Le fonctionnement du programme est résumé dans l'ordinogramme ci-dessous.



Résultats obtenus

Résultats obtenus pour un grand nombre de threads

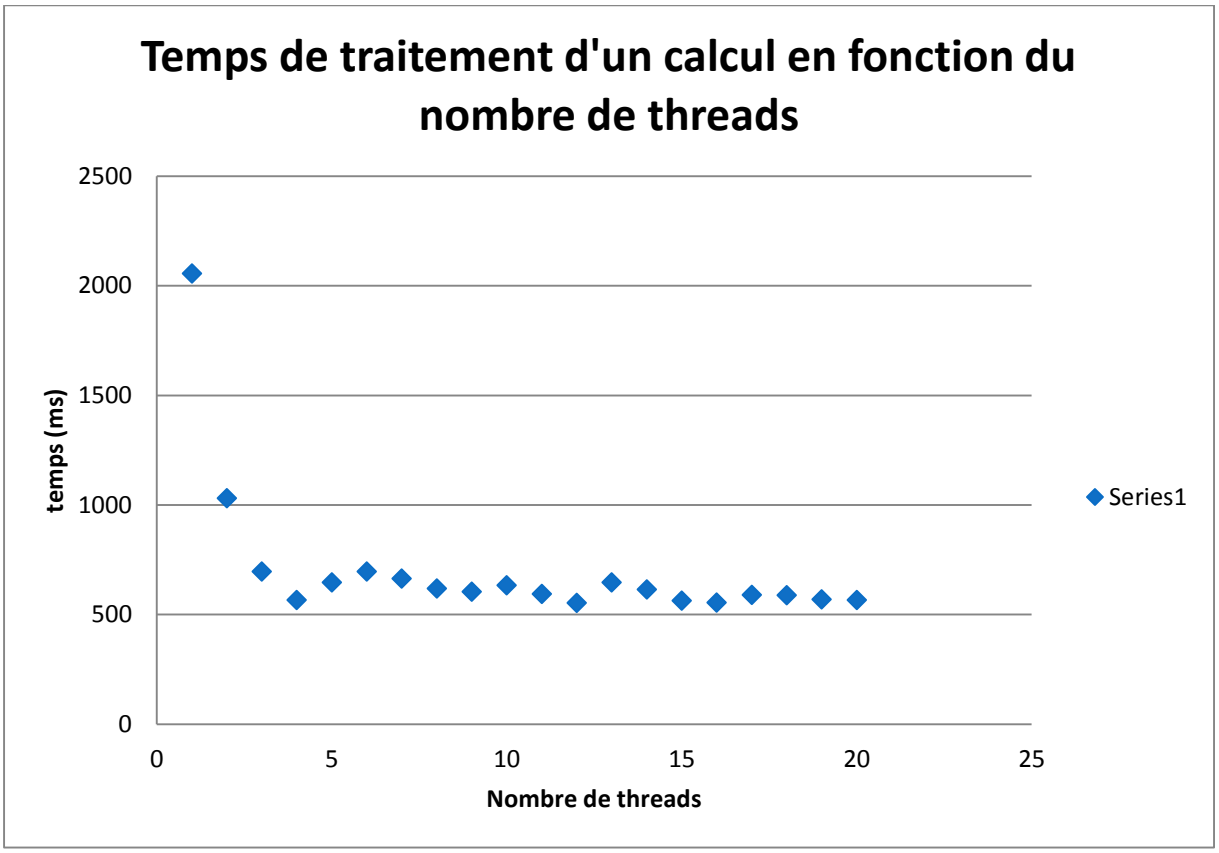


Le fait de mettre en place un nombre de threads trop important va ralentir le traitement. En effet, lorsqu'un thread est créé le contexte est sauvegardé. C'est-à-dire qu'il va enregistrer son chemin dans la pile, il va également enregistrer les variables qui vont être utilisées dans le thread. Une fois le thread terminé il va les restituer et décrémenter la pile.

Toutes ces opérations requièrent des modifications en mémoire, ce qui a pour effet d'allonger la durée des changements de contexte.

Dans le cas de l'exemple ci-dessus l'impact du changement de contexte est négligeable devant le temps de calcul réalisé par les threads. C'est pour cette raison que le temps de calcul avec plus de 4 threads reste relativement proche du temps optimal. Mais, nous avons réalisé des tests avec des calculs dans les threads moins importants qui mettaient bien en évidence ce phénomène.

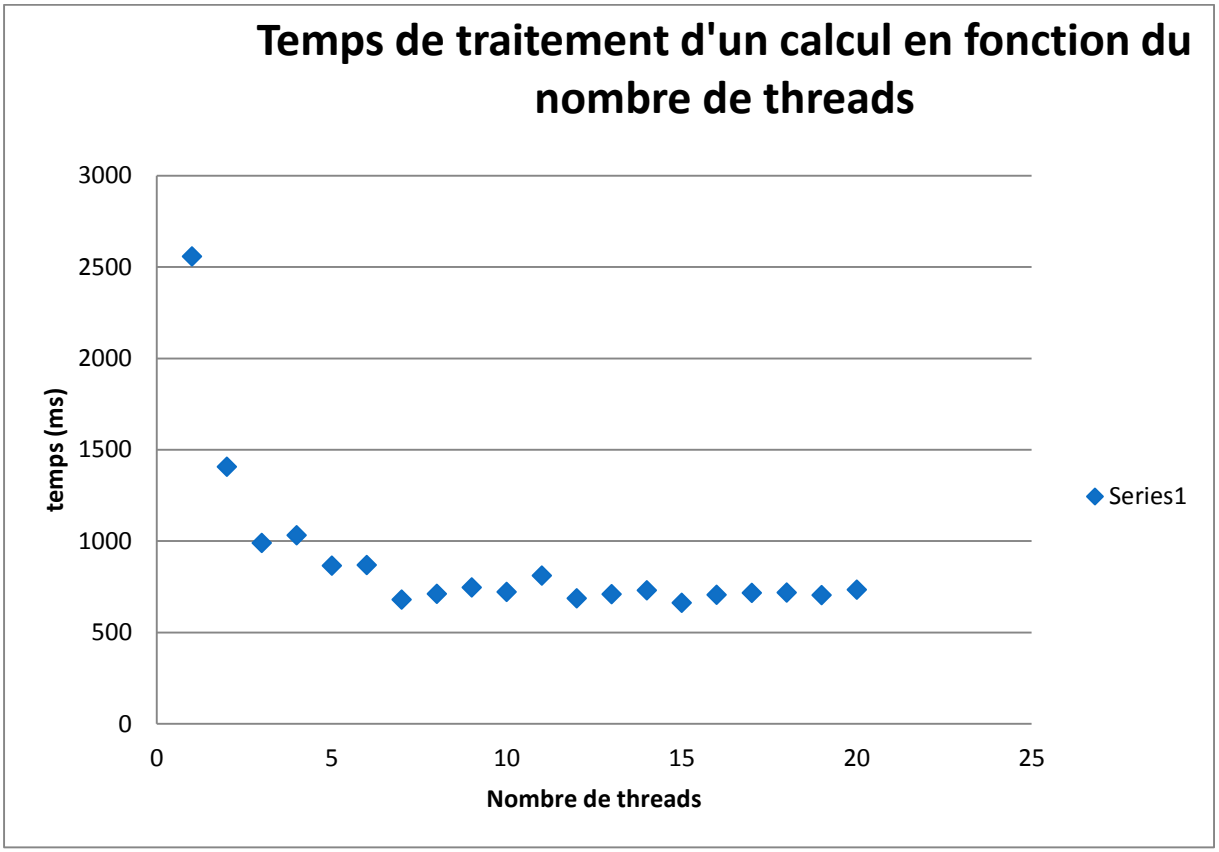
Résultat pour un nombre de threads optimum



Ce graphique permet d'observer de manière plus précise le temps de traitement pour un nombre de threads faible. On peut voir que le nombre de threads optimum est de 4. Cela est lié au processeur utilisé un Intel core i5 2500k (performances indiquées ci-dessous) qui possède 4 cœurs et 4 threads. Il est donc déterminant de connaître correctement sa configuration pour déterminer le nombre de threads optimal.

Performances	
Nb. de cœurs	4
Nb. de threads	4
Fréquence de base	3.3 GHz
Fréquence Turbo maxi	3.7 GHz
PDT	95 W

Par la suite nous avons essayé ce même programme avec un processeur 4 cœurs et 8 threads les résultats sont les suivants :

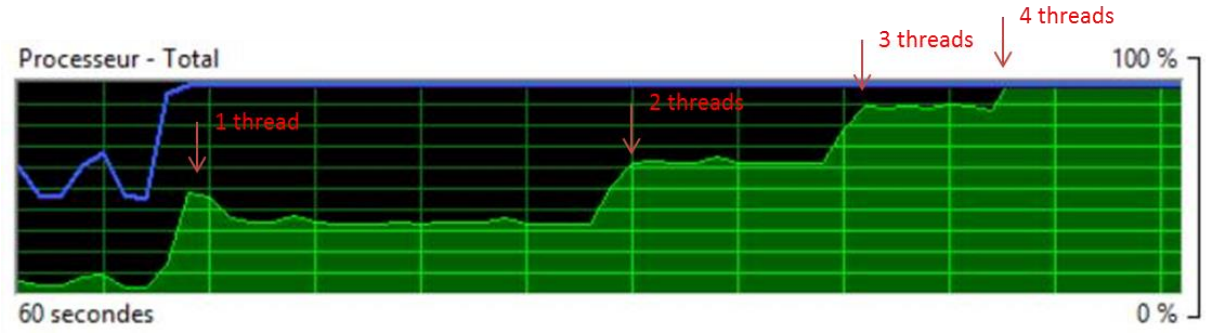


Avec un processeur i7-2630QM à 4 cœurs et 8 threads (performances indiquées ci-dessous) on observe que le nombre de threads optimum est bien de 8 comme attendu.

Performances	
Nb. de cœurs	4
Nb. de threads	8
Fréquence de base	2 GHz
Fréquence Turbo maxi	2.9 GHz
PDT	45 W

Occupation processeur

Le nombre de threads optimum mesuré auparavant se vérifie avec l'occupation CPU ; en effet, on peut observer que durant le traitement avec 1 thread la charge CPU est à 25%, 50% avec 2 threads …



Conclusion

Cette partie nous a permis de mettre en évidence que le nombre de threads optimum était directement lié à l'architecture processeur. En effet, pour un processeur procédant 4 threads nous pouvons observer qu'avec ce nombre de threads l'occupation CPU est de 100% et que c'est avec cette configuration que le temps de traitement d'un calcul est le plus court.

Cette partie nous a permis de nous pencher plus précisément sur le fonctionnement des threads en étudiant par exemple les changements de contexte.

Vectorisation

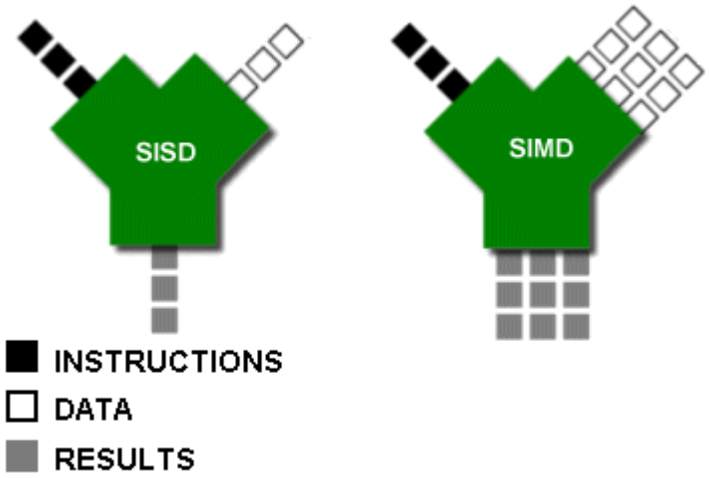
Présentation du sujet

L'algorithme de traitement utilisé pour la sélection de Template repose sur un calcul de base qui est celui du rapport signal sur bruit de 2 images, et ce calcul repose lui-même sur 2 instructions principales qui sont le XOR bit a bit, et le popcount qui compte le nombre de 1 parmi les 0 dans un mot binaire. Notre objectif en utilisant des instructions vectorisé est de réduire grandement le temps global nécessaire pour le traitement en agissant au plus bas, c'est-à-dire sur les instructions de base.

Nous utilisons pour cela une architecture parallèle SIMD (Single Unit on Multiple Data), par opposition au SISD (Single Instruction on Single Data, le fonctionnement traditionnel et le MIMD (Multiple Instruction on Multiple Data). Nous utilisons le SIMD car il est plus performant lorsqu'il y a plusieurs processeurs avec des mémoires indépendantes.

Sachant que nous avions des ordinateurs ayant des processeurs à architecture x64, nous avons dû utiliser les jeux d'instructions SSE2, SSE4.1 et POPCNT.

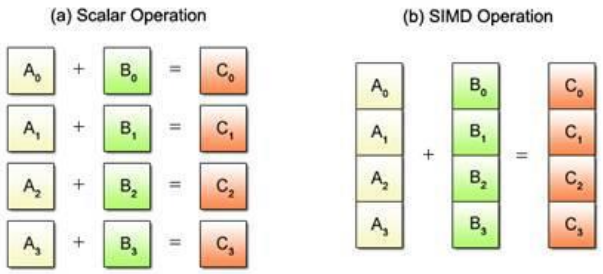
Le principe d'une instruction SIMD est de traiter les données du vecteur indépendamment des autres. Quand on exécute une instruction sur un vecteur, les données présentes dans ce vecteur sont traitées simultanément. Les opérations vont ainsi plus vite.



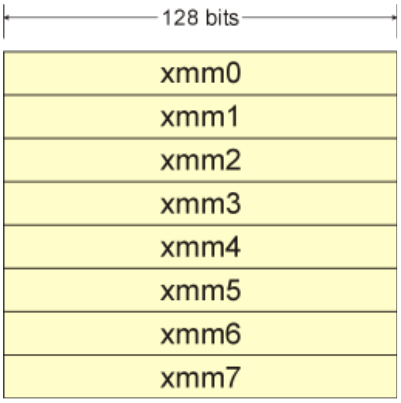
Dans une instruction SIMD, on peut trouver :

- des instructions arithmétiques comme des additions, soustractions, multiplications, etc.
- des opérations logiques (AND, XOR, OR, etc)
- des extractions de données
- des opérations de conversion

Par exemple, ci-dessous, on peut voir l'exécution d'une addition vectorielle représentant une instruction SIMD :



En architecture x86 (SSE), il existe 8 registres SIMD de 128 bits.



En architecture x64, il en existe 8 autres. De xmm8 à xmm15. Pour optimiser les instructions, il faut les remplir au maximum. Pour cela on s'aide du tableau suivant :

Flottant 32 bits	Flottant 32 bits	Flottant 32 bits	Flottant 32 bits
Flottant double précision 64 bits		Flottant double précision 64 bits	
Entier 32 bits	Entier 32 bits	Entier 32 bits	Entier 32 bits
Entier 64 bits		Entier 64 bits	

Entier 16 bits	Entier 16 bits	Entier 16 bits	Entier 16 bits	Entier 16 bits	Entier 16 bits	Entier 16 bits	Entier 16 bits

Dans notre cas, on récupèrera 2 long long, c'est-à-dire, 2 types de 64 bits. On verra plus loin que dans un souci technique, nous avons coupé ces 2 types de 64 bits en 4 types de 32 bits.

Pour rappel, voici la table de vérité de l'opérateur binaire XOR :

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Comparaison entre SISD et SIMD

Afin de mettre en avant l'avantage des instructions vectorisées (SIMD) par rapport aux instructions « normales » (SISD). Nous avons créé un programme qui effectue une fonction SISD et une fonction SIMD avec le jeu d'instruction SSE2 pour mettre en avant la comparaison de ces deux méthodes. On peut ainsi valider le modèle théorique.

Code

Premièrement voici la fonction SISD :

```
//Fonction SISD utilisant une racine carrée ayant pour paramètres:
//"a" qui est l'adresse du float et "N", le nombre de fois que l'on reboucle l'opération
void normal(float* a, int N)
{
  for (int i = 0; i < N; ++i)
    a[i] = sqrt(a[i]);
}
```

Deuxièmement la fonction SIMD :

```
//Fonction SIMD utilisant cette fois ci une instruction __m128 ayant les même paramètres
//que la fonction SISD
void sse(float* a, int N)
{
  // We assume N % 4 == 0.
  int nb_iters = N / 4;

  //Cast de "a" en type __m128
  __m128* ptr = (__m128*)a;

  //Boucle stockant "a" dans "ptr", composé de 4*32 bits
  for (int i = 0; i < nb_iters; ++i, ++ptr, a += 4)
    _mm_store_ps(a, _mm_sqrt_ps(*ptr));
}
```

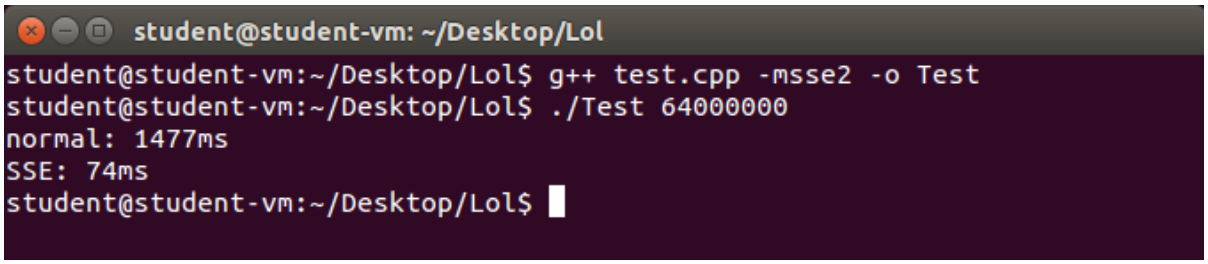
On utilise les instructions SSE sous GCC 4.8.2 avec la bibliothèque xmmintrin.h et math.h.

On compare des racines carrées *sqrt()* en SISD et *__mm_sqrt_ps(),* avec des données en float et en __m128 (soit 4 floats d'un coup) pour la partie SIMD.

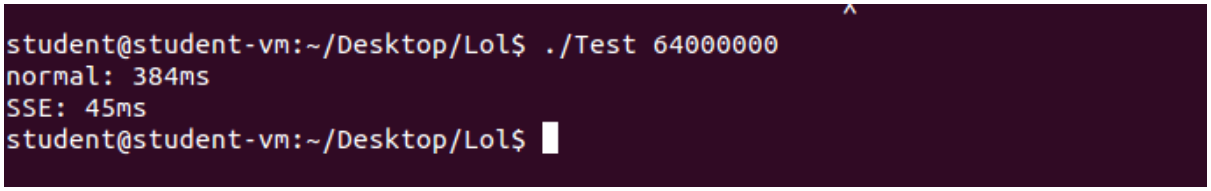
Test

On programme une racine carrée sur un grand nombre de **float** avec GCC 4.8.2 avec Intel Core i7-357U 2.50 GHz.

Sans optimisation (O3).



Avec optimisation (O3).



Ainsi, on peut voir que pour une même opération, une racine carrée, on remarque que le temps passé est de 74 ms en SIMD à la place de 1477 ms en SISD sans l'optimisation –O3 et de 45 ms en SIMD et de 384 ms en SISD.

Vectorisation du calcul Signal/Bruit

Spécifications

Le but de cet algorithme est d'avoir un temps de calcul optimisé par rapport à l'application en Java de M. Delalandre étant de 16 us pour le calcul d'un mot de 64 bits et de 8 us pour le calcul d'un mot de 18 bits.

En entrée de cet algorithme, nous devons avoir 2 mots __m128i → a et b
Ensuite, nous devons effectuer 256 fois les opérations suivantes :
- R = a XOR b
- Count(R) → Compter le nombre de 1 résultant de l'opération bit à bit XOR
- Stocker le temps de l'opération

On répète l'opération 1000 fois pour en faire une moyenne du temps

En utilisant le C++ et la vectorisation des données, nous devrions avoir en théorie de meilleurs temps.

Les outils utilisés

Bien que nous ayons choisi le compilateur Intel C++ au début, nous avons en fait utilisé GCC pour la suite de l'implémentation de notre code car il s'agit d'un compilateur libre ayant une documentation abondante.

Les processeurs utilisés et nécessaires d'architecture x64 ont pu faire tourner les jeux d'instructions les plus récents tels que POPCNT et SSE4.1. Il existe une liste exhaustive des processeurs et architectures compatibles. La liste des processeurs compatible se trouve sur https://gcc.gnu.org

Nous n'avons pas utilisé d'IDE mais simplement un fichier texte plus GCC dû à la facilité d'ajout et de paramétrage des options de compilation.

Puis, la documentation officielle concernant les fonctions intrinsèques Intel :
https://software.intel.com/sites/landingpage/IntrinsicsGuide/

Enfin, la lecture de fond en comble du forum StackOverflow.

Explication du code

Voici la liste des étapes principales du traitement :

Définition de 2 vecteurs de 128 bits __m128i, xmm1 et xmm2

Alignement mémoire sur 16 octets de xmm1 et xmm2 → posix_memalign
Chargement des deux vecteurs __m128i avec 4 entiers de 32 bits chacun provenant du fichier d'entrée
→ _mm_setr_epi32

Début de la boucle permettant de faire la moyenne de temps des opérations
    Lancement du chronomètre

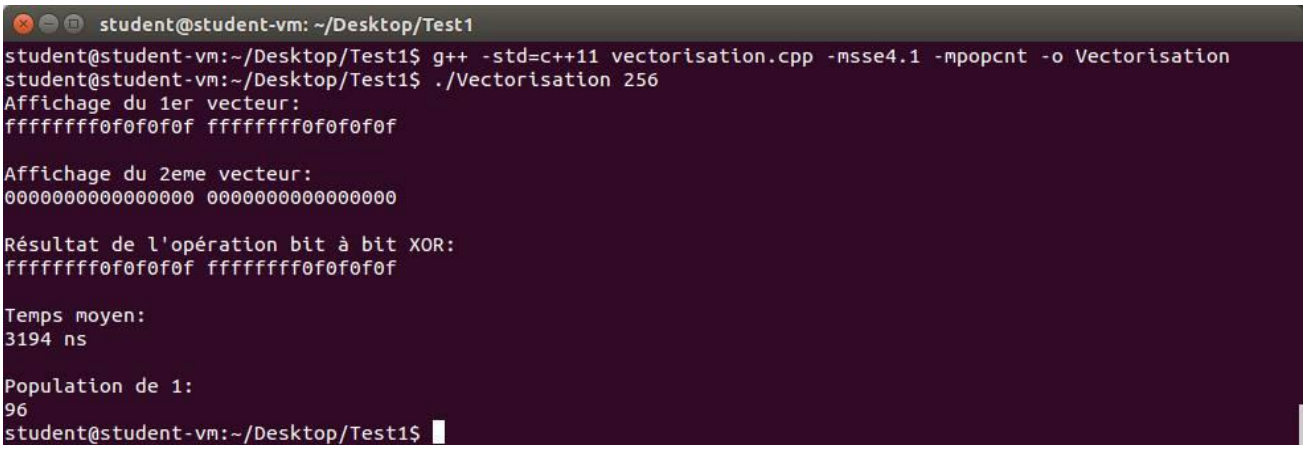
    Début de la boucle faisant les calculs d'une colonne du template de 256 bits de haut
    //permet de faire un calcul de 256 * 128 bits
        Opération bit à bit XOR entre xmm1 et xmm2 → _mm_xor_si128
        Extraction des 4 mots de 32 bits du vecteur 128 bits → _mm_extract_epi32
        Comptage de la population de 1 dans chacun des mots de 32 bits → _mm_popcnt_u32
        Addition des 4 résultats de comptage
    Fin de la boucle faisant les calculs d'une colonne du template de 256 bits de haut

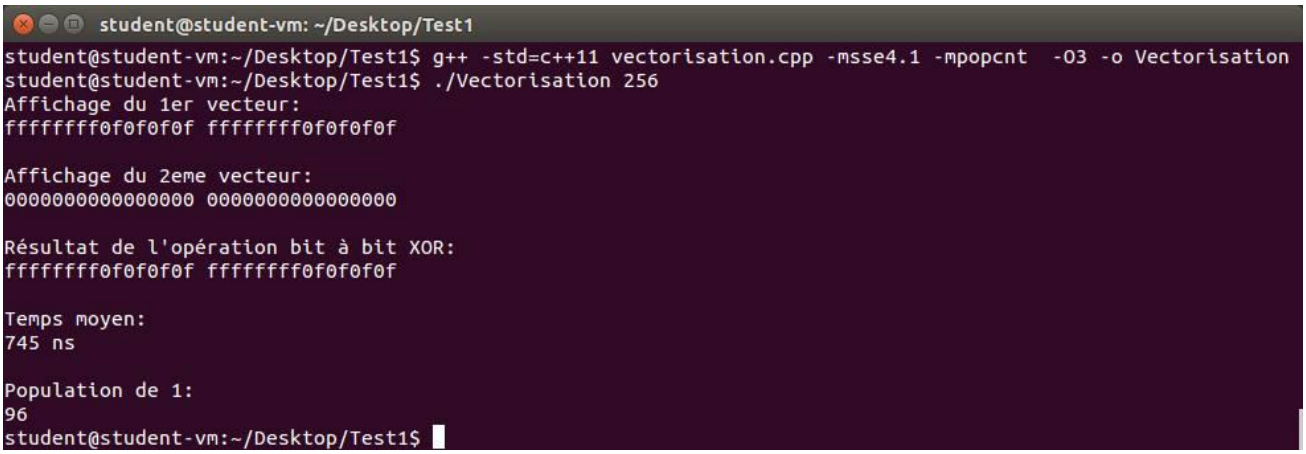
    Arrêt du chronomètre
    Stockage du temps dans un tableau
Fin de la boucle permettant de faire la moyenne de temps des opérations

Moyenne du temps

Résultat

Les impressions d'écran ci-dessous montrent les résultats de l'algorithme respectivement sans optimisation et avec optimisations (-O3)





On remarque dans cet exemple que le XOR opère bien, que le temps moyen est bien inférieur aux temps donnés par M.Delalandre. De plus la fonction comptage est bien fonctionnelle.

N.B. : Si l'on code sans vectorisation, c'est-à-dire avec de simple **xor** et avec le paramètre –O3, le compilateur génère du code assembleur plus succinct et utilisant la vectorisation. De plus, il est plus optimisé que notre code avec vectorisation manuelle sans optimisation ainsi qu'avec optimisation. Pour des performances optimales, il vaut mieux coder normalement en C++ en tenant compte des recommendations de GCC qui optimisera de lui-même.

Mise en commun du système

Cette partie devait s'effectuer lorsque toutes les parties étaient fonctionnelles, malheureusement, nous n'avons pas réussis à les faire toutes fonctionner, nous n'avons donc pas pu implémenter la mise en commun des trois parties différentes.

Le point bloquant en cette fin de projet est la lecture du fichier d'entrée.

Gestion de projet

Rédigé par le chef de projet : Guillaume Coué

Répartissions des tâches

Je me suis volontairement pas défini de tâche précise a effectué durant ce projet, ce qui m'a permis de garder une vision globale, et d'intervenir là ou ca bloquai. Je pense que c'était la meilleur chose à faire, et cela m'as permis d'intervenir et de mener à bien les parties Threads et Vectorisation.

La structure du projet se prêtait bien à une répartition part thème, ce qui a facilité son découpage. Initialement la répartition choisis était :

Fichier d'entrée : Alexandre

Threads : Johan

Vectorisation : Emilie + Thibault

Apres quelques semaines, je me suis rendu compte que la partie d'Alexandre n'avançais pas trop, et que celle de Thibault et Emilie était bien partie, j'ai donc proposé a Emilie d'aider Alexandre plutôt que de continuer avec Thibault, c'est sur cette répartition que nous avons poursuivi le projet.

Rapport avec Client

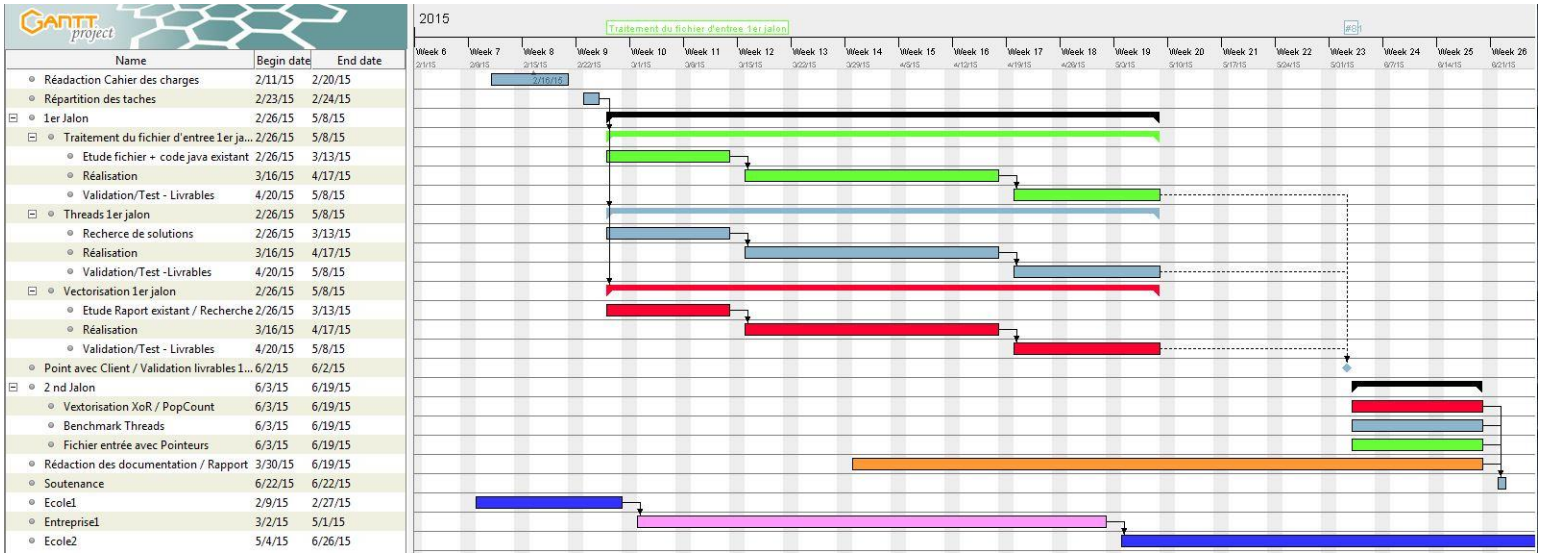
Pour ce qui est du rapport entre l'équipe et le client, seul le chef de projet avait un contact direct avec lui (Mr Delalandre) pour discuter du cahier des charges, de l'avancement, et des différents problèmes rencontrés. Ce choix de relation client/fournisseur a été choisi par Mr Delalandre, et correspond bien au monde professionnel, ou les équipes ont rarement un contact direct avec le client. Il a donc été mon rôle de synthétiser et restituer les spécifications à l'équipe.

Réunions régulière avec l'équipe

Nous avons mis en place des réunions toutes les 3 semaines avec toute l'équipe quand c'était possible. Ou nous discutions des points bloquant de chacun, et essayons d'apporte des solutions.

Planification

La date de rendu des éléments de fin du premier jalon n'as pas été respecté, nous avons été retardé par la fin des projets électroniques, et par le fait que 3/5eme de l'équipe préparait ses rattrapages de mathématique de 3eme année.



Name	Begin date	End date
Réadaction Cahier des charges	2/11/15	2/20/15
Répartition des taches	2/23/15	2/24/15
1er Jalon	2/26/15	5/8/15
Traitement du fichier d'entree 1er ja...	2/26/15	5/8/15
Etude fichier + code java existant	2/26/15	3/13/15
Réalisation	3/16/15	4/17/15
Validation/Test - Livrables	4/20/15	5/8/15
Threads 1er jalon	2/26/15	5/8/15
Recherce de solutions	2/26/15	3/13/15
Réalisation	3/16/15	4/17/15
Validation/Test -Livrables	4/20/15	5/8/15
Vectorisation 1er jalon	2/26/15	5/8/15
Etude Raport existant / Recherche	2/26/15	3/13/15
Réalisation	3/16/15	4/17/15
Validation/Test - Livrables	4/20/15	5/8/15
Point avec Client / Validation livrables 1...	6/2/15	6/2/15
2 nd Jalon	6/3/15	6/19/15
Vextorisation XoR / PopCount	6/3/15	6/19/15
Benchmark Threads	6/3/15	6/19/15
Fichier entrée avec Pointeurs	6/3/15	6/19/15
Rédaction des documentation / Rapport	3/30/15	6/19/15
Soutenance	6/22/15	6/22/15
Ecole1	2/9/15	2/27/15
Entreprise1	3/2/15	5/1/15
Ecole2	5/4/15	6/26/15

(Une version PDF du diagramme est disponible dans le package)

GIT

Pour travailler ensemble, et pour avoir une vue globale de l'avancement de chaque partie, nous avons mis en place un GIT ou chaque groupe tenait ses travaux à jour, que ce soient les codes ou la documentation liée.

Conclusion

Ce projet que nous avons effectué et le premier projet que nous effectuons dans un groupe de cinq personnes. Afin de nous organiser, nous avons élu le chef de projet qui assurera la tutelle du groupe tout au long du projet collectif.

Le projet que nous avons choisi concerne l'étude d'une image de type manga. Le but final est de déterminer la partie la plus importante de cette image afin de pouvoir la comparer aux autres images présente sur internet afin de détecter les scans de manga.

La première partie de ce projet nous a permis de découvrir les trois points principaux du projet dans un cadre « expérimental ». Nous avons pu faire des étude concernant la vectorisation en nous montrant que c'est un élément qui fait gagner énormément de temps pour des calculs compliqués. Nous avons aussi pu aborder le système de parallélisassions de processus qui permettent aussi un gain de temps lors des calculs. Nous avons ainsi pu déterminer la configuration optimale du système. Enfin, nous avons pu travailler sur la mise en place de la lecture et l'instanciation en mémoire du fichier d'entrée de notre système.

Après avoir présenté cette partie à l'encadrant, nous avons eu de nouvelles consignes. Nous avons donc continué à travailler sur les différents éléments afin d'avoir un fonctionnement correspondant au besoin de l'encadrant. Nous devons ensuite mettre en place le regroupement de tout ce que nous avons fait jusqu'à présent afin d'obtenir un système de calcul optimisé.

Le travail sur les différentes parties nous a pris plus de temps qu'il était prévu et nous regrettons de ne pas avoir eu le temps de mettre en œuvre le regroupement des trois parties respectives.

Annexe 1 : Bilan personnel Guillaume COUE

Rôle : Chef de projet

Nombre d'heures travaillées : ~50

Bilan personnel :

Durant ce projet, et en tant que chef de projet, je pense avoir négligé la partie concernant le fichier d'entrée, qui bien qu'assigné a deux personnes n'as pas abouti. Je pense que j'ai plus travaillé sur les parties qui me semblaient attrayantes à savoir les threads et la vectorisation. Cette gestion par affinité avec les taches est je pense une mauvaise façon de gérer un projet global, et ne sera pas à refaire dans le cadre d'un autre projet.

Ce projet m'a montré également qu'il est plus difficile de conduire un projet de ce type, qu'un projet en entreprise, car il y a beaucoup moins d'heures attribuées, et les personnes ne travaillent pas tout le temps ensembles. Un autre aspect difficile dans la conduite de ce projet a été de restituer de façon compréhensible les attentes du client à l'équipe, quand j'avais moi-même quelques difficultés à les comprendre parfois.

D'un point de vue technique, je suis très satisfait de ce projet, surtout de la partie vectorisation, qui m'a fait découvrir une façon de programmer très proche de la machine, a mis chemin entre le C++ et l'assembleur. Et avec des performances remarquables.

Pour conclure, je suis un peu déçu que ce projet n'ait pas aboutit, mais je pense que c'est due à une négligence de ma part sur le point d'entrée du projet, qui me paraissait a première vue triviale.

Annexe 2 : Bilan personnel Thibault ARTUS

Rôle : Vectorisation

Nombre d'heures travaillées : ~35 heures

Bilan personnel :

Ce projet ne fut pas l'un de mes premiers choix favoris lors de la sélection car il n'y a pas de partie « électroniques », ce qui m'intéresse le plus. Cependant, le fait de pouvoir manipuler des instructions de bas niveau par rapport au processeur a redonné de l'intérêt.

Mon rôle fut de m'occuper principalement de la partie vectorisation afin d'optimiser le temps de calcul. J'ai dû, entres autres, manipuler des instructions intrinsèques Intel, ce que je ne connaissais pas. Le fait de travailler tout seul sur une partie nécessitant beaucoup de concentration pour éplucher la documentation fut intéressant. Lorsqu'un problème ou une incompréhension arrivait, Guillaume m'aidait pour que l'on reprenne point par point le code et la pu disséquer la documentation, les forums et le code assembleur.

Ce projet fut très intéressant même si aux premiers abords cela m'a un peu rebuté. De plus, j'ai acquis des connaissances dans ce domaine qui m'était jusqu'à maintenant totalement inconnus, ce qui est bien, non ?

Annexe 3 : Bilan personnel Alexandre BILLAY

Rôle : Fonction de lecture de .map, et d'extraction des données de celui-ci vers un format exploitable dans le projet.

Nombre d'heures travaillées : ~ 40

Bilan personnel :

Projet intéressent, approfondissement des connaissances dans la gestion de fichier en C++.

Les aspects de gestion de projet dans un collectif relativement important sont également un point appréciable de cette expérience.

Déçu de ne pas avoir réussi à l'emmener à terme.

Annexe 4 : Bilan personne Johan KARPINSKY

Rôle :

Mettre en œuvre un logiciel de test permettant de déterminer le nombre optimum de threads à utiliser pour réaliser un calcul.

Nombre d'heures travaillées :

- 4h : Recherche de documentation sur le fonctionnement du multithreading pour déterminer les facteurs limitants.
- 3h : prise en main de parallèle studio XE 2015 et recherche de librairie.
- 1h : Mise en place d'un test pour observer l'impact des threads sur la charge CPU.
- 3h : création d'une architecture de base permettant de créer un nombre de threads donné pour ensuite leur affecter un traitement.
- 3h : Essais d'optimisation du code pour améliorer le nombre de threads optimal.
- 3h : Recherche sur le fonctionnement des threads pour essayer d'améliorer le fonctionnement existant.
- 6h : Mise en œuvre du programme et modifications pour obtenir des résultats parlants.
- 6h : Rédaction des rapports.

Bilan personnel :

Point positif :

- Ce projet m'a permis d'améliorer la compréhension du fonctionnement du multithreading.
- Le fait de travailler en équipe avec un chef de projet était un aspect nouveau et enrichissant. De plus le fait que le chef de projet s'occupe « uniquement » de la gestion de projet est appréciable pour la répartition des tâches.
- Une partie intéressante car basé sur la recherche avec un assez grand degré de libertés.

Point négatif :

- Une impression d'être peu efficace. En effet, le travail qui m'a été confié était avant tout un travail de recherche. Avec le recul ainsi qu'avec les résultats finaux, je me rends compte que j'ai passé beaucoup de temps à chercher dans de mauvaises directions ce qui est un petit peu frustrant.

Annexe 5 : Bilan personnel Emilie LEMEE

Rôle : Documentation, lecture du fichier d'entrée

Nombre d'heures travaillées : ~25 heures

Bilan personnel :

Le projet de sélection de template était un projet relativement abstrait pour moi. Bien qu'intéressant, puisqu'il met en œuvre une partie de la programmation que nous avons très peu vue en cours (programmation multi-thread, étude des instructions vectorisées) l'objectif de mettre en œuvre un système dont les performances sont optimisées est très motivant pour moi.

Mon rôle étant principalement la création d'un programme afin de lire un fichier d'entrée de format « .map » et de le stocker en mémoire m'a permis de travailler avec un fichier que je ne connaissais pas. J'ai aussi pu découvrir la bibliothèque C++ « ifstream » qui permet de travailler avec des fichiers. Le fait de travailler à deux sur un même programme a été assez compliqué. En effet, bien qu'ayant mis en place un logiciel de gestion de version (GitHub) pour pouvoir travailler ensemble sur un même programme, la synchronisation a été assez compliquée. C'était cependant une expérience qui m'a permis de m'améliorer et d'adopter une convention de nommage qui est la même que mon binôme. Nous avons aussi pu vérifier l'importance des commentaires dans un tel code.

Ce projet a donc été très enrichissant, tant sur le plan technique que sur le plan gestion de projet. En effet, en plus d'avoir acquis de nouvelles connaissances en programmation, j'ai pu découvrir la programmation en groupe, ainsi que le travail dans un groupe sous la tutelle d'un chef de projet.