Concours Acceler’8 – Intel

Problème 1 : Smallbrain Numbers

I – Approche mathématique

1. Représentation d’un nombre

On considère un nombre comme un n-upplets d’éléments pris entre 0 et 9 dont la relation est : . De cette manière, nous pouvons le représenter comme un tableau à une dimension de taille n.

1. Relation des SmallBrain Numbers

Un nombre est un nombre Smallbrain s’il obéit à la relation :

1. Existences de collisions

Considérant un nombre comme un tableau d’éléments, on remarque que lorsque l’on permute ces éléments, on fabrique un nombre différent dont la somme des chiffres élevés à la puissance (à présent appelé *Nombre Testé*) est égale à celle du nombre d’origine. Par conséquent, nous pouvons diminuer considérablement le nombre de combinaisons à tester sur la totalité d’un range.

1. Création des combinaisons uniques

Afin d’obtenir une liste de combinaisons uniques, nous permettant de couvrir l’ensemble des nombres testés, nous incrémentons nos combinaisons tel que :

1. Comment détecter un Smallbrain Number

Lors de la décomposition du *Nombre Testé* en tableau, nous réalisation l’intersection de celui-ci avec le tableau d’une des collisions associée. Si cette intersection est complète (tous les éléments sont dans présents dans chacun des tableaux) alors notre *Nombre Testé* est un Smallbrain Number.

II – Approche algorithmique

1. Mises en cache des puissances de 0 à 9

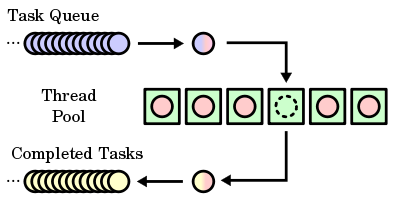
Afin de réduire les calculs de puissance, impliquant une lourde charge au processeur, nous fabriquons lors du lancement du programme un cache des puissances de 0 à 9 pour tous les exposants correspondant à notre besoin.

1. Mises en cache des combinaisons uniques

On fabrique l’ensemble des combinaisons unique au lancement du programme afin de juste calculer les sommes de puissance sur chacune de ces combinaisons pour tous les ordres supérieurs à l’ordre le plus faible de la combinaison.

III- Mise en parallèle du code

Nous étions partis sur l’idée d’un thread pool, c'est-à-dire avoir un nombre de thread fixe et les alimenter à leur demande. Cette solution était idéale dans le cas où nous aurions eu des tâches dont nous ignorions leur temps d’exécution. Par la suite nous avons été en mesure de fabrique un cache de l’ensemble des combinaisons unique sur lesquelles nous connaissions le temps approximatif de traitement. Par conséquent, nous sommes partis sur un algorithme distribué utilisant la librairie openmp.



Thread Pool