Collections Concurrentes

Lab 5 : **Single instruction, Multiple Data (SIMD)**

Calcul parallèle, Vectorisation, Loop decomposition

SORAN Altan - ESIPE INFO 2020

https://github.com/asoran/Structures-et-Algo-Concurentes

ForkJoin & Vector

On a vu avant que pour accélérer le traitement d'une tâche on pouvait la paralléliser via l'API Fork/Join.

Avec la proposition d’API Vector, on peut faire des calculs vectoriel, c'est à dire faire plusieurs opérations en 1 seule !

Ce style de parallélisme est appelé Single Instruction Multiple Data (SIMD).

La différence c'est que fork/join se base sur le multi-coeur des ordinateurs, alors que les Vector ne sont qu'une accélération de vitesse de calcul, on peut donc "potentiellement" utiliser ces 2 techniques en même temps.

Vector

`jdk.incubator.vector.Vector` est une interface qui représente un *Vector*

Un *Vector* est un array d'élément (qui peuvent être différents) du même type sur lequel on peut effectuer plusieurs opérations.

Il existe une implémentation pour chaque type primitif de java (`IntVector` pour les Integer par exemple)
La capacité en bits du vecteur dépend de comment on l'a déclaré avec un objet `VectorSpecies`. On peut par exemple utiliser `IntVector.`*`SPECIES_256`* pour spécifier un vecteur qui peut prendre 8 int (256/ (taille int = 32)).
On peut utiliser *`SPECIES_PREFERRED`* pour laisser le choix a la machine

Important: Un Vector doit contenir EXACTEMENT sa capacité, ni plus ni moins !

Création d'un Vector

On peut créer un Vector à partir d'un array du type de Vector, d'un ByteArray ou bien un ByteBuffer.

Exemple: IntVector depuis un int[]:

```
var vector = IntVector.fromArray(species, array, offset);
```

Ceci va créer un IntVector avec les valeurs du tableau `'array'` allant de l'index `'offset'` à l'index `'offset'` + capacité (indiqué via `'species'`)

On peut aussi utiliser IntVector.*zero*(species) pour créer un IntVector de taille spécifié dans `species` qui aura tous ses éléments égales a 0.

Une fois le traitement terminé, on peut passer d'un Vector a un array avec la méthode *`toArray()`*.

Exemple avec parcours d'un tableau

On a vu qu'un Vector pouvait stocker X valeurs dedans, mais du coup comment ça marche ? Comment on les utilise avec un array qui a Y valeurs ?

On va lire les valeurs de array X par X, traiter ce vecteur dans une main loop, puis traiter les dernières valeurs non traitées de l'array (car Y n'est probablement pas un multiple de X) dans une post-loop:

```
var i = 0; // Je suis un message caché huehuehue
var limit = array.length - (array.length % species.length());
for (; i < limit; i += species.length()) { // main loop
   var iv = IntVector.fromArray(species, array, i); // "load" the values
}
for (; i < array.length; i++) { // post loop
   // treat array[i];
} // HUEHUEHUEHUEHUEHUEHUEHUE
```

Opérations sur Vector & lanewise

On peut effectuer plusieurs opérations sur les vecteurs. Un famille d'opération qu'on peut effectuer sont appelés le opérations *lanewise*, ce sont des opérations que l'on va faire sur chaque 'lane' (élément, int dans le cas d'un IntVector) du Vector. Toutes les opérations que l'on peut effectuer sont listées dans la classe `VectorOperators`.

Il existe 3 types d'opérations: *Unary*, *Binary* et *Ternary*. Comme leur noms l'indique, ce sont le nombre d'opérandes de l'opérations.

Si l'on fait une opération *Unary* comme *NOT*, elle se fera sur chaque lane du Vector. Alors que si on fait une opération Binary come SUM, elle se fera entre chaque lane et un élément passer en paramètre ou avec une lane d'un autre Vector

Exemple: addition lanewise

Voici un exemple d'une addition lanewise:

On a un array avec 8 valeurs: array = [0, 1, 2, 3, 4, 5, 6, 7]
On crée 2 IntVector de capacité 128 bits, iv et iv2, qui auront respectivement les 4 premières et 4 dernières valeurs de cette array.

Si l'on exécute `iv.add(iv2)` (qui revient à faire `iv.lanewise(ADD, iv2)`):

iv1	0	1	2	3
iv2	4	5	6	7
iv.add(iv2)	4	6	8	10

Reducing operation

Maintenant que l'on a un IntVector qui contient les sommes lanewise du tableau, on voudrait bien par exemple faire la somme de chaque lane et récupérer cette valeur.

Cela s'appelle faire une réduction! On peut effectuer des opérations **associative** successive sur chaque lane.
Si on veut récupérer la somme de chaque lane on peut le faire comme ceci:

```
var sum = iv.reduceLanes(VectorOperators.ADD);
```

Ici `iv` contient (4, 6, 8, 10), ce qui va se passer c'est que l'on effectuer l'opération *ADD* entre les 2 premiere valeurs, puis successivement avec le résultat de cette opération et la prochaine lane. A la fin on devrait obtenir (((4 + 6) + 8) + 10) = 28