Dans les exercices qui suivent, nous utiliserons le fichier `bsmlAnalyser.ml` qui contient un module BsmlAnalyser. Ce module contient des versions instrumentées des primitives BSML ainsi qu'une fonction bsp_test : ('a→ 'b)→ 'a→ 'b qui prend en paramètre une fonction et une valeur et affiche le coût BSP de l'application de la fonction à la valeur.

Dans la suite vous utiliserez les fonctions BSML déjà vues ou implémentées ainsi que la fonction **put**.

1 diffusion, réduction et préfixe parallèle

Exercice 1. Diffusion.
Définir la fonction bcast: int → 'a **par** → 'a **par** telle que
bcast i $<x_0 , x_1 , \ldots , x_{(p-1)}> = <x_i , x_i , \ldots , x_i >$.

a. Faites une première version à l'aide de la primitive de communication **proj**. Estimez le temps nécessaire à l'évaluation de bcast i x (par rapport à la taille des éléments de x).

b. Faites une seconde version à l'aide de la primitive de communication **put**. Estimez le temps nécessaire à l'évaluation de bcast i x.

c. Y a-t-il des cas où la version utilisant **put** est plus efficace ?

Rappel : *L'élément neutre* e *d'un opérateur* op *est tel que* op e a = op a e = a.

Exercice 2. Réduction.
Définir fold: ('a → 'a → 'a) → 'a → 'a **par** → 'a telle que fold op e $<x_0 , x_1 , \ldots , x_{(p-1)}>$ soit la (op)-réduction des x_i avec l'élément neutre e.

a. Faites une première version à l'aide de la primitive de communication **proj**. Estimez le temps nécessaire à l'évaluation de fold (op) x.

b. Faites une seconde version à l'aide de la primitive de communication **put**. Estimez le temps nécessaire à l'évaluation de fold (op) x.

c. Y a-t-il des cas où la version utilisant **put** est plus efficace ?

Exercice 3. Préfixe.
Définir scan: ('a → 'a → 'a) → 'a → 'a **par** → 'a **par** telle que, si e est le neutre de op alors scan (op) e $<x_0 , x_1 , \ldots , x_{(p-1)}> = <y_0 , y_1 , \ldots , y_{(p-1)}>$ avec $y_{i+1} = op\ y_i\ x_{i+1}$ et $y_0 = x_0$.

Estimez le temps nécessaire à l'évaluation de scan (op) x.

Exercice 4. Diffusion en deux phases.
Définir bcast2phases: 'a list **par** → 'a list **par** telle que
bcast2phases $<bloc_0 , \ldots , bloc_{(p-1)}> = <bloc_0 , \ldots , bloc_0>$ en utilisant l'algorithme BSP à deux phases (le plus efficace quand lat taille de bloc_0 tend vers l'infini).

Estimez et mesurez le temps nécessaire pour évaluer (bcast2phases blocs).

Exercice 5. reduction par blocs.
Définir fold_par_blocs: ('a → 'a → 'a) → 'a → 'a list **par** → 'a telle que
fold_par_blocs op z $<bloc_0 , \ldots , bloc_{(p-1)}>$ soit la op-réduction des $bloc_i$. Ici la valeur z:'a est l'élément neutre de op.

Estimez et mesurer le temps nécessaire pour évaluer fold op z blocs.

Exercice 6. Décalage.
Définir shift: int → 'a list **par** → 'a list **par** telle que
shift 1 < [0;1], [2;3;4];[5;6];[7]> = < [7;0], [1;2;3],[4;5],[],[6]>
shift 2 < [0;1], [2;3;4];[5;6];[7]> = < [6;7], [0;1;2],[3;4],[],[5]>
shift 3 < [0;1], [2;3;4];[5;6];[7]> = < [5;6], [7;0;1],[2;3],[],[4]>
...