mkpar

Rappel : *La fonction* **mkpar** : (int → 'a) → 'a **par** *appliquée à une fonction* f *construit le vecteur* ≪ f 0 ; f 1 ; . . . ; f (p−1)≫ . f pid *est calculé sur le processeur* pid

Exercice 1. Un vecteur simple : this.
Faites un fonction this : int **par** qui contient à chaque processeur l'identifiant du processeur (que l'on nommera toujours pid dans la suit de ce TD).

$$\text{this} = \ll 0 \ ; \ 1 \ ; \ \ldots \ ; \ p{-}1 \gg$$

Exercice 2. Une fonction simple : replicate.
Faites une fonction replicate : 'a → 'a **par** qui prend une valeur et construit un vecteur parallèle ayant cette valeur à chaque processeur.

$$\text{replicate a} = \ll \text{a} \ ; \ \text{a} \ ; \ \ldots \ ; \ \text{a} \gg$$

Exercice 3. Construire des vecteurs parallèles.
À l'aide de la fonction **mkpar** construisez les vecteurs parallèles suivant :

a. *le vecteur* trois , qui contient le même entier 3 partout ;

b. *le vecteur* revs , qui contient les entiers 0,...,p-1 mais en ordre inverse ≪ p−1, p−2 . . . ≫ ;

c. *le vecteur* howmany , qui contient la chaîne de caractères correspondant au nombre de processeur : "8" partout si la machine parallèle a 8 processeurs ;

d. *le vecteur* strlens , qui contient en pid la longueur (en nombre de caractères) de la chaîne de caractères (string_of_int pid) ;

e. *le vecteur* emptylists , qui contient partout la liste vide ;

f. *le vecteur* from_0_to_pid , qui contient en pid la liste [0;. . . ;pid].

apply

Rappel : *La fonction* **apply** : ('a → 'b) **par** → 'a **par** → 'b **par** *appliquée à un vecteur de fonctions* $\ll f_0 \ ; \ f_1 \ ; \ \ldots \ ; \ f_{(p-1)} \gg$ *et à un vecteur de valeurs* $\ll v_0 \ ; \ v_1 \ ; \ \ldots \ ; \ v_{(p-1)} \gg$ *construit le vecteur* $\ll f_0 \ v_0 \ ; \ f_1 \ v_1 \ ; \ \ldots \ ; \ f_{(p-1)} \ v_{(p-1)} \gg$.

Exercice 4. Fonction simple : parfun : ('a → 'b) → 'a **par** → 'b **par.**
À l'aide des fonctions **mkpar** et **apply**, faites :

a. une fonction qui prend en paramètre une fonction f et un vecteur v et qui applique f en tout points du vecteur v :

$$\text{parfun f} \ll v_0 \ ; \ v_1 \ ; \ \ldots \ ; \ v_{(p-1)} \gg = \ll \text{f } v_0 \ ; \ \text{f } v_1 \ ; \ \ldots \ ; \ \text{f } v_{(p-1)} \gg$$

b. une fonction parfun2 : ('a → 'b → 'c) → 'a **par** → 'b **par** → 'c **par** pour appliquer à deux vecteurs parallèles une fonction à deux paramètres.

$$\begin{aligned}\text{parfun2 f} \ll v_0 \ ; \ v_1 \ ; \ \ldots \ ; \ v_{(p-1)} \gg \ll x_0 \ ; \ x_1 \ ; \ \ldots \ ; \ x_{(p-1)} \gg \\ = \ll \text{f } v_0 \ x_0 \ ; \ \text{f } v_1 \ x_1 \ ; \ \ldots \ ; \ \text{f } v_{(p-1)} \ x_{(p-1)} \gg\end{aligned}$$

Exercice 5. Fonctions de type 'a **par** → 'b **par.**
À l'aide des fonctions **mkpar**, **apply**, replicate et parfun faites une fonction :

a. qui ajoute pid à la valeur entière se trouvant au processeur pid ;

b. qui soustrait la valeur trouvée en pid du nombre total de processeurs ;

c. qui concatène " processeurs" à la valeur trouvée en pid ;

d. qui convertit en string l'entier trouvé en pid et lui concatène la chaîne de caractères "␣chars␣in␣"^(string_of_int pid) ;

e. qui applique partout la fonction List.length ;

f. qui applique partout la fonction List.rev.

Applications

Exercice 6. Réutilisation des fonctions précédentes.
Pour n=a,b,c,d,e, appliquer la fonction définie à la question 5(n) au vecteur défini en 3(n).

Exercice 7. Génération d'un vecteur aléatoire.
Construire un vecteur parallèle de type int list Par dont la valeur au processeur pid est une liste de longueur aléatoire (au plus **bsp_p** processeurs) contenant des entiers aléatoires (de valeurs entre 0 et **bsp_p**).

Exercice 8. Tri locaux.
Définissez une fonction de tri local sort_par qui trie chacune des listes d'un vecteur de liste à l'aide de la fonction List.sort. Appliquez la au vecteur créé à l'exercice précédent.

Exercice 9. Répartition d'une liste.
Faites un fonction distribute : 'a list → 'a list **par** qui prend une liste séquentielle et construit un vecteur parallèle où chaque processeur contient un morceau de la liste. La liste devra être répartie de façon équilibrée (même longueur sur chaque processeur à un élément près).

Exercice 10. Hello world.
Faites un fichier `hello.ml` contenant la fonction hello_world : unit → unit Par. hello_world utilise print_endline et **mkpar** pour que chaque processeur affiche Hello from processor pid. Le fichier se terminera par **let** _ = hello_world () pour exécuter la fonction.

Compilez le fichier à l'aide de bsmlopt.mpi et lancez plusieurs exécutions. Observez l'ordre d'affichage des numéros de processeurs.