1 Rappels OCaml

Langage functionnel d'ordre supérieur

```
(** definition simple de type int → int → int *)
let my_addition a b = a + b

(* definition recursive, pattern matching, fonction en parametre *)
(* type : ( α → β ) → α list → β list *)
let rec my_map f l =
  match l with
  | [] → []
  | h::t → (f h)::(my_map f t)

(* fonction anonyme en parametre *)
(* type : int → int list → int list *)
let map_addition a l = my_map (fun b → my_addition a b) l
```

2 Bulk Synchronous Parallel ML

Vecteur Parallèle

- un type de donnée abstrait polymorphe : α **par**
- une taille fixe p : chaque processeur a une valeur de type α
- imbrication de vecteur interdite

Accès aux paramètres BSP

bsp_p: int
bsp_g: float
bsp_l: float

Création de vecteur parallèles

— **mkpar** : (int → α) → α **par**

(**mkpar** f)

$(f\ 0)$	$(f\ 1)$	$\cdots$	$(f\ (p-1))$

Application parallèle point à point

— **apply** : (α → β) **par** → α **par** → β **par**

(**apply**

f_0	f_1	$\cdots$	f_{p-1}

v_0	v_1	$\cdots$	v_{p-1}

)

=

$(f_0\ v_0)$	$(f_1\ v_1)$	$\cdots$	$(f_{p-1}\ v_{p-1})$

Syntaxe alternative

— ≪ a ≫

a	a	$\cdots$	a

— ≪ $this$ ≫

0	1	$\cdots$	$p-1$

Equivalence avec les primitives précédentes

let mkpar f = ≪ f $this$ ≫ **let apply** g v =≪ g v ≫

Du local au global : proj

— **proj**: α **par** → (int → α)

$$\left(\textbf{proj}\ \boxed{v_0 \mid v_1 \mid \cdots \mid v_{p-1}}\right) = \begin{array}{lll} \textbf{function} & 0 & \rightarrow\ v_0 \\ | & 1 & \rightarrow\ v_1 \\ & \vdots & \\ | & p-1 & \rightarrow\ v_{p-1} \end{array}$$

Communications : put

— **put**: (int → α) **par** → (int → α) **par**

$$\left(\textbf{put}\ \boxed{f_0 \mid f_1 \mid \cdots \mid f_{p-1}}\right) = \boxed{g_0 \mid g_1 \mid \cdots \mid g_{p-1}}$$

$g_i\ j = f_j\ i$

0	1	2	3
$(f_0\ 0) = A$	$(f_1\ 0) = B$	$(f_2\ 0) = C$	$(f_3\ 0) = D$
$(f_0\ 1)$	$(f_1\ 1)$	$(f_2\ 1)$	$(f_3\ 1)$
$(f_0\ 2)$	$(f_1\ 2)$	$(f_2\ 2)$	$(f_3\ 2)$
$(f_0\ 3)$	$(f_1\ 3)$	$(f_2\ 3)$	$(f_3\ 3)$

⇒

0	1	2	3
A	$(f_0\ 1)$	$(f_0\ 2)$	$(f_0\ 3)$
B	$(f_1\ 1)$	$(f_1\ 2)$	$(f_1\ 3)$
C	$(f_2\ 1)$	$(f_2\ 2)$	$(f_2\ 3)$
D	$(f_3\ 1)$	$(f_3\ 2)$	$(f_3\ 3)$

3 BSML en pratique

Implantations actuellement disponibles

— Bibliothèque de programmation OCaml
— implantation Modulaire : MPI, TCP/IP, BSPlib, ...
— boucle interactive séquentielle

Parallèle avec mpi

— (sous certain OS) $ module load mpi
— compilation
$ bsmlopt.mpi hello.ml −o hello.mpi
— execution parallèle
$ mpirun −np 16 ./hello.mpi

Boucle interactive (toplevel)

— S'assurer de l'existence d'un fichier `.bsmlrc` dans le répertoire personnel contenant nombre de nœuds, pref réseau, latence et perf processeur de la machine parallèle simulée :
$ **echo** '8,50.,5000.,400000.' > ∼/.bsmlrc
— lancer la boucle interactive :
$ bsml
— Dans le boucle interactive, ouvrez le module contenant les primitives BSML
open Bsml;;
— Pour charger un fichier source `mon_code.ml` :
#use "mon_code.ml";;