Tutoriel : utilisation de MPI en OCaml

Camille Coti

14 octobre 2016

Il existe des bindings MPI non-officiels pour OCaml, distribués sur la forge de OCaml [1]. J'ai corrigé un petit bug, et je distribue la version corrigée sur GitHug [2].

1 Inclusion et compilation

On doit utiliser le module appelé `Mpi` :

```
1 open Mpi
```

Pour compiler, on spécifie qu'il faut linker avec le module :

```
1 ocamlc -I +ocamlmpi -o progparallele mpi.cma progparallele.ml
```

2 Appel de fonctions MPI

2.1 Communicateurs MPI

On a le communicateur `Mpi.comm_world`, sur lequel on peut récupérer son rang et le nombre de processus :

```
1 let size = Mpi.comm_size Mpi.comm_world in
2 let rank = Mpi.comm_rank Mpi.comm_world in
```

2.2 Communications point-à-point

On n'a pas à spécifier le type de données envoyées. On envoie des données avec Mpi.send, suivant la syntaxe :

```
1 send variable rank tag communicateur
```

Par exemple :

```
1 Mpi.send i src data_tag comm_world ;
```

Pour recevoir, tout dépend si l'on utilise une wildcard ou non. Si l'on connait tous les arguments (et en particulier la source et le tag du message), on utilise la fonction Mpi.receive qui retourne les données reçues, suivant la syntaxe :

```
1 res = receive source tag communicateur
```

Par exemple :

```
1 res = Mpi.receive src result_tag Mpi.comm_world;
```

1. https://forge.ocamlcore.org/projects/ocamlmpi/
2. https://github.com/coti/ocamlmpi

Les wildcards sont `Mpi.any_source` pour recevoir depuis n'importe quel expéditeur et `Mpi.any_tag` pour recevoir avec n'importe quel tag. On utilise la fonction `Mpi.receive_status` qui retourne un triplet contenant les données reçues, l'expéditeur et le tag :

```
res, source, tag = receive_status source tag communicateur
```

Par exemple :

```
(res, src, tag) = Mpi.receive_status Mpi.any_source Mpi.any_tag Mpi.comm_world;
```

2.3 Communications collectives

Les communications collectives s'utilisent dans le même esprit : on ne spécifie pas le type de données envoyées ni la taille, les données obtenues sont retournées par la fonction.

Par exemple, pour la barrière, on spécifie le communicateur :

```
Mpi.barrier Mpi.comm_world;
```

Pour le broadcast, on donne le buffer à envoyer par la racine, la racine et le communicateur ; le buffer résultat est retourné par la fonction appelée.

```
token := Mpi.broadcast !token 0 Mpi.comm_world;
```

Ci-dessous un petit exemple où le processus de rang 0 récupère son pid et le diffuse à tous les autres processus :

```
open Mpi
open Unix

let main () =

  let rank = Mpi.comm_rank Mpi.comm_world in
  let token = ref 0 in

  if rank = 0 then
    token := Unix.getpid()
  ;

    token := Mpi.broadcast !token 0 Mpi.comm_world;

  print_string "[" ; print_int rank ; print_string "] token is " ;
  print_int !token ; print_newline();

    Mpi.barrier Mpi.comm_world
;;

if !Sys.interactive then () else main ();;
```

Le code ci-dessus se compile avec :

```
ocamlc -I +ocamlmpi -o exemplebcast mpi.cma unix.cma exemplebcast.ml
```

Et s'exécute avec :

```
mpiexec -n 4 ./exemplebcast
```

3 Exemple : anneau à jeton

Le code suivant se compile avec la commande :

```
ocamlc -I +ocamlmpi -o tokenring mpi.cma tokenring.ml
```

Et s'exécute de la façon suivante :

```
mpiexec -n 4 ./tokenring
```

Le code lui-même est composé d'une seule fonction `main()`. On commence par déterminer son rang et le nombre de processus impliqués. Les processus font circuler un jeton qui est incrémenté à chaque passage dans un processus. Le processus 0 initie la circulation, et l'arrête une fois qu'il a réalisé 20 tours.

```
open Mpi

let main () =

  let size = Mpi.comm_size Mpi.comm_world in
  let rank = Mpi.comm_rank Mpi.comm_world in

  let token = ref 0 in
  let cnt = ref 0 in

  let my_tag = 1664 in

  if rank = 0 then
    Mpi.send !token 1 my_tag Mpi.comm_world
  ;

    let left = (( rank + size - 1 ) mod size ) in
    let right = (( rank + 1 ) mod size ) in

    while !cnt < 20 do

      token := receive left my_tag Mpi.comm_world;

      print_string "[" ; print_int rank ; print_string "] received token " ;
      print_int !token ; print_newline();
      token := !token + 1;

      if not ( ( rank = 0 ) && ( !cnt = 20 ) ) then
        begin
          Mpi.send !token right my_tag Mpi.comm_world;
          cnt := !cnt + 1;
          print_string "[" ; print_int rank ; print_string "] cnt is " ;
          print_int !cnt ; print_newline();
        end
      ;

    done;

    Mpi.barrier comm_world
;;

if !Sys.interactive then () else main ();;
```