

Bulk Synchronous Parallel ML Modular Implementation and Performance Prediction

Frédéric Louergue

Laboratory of Algorithms, Complexity and Logic
Université Paris Val de Marne – CNRS
Créteil, France

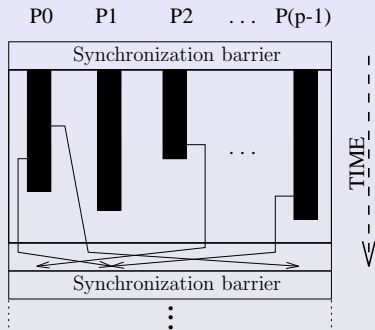
Laboratoire d'Informatique Fondamentale d'Orléans
Université d'Orléans

28 juin 2005

Introduction

- The design of a parallel language is a trade-off between :
 - expressivity
 - simple semantics & performance prediction
- Bulk Synchronous Parallel ML :
 - describes Bulk Synchronous Parallel algorithms
(explicit processes & communications)
 - deadlock free & deterministic
 - pure functional semantics
(based on a confluent extension of the λ -calculus)

Bulk Synchronous Parallelism



$$T(s) = \max_{0 \leq i < p} w_i + h \times g + L$$

Overview

- Functional language (ML family) + BSP operations
- BSML prog. = usual ML prog. + operations on parallel vectors
- Parallel vector : α `par` (size p)
- Access to BSP parameters :

`bsp_p`: unit \rightarrow int

`bsp_g`: unit \rightarrow float

`bsp_l`: unit \rightarrow float

Creation of parallel vectors

$\text{mkpar} : (\text{int} \rightarrow \alpha) \rightarrow \alpha \text{ par}$

$(\text{mkpar } f)$

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

BSP cost : $\max_{0 \leq i < p} w_i$

Point-wise parallel application

$\text{apply} : (\alpha \rightarrow \beta) \text{ par } \rightarrow \alpha \text{ par } \rightarrow \beta \text{ par}$

$$\begin{aligned} & \left(\text{apply} \begin{array}{|c|c|c|c|} \hline f_0 & f_1 & \dots & f_{p-1} \\ \hline v_0 & v_1 & \dots & v_{p-1} \\ \hline \end{array} \right) \\ = & \begin{array}{|c|c|c|c|} \hline (f_0 \ v_0) & (f_1 \ v_1) & \dots & (f_{p-1} \ v_{p-1}) \\ \hline \end{array} \end{aligned}$$

BSP cost : $\max_{0 \leq i < p} w_i$

Communication

type α option = None | Some of α
 put: $(\text{int} \rightarrow \alpha \text{ option}) \text{ par} \rightarrow (\text{int} \rightarrow \alpha \text{ option}) \text{ par}$

$$\left(\text{put } \boxed{f_0 \quad f_1 \quad \dots \quad f_{p-1}} \right) = \boxed{g_0 \quad g_1 \quad \dots \quad g_{p-1}}$$

	0	1	2	3	
None	$(f_1 \ 0)$	$(f_2 \ 0)$	$(f_3 \ 0)$		⇒
None	$(f_1 \ 1)$	$(f_2 \ 1)$	$(f_3 \ 1)$		
None	$(f_1 \ 2)$	$(f_2 \ 2)$	$(f_3 \ 2)$		
Some v	$(f_1 \ 3)$	$(f_2 \ 3)$	$(f_3 \ 3)$		

	0	1	2	3
None	None	None	None	Some v
	$(g_0 \ 1)$	$(g_1 \ 1)$	$(g_2 \ 1)$	$(g_3 \ 1)$
	$(g_0 \ 2)$	$(g_1 \ 2)$	$(g_2 \ 2)$	$(g_3 \ 2)$
	$(g_0 \ 3)$	$(g_1 \ 3)$	$(g_2 \ 3)$	$(g_3 \ 3)$

$$\text{BSP cost} : \max_{0 \leq i < p} w_i + h \times g + L$$

Projection

`proj`: α option `par` \rightarrow (int \rightarrow α option)

$$\left(\text{proj } \boxed{v_0 \mid v_1 \mid \dots \mid v_{p-1}} \right) = \begin{array}{l} \text{function } 0 \quad \rightarrow v_0 \\ \quad \quad \quad 1 \quad \rightarrow v_1 \\ \quad \quad \quad \vdots \\ \quad \quad \quad p-1 \rightarrow v_{p-1} \end{array}$$

$$\text{BSP cost : } \max_{0 \leq i < p} w_i + h \times g + L$$

Examples

(1)

```
let noSome (Some x) = x
```

```
let procs () = [0;1;2;...;bsp_p()-1] (* Pseudo code *)
```

```
let replicate x = mkpar(fun pid → x)
```

```
let parfun f vec = apply (replicate f) vec
```

```
let parfun2 f vec1 vec2 = apply (parfun f vec1) vec2
```

```
let _ =
```

```
let print = fun i n → Printf.printf "Processor_%d_of_%d\n" i n in  
parfun2 print (mkpar(fun pid → pid)) (replicate (bsp_p()))
```

Examples

(2)

```
(* totex:  $\alpha$  par  $\rightarrow$  (int  $\rightarrow$   $\alpha$ ) par *)  
let totex vv = parfun (compose noSome)  
  (put(parfun (fun v dst $\rightarrow$  Some v) vv))
```

```
(* total_exchange:  $\alpha$  par  $\rightarrow$   $\alpha$  list par *)  
let total_exchange vec =  
  parfun2 List.map (totex vec) (replicate (procs()))
```

```
(* fold_direct: ( $\alpha \rightarrow \alpha \rightarrow \alpha$ )  $\rightarrow$   $\alpha$  par  $\rightarrow$   $\alpha$  par *)  
let fold_direct op vec =  
  let local_reduce = function h::t $\rightarrow$  List.fold_Left op h t in  
  parfun local_reduce (total_exchange vec)
```

Examples

(3)

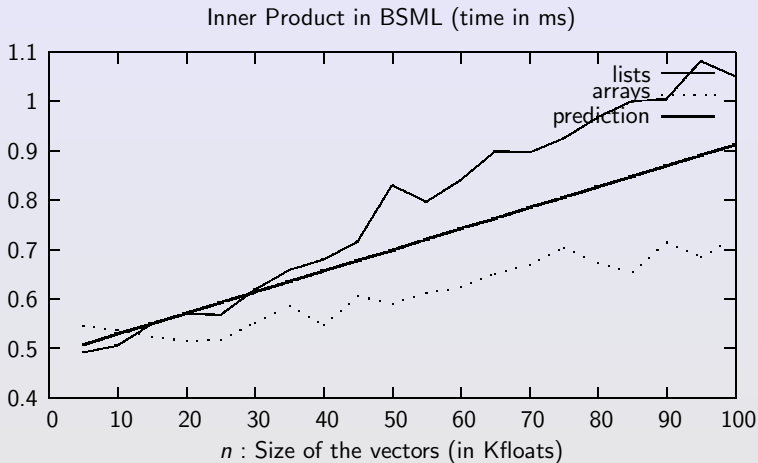
```
let inprod_array v1 v2 = let s = ref 0. in
  for i = 0 to (Array.length v1)-1 do
    s:=!s+.(v1.(i)*.v2.(i));
  done; !s
```

```
let inprod_list v1 v2 =
  List.fold_left2 (fun s x y → s+.x*.y) 0. v1 v2
```

```
let inprod seqinprod v1 v2 =
  let local_inprod = parfun2 seqinprod v1 v2 in
  fold_direct (+.) local_inprod
```

Examples

(4)



Overview

- Two parts :
 - Primitives
 - Standard library
- Modular implementation :
 - The module of primitives is a functor
 - Low-level communication module (MPI, PUB, TCP/IP, SEQ)
 - Module for input/output (PAR, SEQ)
 - Module for parallel composition (Generic)

Conclusions and Future Work

- Bulk Synchronous Parallel ML is :
 - Simple
 - Efficient
 - Predictable
 - Partially certified
- Current implementation : 0.25 (MPI)
- Next release : 0.5 (end of July)

<http://bsmlib.free.fr>

Other Languages

- Minimally Synchronous Parallel ML
 - (Almost) same set of primitives
 - Different distribution execution :
without global synchronization barrier
- Departmental Metacomputing ML
 - Metacomputer : set of clusters
 - Each cluster programmed with BSML
 - Coordination with MSPML-like primitives