# HYDROML, HYbriD PaRallel PrOgramming in ML

Victor Allombert

with

Frédéric GAVA and Julien TESSON

Laboratory of Algorithmic Complexity and Logic
Université Paris-Est

19 mars 2015

## Plan

## Plan

## Ocaml : a ML language

Strengths of Ocaml

## Ocaml : a ML language

### Strengths of Ocaml

- A functionnal programming language

## Ocaml : a ML language

### Strengths of Ocaml

- A functionnal programming language
- A powerful type system

## Ocaml : a ML language

### Strengths of Ocaml

- A functionnal programming language
- A powerful type system
- User-definable algebraic data types and pattern matching

## Ocaml : a ML language

### Strengths of Ocaml

- A functionnal programming language
- A powerful type system
- User-definable algebraic data types and pattern matching
- An expressive object-oriented layer

## Ocaml : a ML language

### Strengths of Ocaml

- A functionnal programming language
- A powerful type system
- User-definable algebraic data types and pattern matching
- An expressive object-oriented layer
- Automatic memory management

## Ocaml : a ML language

Strengths of Ocaml

- A functionnal programming language
- A powerful type system
- User-definable algebraic data types and pattern matching
- An expressive object-oriented layer
- Automatic memory management
- Efficient native code compilers

## The Coq proof assistant

**Coq implements :**

## The Coq proof assistant

**Coq implements :**

- High order logic

## The Coq proof assistant

Coq implements :

- High order logic
- Richly-typed functional programming language

## The Coq proof assistant

### Coq implements :

- High order logic
- Richly-typed functional programming language
- Predicates and theorems declaration

## The Coq proof assistant

**Coq implements :**

- High order logic
- Richly-typed functional programming language
- Predicates and theorems declaration
- Interactive proving

## The Coq proof assistant

**Coq implements :**

- High order logic
- Richly-typed functional programming language
- Predicates and theorems declaration
- Interactive proving
- Extract certified programs

## Bridging model : Bulk Synchronous Parallelism

## Bridging model : Bulk Synchronous Parallelism

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network
- Synchronization unit

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network
- Synchronization unit

### Properties :

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network
- Synchronization unit

### Properties :

- Confluent

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network
- Synchronization unit

### Properties :

- Confluent
- Deadlock-free

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network
- Synchronization unit

### Properties :

- Confluent
- Deadlock-free
- Predictable performances

## Bridging model : Bulk Synchronous Parallelism

### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network
- Synchronization unit

### Properties :

- Confluent
- Deadlock-free
- Predictable performances
- Super-steps execution

## Bridging model : Bulk Synchronous Parallelism

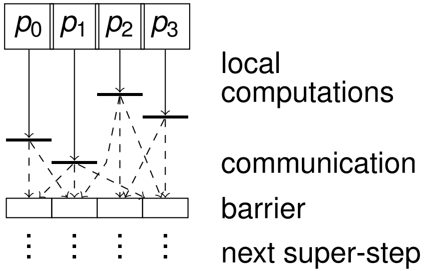
### The BSP computer

Defined by :

- **p** pairs CPU/memory
- Communication network
- Synchronization unit

### Properties :

- Confluent
- Deadlock-free
- Predictable performances
- Super-steps execution



## Bulk Synchronous ML

### What is BSML ?

## Bulk Synchronous ML

### What is BSML ?

- Explicit BSP programming with a functional approach

## Bulk Synchronous ML

### What is BSML ?

- Explicit BSP programming with a functional approach
- Based upon ML an implemented over Ocaml

## Bulk Synchronous ML

### What is BSML ?

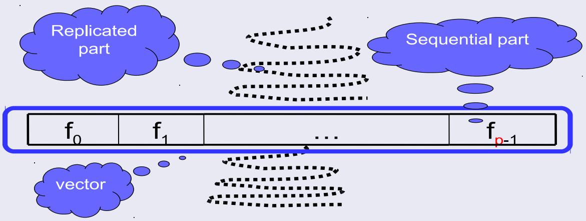
- Explicit BSP programming with a functional approach
- Based upon ML an implemented over Ocaml
- Formal sementics $\rightarrow$ computer-assisted proofs

## What is BSML ?

- Explicit BSP programming with a functional approach
- Based upon ML an implemented over Ocaml
- Formal sementics $\rightarrow$ computer-assisted proofs

## Main idea

Parallel data structure $\Rightarrow$ vectors :



$\langle\, v_0, \ldots, v_{p-1} \,\rangle : \alpha\, \text{par} \equiv v_i$ on node $i$

## Bulk Synchronous ML

### The main primitives

## Bulk Synchronous ML

### The main primitives

- mkpar : $(int \rightarrow' a) \rightarrow' a\ par$

## Bulk Synchronous ML

### The main primitives

- mkpar : $(int \rightarrow' a) \rightarrow' a\ par$
- proj : $'a\ par \rightarrow (int \rightarrow' a)$

## Bulk Synchronous ML

### The main primitives

- mkpar : $(int \rightarrow' a) \rightarrow' a\ par$
- proj : $'a\ par \rightarrow (int \rightarrow' a)$
- apply : $('a \rightarrow' b)\ par \rightarrow' a\ par \rightarrow' b\ par$

## Bulk Synchronous ML

### The main primitives

- mkpar : $(int \rightarrow' a) \rightarrow' a\ par$
- proj : $'a\ par \rightarrow (int \rightarrow' a)$
- apply : $('a \rightarrow' b)\ par \rightarrow' a\ par \rightarrow' b\ par$
- put : $(int \rightarrow' a)\ par \rightarrow (int \rightarrow' a) par$

| Model | | Language |
|---|---|---|
| BSP | $\rightarrow$ | BSML |

## Multi-BSP

### What is Multi-BSP ?

## What is Multi-BSP ?

1. A tree structure with nested components

## Multi-BSP

### What is Multi-BSP ?

1. A tree structure with nested components
2. where nodes have a storage capacity

# Multi-BSP

## What is Multi-BSP ?

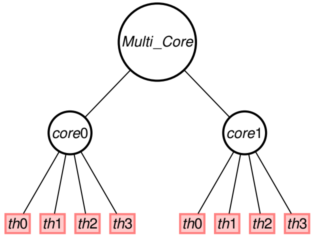
1. A tree structure with nested components
2. where nodes have a storage capacity
3. and leaf are processors

# Multi-BSP

## What is Multi-BSP ?

1. A tree structure with nested components
2. where nodes have a storage capacity
3. and leaf are processors

Multi-BSP

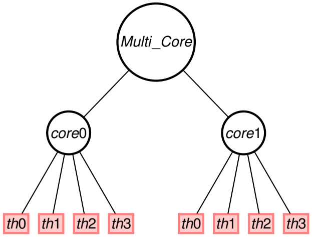


# Multi-BSP

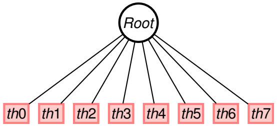
## What is Multi-BSP ?

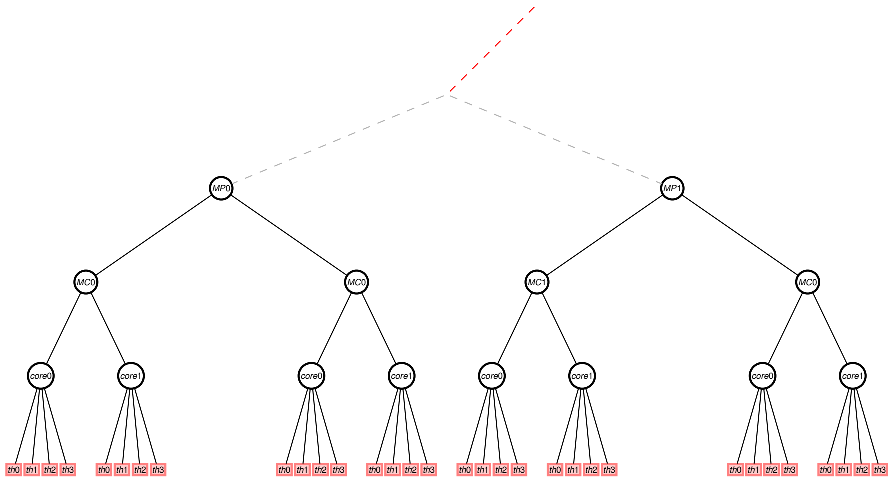
1. A tree structure with nested components
2. where nodes have a storage capacity
3. and leaf are processors

Multi-BSP



BSP




MP0
MP1
MC0
MC0
MC1
MC0
core0
core1
core0
core1
core0
core1
core0
core1
th0
th1
th2
th3

| Model | | Language |
|---|---|---|
| BSP | $\rightarrow$ | BSML |

| Model | | Language |
|---|---|---|
| BSP | $\rightarrow$ | BSML |
| M-BSP | $\rightarrow$ | |

| Model | | Language |
|---|---|---|
| BSP | $\rightarrow$ | BSML |
| M-BSP | $\rightarrow$ | H-BSML |

## Plan

## H-BSML

### Main ideas

## H-BSML

### Main ideas

- Easy way to program hierarchical architectures

## H-BSML

### Main ideas

- Easy way to program hierarchical architectures
- Hybrid programming

## H-BSML

### Main ideas

- Easy way to program hierarchical architectures
- Hybrid programming
- High-level programming

## H-BSML

### Main ideas

- Easy way to program hierarchical architectures
- Hybrid programming
- High-level programming
- Certified programs

## H-BSML

### Main ideas

- Easy way to program hierarchical architectures
- Hybrid programming
- High-level programming
- Certified programs
- Powerful type system

## H-BSML

### Main ideas

- Easy way to program hierarchical architectures
- Hybrid programming
- High-level programming
- Certified programs
- Powerful type system

### Goals

$$\text{Formal proof} \xrightarrow{\text{Extraction}} \text{H-BSML} \xrightarrow{\text{Compiler}} \text{Hierarchical program}$$

## Thank you for your attention !

Any questions ?