A formal semantics of the MULTI-ML language

VICTOR ALLOMBERT\textsuperscript{1}, Frédéric GAVA\textsuperscript{2}, JULIEN TESSON\textsuperscript{2}

\textsuperscript{1}LIFO - Université d’Orléans, France
\textsuperscript{2}LAACL - Université Paris Est, France

27 June 2018
Table of Contents

1 Introduction

2 Semantics

3 Conclusion
Table of Contents

1 Introduction
   Structured parallel computing
   BSP and BSML
   MULTI-BSP and MULTI-ML

2 Semantics

3 Conclusion
The world of parallel computing

Simulations:
- Fluid simulation
- 3D Visualisation

Big-Data:
- IoT
- Social Networking
- Data science

Symbolic computation:
- Model-Checking
- Formal computing

Super-computer
Hierarchical architectures

Characterised by:

- Interconnected units
- Both shared and distributed memories
- Hierarchical memories
A sequential bridging model

Von Neumann

Hardware

x86

x64

ARM

PowerPC

Software

Quick Sort

Compiler X

ML

C
A parallel bridging model

Hardware
- Multi-Core
- Cluster
- Super-computer
- GPU
- FPGA

Software
- Sorting algorithms
- Compilers
- Language
- Skeletons

V. Allombert, F. Gava, J. Tesson

ISPDC 2018
A parallel bridging model

Hardware

- Multi-core
- Cluster
- Supercomputer

Software

- Parallel Sorting by Regular Sampling
- Heat equation
- BSPLIB
- BSML

BSP
**Bulk Synchronous Parallelism**

### The BSP computer

Defined by:
- \(p\) pairs CPU/memory
- Communication network
- Synchronisation unit
- Super-steps execution

### Properties:
- Deadlock-free
- Predictable performances

![Diagram showing the BSP computer with p0, p1, p2, p3, local computations, communication barrier, and next super-step]
A parallel bridging model

Hardware

- Multi-core
- Cluster
- Super-computer

Software

- Parallel Sorting by Regular Sampling
- Heat equation
- BSPLIB
- 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 and implemented over OCAML
**Bulk Synchronous ML**

**What is BSML?**

- Explicit BSP programming with a functional approach
- Based upon ML and implemented over OCAML
- Formal semantics → computer-assisted proofs (COQ)
**Bulk Synchronous ML**

**What is BSML?**

- Explicit BSP programming with a functional approach
- Based upon ML and implemented over OCAML
- Formal semantics → computer-assisted proofs (COQ)

**Main idea**

Parallel data structure ⇒ *parallel vector*:

```
Replicated part (BSP) →
```

```
f_0
f_1
...
```

```
Sequential part
```

```
f_{p-1}
```

`parallel vector`
A parallel bridging model

Hardware
- Multi-core
- Cluster
- Super-computer

Software
- Parallel Sorting by Regular Sampling
- Heat equation
- BSPLIB
- BSML

BSP

Hierarchical architecture
A parallel bridging model

Why?
- Flat memories
- No sub-synchronisation

Hardware
- Multi-core
- Cluster
- Super-computer
- Hierarchical architecture

Software
- Parallel Sorting by Regular Sampling
- Heat equation
- BSPLIB
- BSML
A parallel bridging model

Hardware

- Multi-core
- Cluster
- Super-computer
- Hierarchical architecture

Software

- MULTI-BSP sorting
- State space
- MPI (sub-group)
- ????-ML
What is MULTI-BSP?
What is MULTI-BSP?

1. A tree structure with nested components
What is **MULTI-BSP**?

1. A tree structure with nested components
2. Where nodes have a storage capacity
What is MULTI-BSP?

1. A tree structure with nested components
2. Where nodes have a storage capacity
3. And leaves are processors
What is **MULTI-BSP**?

1. A tree structure with nested components
2. Where nodes have a storage capacity
3. And leaves are processors
4. With sub-synchronisation capabilities
What is **MULTI-BSP**?

- Stage 3: 4 nodes with a network access
- Stage 2: one node has 4 chips plus RAM
- Stage 1: one chip has 8 cores plus L3 cache
- Stage 0: one core with L1/L2 caches
The MULTI-BSP model

Execution model

A level $i$ superstep is:

- Level $i$ executes code independently
- Exchanges information with the $m_i$ memory
- Synchronises
A level $i$ superstep is:

- Level $i-1$ executes code independently
The MULTI-BSP model

Execution model

A level $i$ superstep is:

- Level $i - 1$ executes code independently
- Exchanges information with the $m_i$ memory
The MULTI-BSP model

Execution model

A level $i$ superstep is:

- Level $i - 1$ executes code independently
- Exchanges information with the $m_i$ memory
- Synchronises
A parallel bridging model

Hardware

- Multi-core
- Cluster
- Super-computer
- Hierarchical architecture

Software

- MULTI-BSP sorting
- State space
- MPI (sub-group)
- MULTI-ML
The MULTI-ML language

Basic ideas
The MULTI-ML language

Basic ideas

- BSML-like code on every stage of the MULTI-BSP architecture

```
let v = <<e>>
```

```
<< e ... e >>
```

```
Replicated part (BSP)
```

```
Sequential part
```

```
parallel vector
```

V. Allombert, F. Gava, J. Tesson
The MULTI-ML language

Basic ideas

- BSML-like code on every stage of the MULTI- BSP architecture
- Specific syntax over ML: eases programming

```
let v = <<e>>
```

Replicated part (BSP) → parallel vector

Sequential part
The **MULTI-ML** language

**Basic ideas**

- BSML-like code on every stage of the **MULTI-BSP** architecture
- Specific syntax over **ML**: eases programming
- *Multi-functions* that recursively go through the **MULTI-BSP** tree
MULTI-ML: Tree recursion

Recursion structure

```ml
let multi f [args]=
  where node =
    (* BSML code *)
    ...
    <<< f [args] >>>
    ... in v
  where leaf =
    (* OCaml code *)
    ... in v
```
MULTI-ML: Tree recursion

Recursion structure

```ocaml
globally:

let multi f [args] =
  where node =
    (* BSML code *)
    ...
    < f [args] >>
    ...
    in v
  where leaf =
    (* OCaml code *)
    ...
    in v
```
**MULTI-ML: Tree recursion**

Recursion structure

```ocaml
let multi f [args] =
    where node =
      (* BSML code *)
      ...
      << f [args] >>
      ...
    in v
where leaf =
  (* OCaml code *)
  ...
  in v
```
MULTI-ML: Tree recursion

Recursion structure

```ocaml
let multi f [args]=
  where node =
    (* BSML code *)
    ...
    <<< f [args] >>>
    ... in v
  where leaf =
    (* OCaml code *)
    ... in v
```

Result: $v_0$: 0, 1, 0, 1, 0

Diagram:

```
  f
 /   \
 |   |
 v_1  v_2
```

```
  f
 /   \
 |   |
 v_0  v_0
```
MULTI-ML: Tree recursion

Recursion structure

```
let multi f [args]=
    where node =
      (* BSML code *)
      ...
      <<< f [args] >>
      ... in v
    where leaf =
      (* OCaml code *)
      ...
      in v
```
MULTI-ML: Tree recursion

Recursion structure

```ocaml
let multi f [args] =
  where node =
    (* BSML code *)
    ...
    << f [args] >>
    ...
  in v
where leaf =
  (* OCaml code *)
  ...
  in v
```

Result

\[ v_{0.0.0} \quad v_{0.0.1} \quad v_{0.1.0} \quad v_{0.1.1} \]
MULTI-ML: Tree recursion

Recursion structure

```ocaml
let multi f [args]=
  where node =
    (* BSML code *)
    ...
    f [args] >>
    ...
  in v
where leaf =
  (* OCaml code *)
  ...
  in v
```

Result

$v_{0.0}$  $v_{0.1}$
MULTI-ML: Tree recursion

Recursion structure

```ocaml
let multi f [args]=
  where node =
    (* BSML code *)
    ...
  <<< f [args] >>>
  ...
  in v
where leaf =
  (* OCaml code *)
  ...
  in v
```

Result

\[ \uparrow v_0 \]
Table of Contents

1 Introduction

2 Semantics
   Execution model
   Big step semantics
   Semantics properties

3 Conclusion
The MULTI-ML localities
The **MULTI-ML** localities

![Diagram showing the multi-ML localities]

$m$
The **MULTI-ML** localities
The **MULTI-ML** localities

\[ m \]

\[ b \]

\[ s \]
The **MULTI-ML** localities

![Diagram of multi-ML localities]

- $m$
- $b$
- $s$

V. Allombert, F. Gava, J. Tesson

ISPDC 2018
The **MULTI-ML** localities

\[m\]

\[b\]

\[s\]

\[c\]
The **MULTI-ML** localities
A core language: $\mu$MULTI-ML

\[
\begin{align*}
e & ::= \quad x & \text{Variable} \\
& \mid \text{op, cst} & \text{Operator, Constant} \\
& \mid \text{let } x = e \text{ in } e & \text{Let binding} \\
& \mid \text{fun } x \rightarrow e & \text{Function} \\
& \mid \text{multi } f x \rightarrow e \uparrow e & \text{Multi-function} \\
& \mid (e e) & \text{Application} \\
& \mid \text{if } e \text{ then } e \text{ else } e & \text{Conditional} \\
& \mid \text{mkpar } e & \text{Parallel primitives} \\
& \mid \text{proj } e \mid \text{put } e & \text{Synchro. parallel primitives} \\
& \mid \ldots
\end{align*}
\]

\[
\begin{align*}
v & ::= \quad \text{Values} \\
& \mid \text{op, cst} & \text{Operator, Constant} \\
& \mid (\text{fun } x \rightarrow e)[\mathcal{E}] & \text{Closure} \\
& \mid (\text{multi } f x \rightarrow e \uparrow e)[\mathcal{E}] & \text{Multi-function closure} \\
& \mid (v, v) & \text{Pair} \\
& \mid < v, \ldots, v > & \text{Parallel vector} \\
& \mid \text{op} & \text{Basic operators} \\
& \mid \text{cst} & \text{Constants} \\
& \mid \mathcal{E} & \text{Environment}
\end{align*}
\]
Inference rules

Inductive inference rule:

\[ \frac{\mathcal{P}}{\mathcal{M} \vdash e \downarrow^c_p v} \]

Co-inductive inference rule:

\[ \frac{\mathcal{P}}{\mathcal{M} \vdash e \downarrow^c_p \infty} \]
Inference rules

Inductive inference rule:

\( \vdash_{P} e \downarrow P \downarrow_{P} v \)

(Multi-)environment

Co-inductive inference rule:

\( \vdash_{P} e \downarrow P \infty \)
Inference rules

Inductive inference rule:

\[
\frac{P}{\mathcal{M} \vdash e \Downarrow^P v}
\]

(Multi-)environment

Expression

Co-inductive inference rule:

\[
\frac{P}{\mathcal{M} \vdash e \Downarrow^P \infty}
\]
Inference rules

Inductive inference rule:

\[ P \]
\[ M \vdash e \downarrow_p^F v \]

(Multi-)environment  
Expression  
Position

Co-inductive inference rule:

\[ P \]
\[ M \vdash e \downarrow_p^F \infty \]
Inference rules

Inductive inference rule:

\[ \frac{\mathcal{P}}{\mathcal{M} \vdash e \Downarrow_p^L v} \]

(Multi-)environment

Expression

Position

Locality

Co-inductive inference rule:

\[ \frac{\mathcal{P}}{\mathcal{M} \vdash e \Downarrow_p^\infty} \]
Inference rules

Inductive inference rule:

\[
\frac{\mathcal{P}}{\mathcal{M} \vdash e \Downarrow_p \mathcal{v}}
\]

(Multi-)environment

Expression

Position

Locality

Value

Co-inductive inference rule:

\[
\frac{\mathcal{P}}{\mathcal{M} \vdash e \Downarrow_p \infty}
\]
Inductive inference rule:

\[
\frac{\mathcal{P} \rightarrow \mathcal{M} \vdash e \downarrow^L_{\mathcal{P}} v}{\mathcal{M} \vdash e \downarrow^L_{\mathcal{P}} v}
\]

Co-inductive inference rule:

\[
\frac{\mathcal{P} \rightarrow \mathcal{M} \vdash e \downarrow^L_{\mathcal{P}} \infty}{\mathcal{M} \vdash e \downarrow^L_{\mathcal{P}} \infty}
\]
Inference rules

Inductive inference rule:

\(\mathcal{P} \quad \frac{\mathcal{M} \vdash e \downarrow_p v}{\mathcal{M} \vdash e \downarrow_p v}\)

(Multi-)environment

Expression

Position

Locality

Value

Premises

Co-inductive inference rule:

\(\mathcal{P} \quad \frac{\mathcal{M} \vdash e \downarrow_p \infty}{\mathcal{M} \vdash e \downarrow_p \infty}\)

Divergence
The \texttt{mkpar} case

\begin{center}
\begin{tikzpicture}
  \node {\texttt{mkpar (fun i -> vi)}};
  \node[below] at (0,0) {\texttt{f0}};
  \node[below] at (0,-1) {\texttt{v0}};
  \node[below] at (1,-2) {\texttt{v1}};
  \node[below] at (1,-3) {\texttt{f1}};
  \node[below] at (-1,-3) {\texttt{v1}};
\end{tikzpicture}
\end{center}
The `mkpar` case

```
N
mkpar (fun i -> vi)
```

```
f 0; f 1
```
The \texttt{mkpar} case

\begin{center}
\begin{tikzpicture}
  \node [circle, draw] (v0) at (0,0) {\texttt{v0}};
  \node [circle, draw] (v1) at (1,0) {\texttt{v1}};
  \node [circle, draw, above] (mkpar) at (0.5,1) {mkpar (\texttt{fun i -> vi})};
  \draw [->] (mkpar) -- (v0);
  \draw [->] (mkpar) -- (v1);
  \draw [->] (v0) -- ++(0,-1);
  \draw [->] (v1) -- ++(0,-1);
  \draw [-, dashed] (v0) -- ++(0,-1);
  \draw [-, dashed] (v1) -- ++(0,-1);
\end{tikzpicture}
\end{center}
The \textit{mkpar} case

\textbf{Inductive rule:}

\[ \begin{align*}
\forall i \in p \quad & M \oplus_p E' \vdash e' \quad i \downarrow^b_p v_i \\
\therefore M \vdash \text{mkpar} \quad & e \downarrow^b_p \langle v_0, \ldots, v_n \rangle
\end{align*} \]

\textbf{MKPAR}

\[ \begin{align*}
M \vdash e \downarrow^b_p (e')[E'] \\
M \oplus_p E' \vdash e' \quad i \downarrow^b_p v_i
\end{align*} \]
The \texttt{mkpar} case

Co-inductive rule:

\[
\begin{align*}
\mathcal{M} \vdash e \Downarrow_p^b \infty \\
\text{MKPAR-E} & \quad \frac{}{\mathcal{M} \vdash \text{mkpar} \ e \Downarrow_p^b \infty} \\
\mathcal{M} \vdash e \Downarrow_p^b (e')[\mathcal{E'}] \\
\exists i \in p & \quad \mathcal{M} \oplus_p \mathcal{E}' \vdash e' \ i \Downarrow_p^b \infty \\
\text{MKPAR-V} & \quad \frac{}{\mathcal{M} \vdash \text{mkpar} \ e \Downarrow_p^b \infty}
\end{align*}
\]
The MULTI-ML semantics ...

VALUES
\[ \mathcal{M} \vdash c \downarrow^p_v c \]

VAR
\[ \{ x \mapsto v \} \in \text{lookup}(x, \mathcal{M}, p, \mathcal{E}) \]
\[ \mathcal{M} \vdash x \downarrow^p_v v \]

\[ \mathcal{E} = \text{select}(\mathcal{M}, \mathcal{F}(\text{fun } x \rightarrow v), p, \mathcal{E}) \]
\[ v \equiv (\text{fun } x \rightarrow e)(\mathcal{E}) \]
\[ \mathcal{M} \vdash \text{fun } x \rightarrow e \downarrow^p_v v \]

CLS
\[ \mathcal{M} \vdash e_1 \downarrow^p_v (\text{fun } x \rightarrow e)(\mathcal{E}) \]
\[ \mathcal{M} \vdash e_2 \downarrow^p_v v \]
\[ \mathcal{M} \vdash \text{fun } x \rightarrow e \downarrow^p_v v \]

APP
\[ \mathcal{M} \vdash e \downarrow^p_v v_1 \]
\[ \mathcal{M} \vdash \text{fun } x \rightarrow v_1 \downarrow^p_v v_2 \]
\[ \mathcal{M} \vdash \text{let } x = e_1 \text{ in } e_2 \downarrow^p_v v_2 \]

LET_IN
\[ \mathcal{M} \vdash e_1 \downarrow^p_v \mathcal{E} \]
\[ \mathcal{M} \vdash e_2 \downarrow^p_v v \]
\[ \mathcal{M} \vdash (e_1 \cdot e_2) \downarrow^p_v v' \]

REPlicate
\[ \forall i \in p \quad \mathcal{M} \downarrow^p \{ f \mapsto (\text{fun } _- \rightarrow e \mathcal{E}) \} \vdash f \downarrow^p_v v_i \]
\[ \mathcal{M} \downarrow^p \text{replicate } (\text{fun } _- \rightarrow e \mathcal{E}) \downarrow^p_v v_0, \ldots, v_n \]

DOWN
\[ \mathcal{M} \vdash x \downarrow^p_v v \]
\[ \mathcal{M} \vdash \text{down } x \downarrow^p_v v \]

MULTI_NODE
\[ \{ f \mapsto (\text{multi } f x \rightarrow e'_1 \downarrow^p_v e'_2)\mathcal{M}' \} \vdash e'_1 \downarrow^p_v v' \]
\[ \mathcal{M} \vdash (e_1 \cdot e_2) \downarrow^p_v v' \]

MULTI_LEAF
\[ \mathcal{M} \vdash e \downarrow^p_v \mathcal{E} \]
\[ \forall i \in p \quad \mathcal{M} \downarrow^p \{ f_i \mapsto (e_i)(\mathcal{E}) \} \vdash f_i \cdot x_i \downarrow^p_v v_i \]
\[ \mathcal{M} \downarrow^p \text{apply } e_1 \cdot e_2 \downarrow^p_v v_0, \ldots, v_n \]

MULTI_DEF
\[ \mathcal{M} \vdash (\text{multi } f x \rightarrow e_1 \downarrow^p_v e_2)\mathcal{M}' \]
\[ \mathcal{M} \vdash (\text{multi } f x \rightarrow e_1 \downarrow^p_v e_2)\mathcal{M}' \]

MULTI_CALL
\[ \mathcal{M} \vdash (\text{multi } f x \rightarrow e_1 \downarrow^p_v e_2)\mathcal{M}' \]
\[ \mathcal{M} \vdash (\text{multi } f x \rightarrow e_1 \downarrow^p_v e_2)\mathcal{M}' \]

MULTI_ROOT
\[ \mathcal{M} \vdash (\text{multi } f x \rightarrow e_1 \downarrow^p_v e_2)\mathcal{M}' \]
\[ \mathcal{M} \vdash (\text{multi } f x \rightarrow e_1 \downarrow^p_v e_2)\mathcal{M}' \]

MULTI_PUT
\[ \mathcal{M} \vdash e \downarrow^p_v \mathcal{E} \]
\[ \forall i, j \in p \quad \mathcal{M} \downarrow^p \{ f_i \mapsto (e_i)(\mathcal{E}) \} \vdash f_i \cdot j \downarrow^p_v v_{ij} \]
\[ \mathcal{M} \downarrow^p \text{put } e \downarrow^p_v \mathcal{E} \]
\[ \mathcal{M} \vdash (e_1 \cdot e_2) \downarrow^p_v v' \]
But why?

- Ensure consistency with the MULTI-BSP model
- Prove:
  - Evaluation determinism
    \[ \text{If } M \vdash e \Downarrow_p^C v_1 \text{ and } M \vdash e \Downarrow_p^C v_2 \text{ then } v_1 = v_2. \]
  - Evaluation (or not)
    \[ \text{It is impossible to obtain a value } v \text{ such that } M \vdash e \Downarrow_p^C v, \]
    \[ \text{or there exists a value } v \text{ such that } M \vdash e \Downarrow_p^C v. \]
  - Evaluation “does not go wrong”
    \[ \text{If } M \vdash e \Downarrow_p^C v \text{ and } M \vdash e \Downarrow_p^C \infty \text{ then there is a contradiction.} \]
Table of Contents

1 Introduction

2 Semantics

3 Conclusion
## Conclusion

### Presented work

- MULTI-BSP extension of ML: MULTI-ML
- Operational semantics
  - Evaluation determinism
  - Evaluation (or not)
  - Evaluation “does not go wrong” …

### Ongoing and future work

- Automatic cost analysis
- Type system
- Certified parallel programming
Thank you for your attention 😊

Questions ?