UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

BSPA : Une algèbre de processus BSP

Armelle Merlin and Gaétan Hains
Université d'Orléans, LIFO

Rapport N^o **2004-06**

A bulk-synchronous parallel process algebra

Armelle Merlin and Gaétan Hains

Abstract

The CCS (Calculus of Communicating System) process algebra is a well-known formal model of synchronization and communication, useful for the analysis of safety and liveness in protocols or distributed programs, and in more recent works their security properties. BSP (Bulk-synchronous parallelism) is an algorithm- and programming model of data-parallel computation. It is useful for the design, analysis and programming of scalable parallel algorithms.

Many current evolutions require the integration of distributed- and parallel programming: grid systems for sharing ressources across the internet, secure and reliable global access to parallel computer systems, geographic distribution of confidential data on randomly accessible systems, etc. Such software services must provide guarantees of safety, liveness, security together with scalable and reliable performance. Formal models are therefore needed to combine parallel performance and concurrent behaviour. With this goal in mind, we propose here an integration of BSP with CCS semantics, generalize its cost (performance) model and sketch its application to scheduling problems in meta-computing.

1 Introduction

The BSP execution model [10] represents a parallel computation on p identical processors as an alternating sequence of computation *supersteps* (p asynchronous computations) and communications supersteps (data exchanges between processors with global synchronization). The BSP *cost* model estimates execution times as multiples of the time for a sequential operation by a simple formula based on p, on the sequential speed of processors, and the network bandwidth g and latency L. The execution time of a computation superstep followed by a synchronization

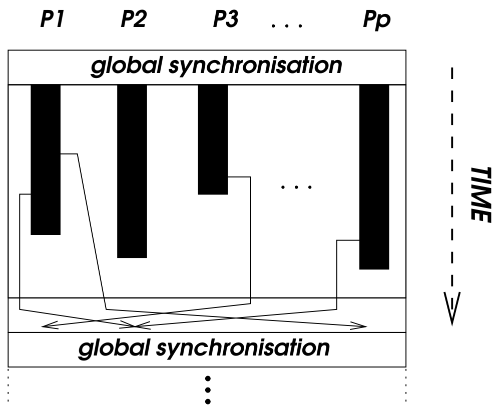

Figure 1: BSP superstep

superstep is estimated by $\max_{0\leq i<p} w_i + \max_{i:proc} h_i * g + L$
where $h_i = max(h_{i+}, h_{i-})$ and h_{i+} (resp. h_{i-}) is the total size of the messages sent (resp. received) by the processor i during the communication superstep. Here w_i is the sequential speed of the processor i during the computation superstep. This estimate has been shown to be accurate for almost all practical situations arising in parallel computing. Notable exceptions are: series of very small messages with barriers (useless for accelerating sequential computations) and heterogeneous architectures (which can however be represented by the cost model we will define below). BSP is thus a simplified and portable parallel architecture model, useful for algorithm design, scalability analysis and programming.

We formalized an ML-like BSP programming language by a BSP extension of the λ-calculus [5] called BS-λ. That model is appropriate for pure data-parallelism but lacks expressive power to describe meta-computing systems (where multiple data-parallel processes and/or multiple parallel systems interact), which leads to the work described here.

We will next summarize the CCS process algebra and propose its extension to express BSP-like processes. We then describe a formal cost model for this extended algebra in the standard graph theoretic manner: to associate "customized" semi-ring elements to sets of paths in the transition systems of CCS processes. The result is a flexible formal setup to describe and analyze complex mixtures of: data-parallelism and execution time, process synchronization, liveness and safety. We finally outline its application to scheduling problems in meta-computing, and some future developments.

2 A data-parallel process algebra

In this section, we describe the syntax and semantics of CCS and their adaptation to BSP, yielding a model we call BSPA. As in BSλ, the process algebra is divided in a local (resp. a global level) representing individual processors from parallel systems (resp. combinations of parallel systems and isolated processors).

2.1 Local processes

First we present the local level, which represents the actions that can be done locally by any processor of a parallel system.

2.1.1 Definitions

All process expressions are in the set $\mathcal{E}$. $\mathcal{E}_{FIN}$ is the set of terms without the `rec` operator. $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ is the set of *labels* where $\mathcal{A}$ is the set of input *labels*, and $\overline{\mathcal{A}}$ output *labels*. $\alpha \in \mathcal{A}$ and $\bar{\alpha} \in \overline{\mathcal{A}}$ are said to be inverses. Labels are also called *actions* or *events*.

Here is the syntax of process (or agent) expressions where $\alpha \in \mathcal{L}$ and X is any process variable:

- $P ::= 0 \mid \alpha.P \mid P + P \mid P\backslash\alpha \mid \texttt{let } X = P \texttt{ in } P \mid \texttt{rec } X.P \mid (P|P) \mid X$
- $\texttt{Act} ::= \alpha \mid \texttt{put}[\sigma] \mid \tau$ where $\alpha \in \mathcal{L}$.

  Remark : τ and `put` *have no inverse.*
- $\sigma ::= [i \leftarrow \lambda]^*$ the syntactic category of substitutions to express the exchange due to a `put`. i is the number of the processor where the λ substitutions are sent. It can be empty.

- $\lambda ::= [X \leftarrow P]^*$ the syntactic category Subst of substitutions that are to be sent. It can also be empty.

2.1.2 Local rules

The transition semantics of the local terms is as follows. The rule `ACT` relates to the prefix operator, `SUM` relates to summation, `RES` to restriction, `LET` to relabelling, `REC` to recursion, and `ASYNCG, PAR and ASYNCD` are the parallel composition's left-asynchronous, right-asynchronous or synchronous transition rules.

$$\texttt{ACT}\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\texttt{REC}\frac{P[X \leftarrow \text{rec}X.P] \xrightarrow{\alpha} P'}{\texttt{rec}X.P \xrightarrow{\alpha} P'}$$

$$\texttt{SUM}_0\frac{P_i \xrightarrow{\alpha} P_i'}{P_i + P_j \xrightarrow{\alpha} P_i'}$$

$$\texttt{AYNCG}\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\texttt{SUM}_1\frac{P_j \xrightarrow{\alpha} P_j'}{P_i + P_j \xrightarrow{\alpha} P_j'}$$

$$\texttt{PAR}\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\texttt{RES}\frac{P \xrightarrow{\alpha} P'}{P\backslash\beta \xrightarrow{\alpha} P'\backslash\beta}(\alpha \neq \{\beta, \bar{\beta}\})$$

$$\texttt{LET}\frac{Q[X \leftarrow P] \xrightarrow{\alpha} R}{\texttt{let } X = P \texttt{ in } Q \xrightarrow{\alpha} R}$$

$$\texttt{ASYNCD}\frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

2.2 Global processes

Then, we present the global level. The definitions, syntax and semantics are very similar, except concerning the synchronization barrier. For this chapter, we will always refer to a vector of fixed size which represents a fixed parallel machine of size p ([BSλ]-like).

2.2.1 Definitions

$\mathcal{E}_v$ is the set of global (process) expressions. $\mathcal{E}_{v,FIN}$ is the set of global expressions without `REC`.

We define:

- these following sets: input labels $\mathcal{A}_V = \mathcal{A} \cup \{< \alpha@i > | i \in 0,..,p-1, \alpha \in \mathcal{A}\}$, output labels $\bar{\mathcal{A}}_V = \bar{\mathcal{A}} \cup \{< \bar{\alpha}@i > | i \in 0,..,p-1, \bar{\alpha} \in \bar{\mathcal{A}}\}$ and all labels $\mathcal{L}_V = \mathcal{A} \cup \bar{\mathcal{A}}$ for global level.

- the processor position numbers `i` $::= 0 \mid 1 \mid ... \mid p-1$.

- the terms `V` $::= 0_v \mid \alpha.V \mid V + V \mid V\backslash\alpha \mid$ `let` $X = V$ `in` $V \mid$ `rec` $X.V$ $\mid (V|V) \quad \mid < P,..,P > \mid X$
  where $\alpha \in$ Act.

  For every $p \geq 1$ there are global process vectors $< P,..,P >_p$, representing a group of local processes engaged in a collective data-parallel computation, such as an MPI *communicator*. The size is left unspecified when not relevant. The p processes are assumed to be allocated to p distinct processors.

- the actions $\texttt{Act}_V ::= \alpha \mid T_{\sigma,\pi} \mid \ < \tau@ij >_p \ \mid \tau_{Net}$ where $\alpha \in \mathcal{L}_V$

  - $< \alpha@i >_p$ is the asynchronous occurrence of action α on processor i within a vector of size p. $< \alpha@i >_p$ can synchronize with the action $\bar{\alpha}$ or $< \bar{\alpha}@j >_p$ with $i \neq j$.

- $T_{\sigma,\pi}$ is an abreviation of $T_{\forall i \forall j \sigma_i \pi_j}$. It is analogous to τ for global synchronization. σ_i is the set of substitutions send by the processor i, and π_j is the set of substitutions received by the processor j.
- $< \tau @ ij >_p$ is the τ synchronization between two different ($i \neq j$) processes inside the parallel vector .
- τ_{Net} is the synchronization between processors over the external network, i.e. between two parallel vectors.

For simplicity, a process of form $\alpha.0$ will sometimes be written α.

2.2.2 Global rules

The transition semantics of global processes is defined by the following set of rules.

$$\texttt{ACT}\frac{}{\alpha.V \xrightarrow{\alpha} V} \qquad \texttt{LET}\frac{W[X \leftarrow V] \xrightarrow{\alpha} W'}{\texttt{let } X = V \texttt{ in } W \xrightarrow{\alpha} W'}$$

$$\texttt{REC}\frac{V[X \leftarrow \mathrm{rec}X.V] \xrightarrow{\alpha} V'}{\texttt{rec } X.V \xrightarrow{\alpha} V'}$$

$$\texttt{SUM}_0\frac{V_i \xrightarrow{\alpha} V_i'}{V_i + V_j \xrightarrow{\alpha} V_i'} \qquad \texttt{ASYNCGV}\frac{V_0 \xrightarrow{\alpha} V_0'}{V_0|V_1 \xrightarrow{\alpha} V_0'|V_1}$$

$$\texttt{SUM}_1\frac{V_j \xrightarrow{\alpha} V_j'}{V_i + V_j \xrightarrow{\alpha} V_j'} \qquad \texttt{ASYNCDV}\frac{V_1 \xrightarrow{\alpha} V_1'}{V_0|V_1 \xrightarrow{\alpha} V_0|V_1'}$$

$$\texttt{RES}\frac{V \xrightarrow{\alpha} V'}{V\backslash\beta \xrightarrow{\alpha} V'\backslash\beta}(\alpha \neq \beta \neq \bar{\beta}) \qquad \texttt{PARV}\frac{V_0 \xrightarrow{\alpha} V_0' \quad V_1 \xrightarrow{\bar{\alpha}} V_1'}{V_0|V_1 \xrightarrow{\tau_{Net}} V_0'|V_1'}$$

$$\texttt{ASYNC}\frac{P_i \xrightarrow{\alpha} P_i'}{<P_0,\ldots,P_i,\ldots P_{p-1}> \xrightarrow{<\alpha @ i>_p} <P_0,\ldots,P_i',\ldots,P_{p-1}>} \quad \alpha \in \mathcal{L}_V \cup \{\tau\}$$

This rule is an emission (or reception), local at i inside the parallel vector.

$$\texttt{DUOCHRONE}\frac{P_i \xrightarrow{\alpha} P_i' \quad P_j \xrightarrow{\bar{\alpha}} P_j' \quad i \neq j}{<P_0,\ldots,P_i,\ldots,P_j\ldots,P_{p-1}> \xrightarrow{<\tau @ ij>_p} <P_0,\ldots,P_i',\ldots,P_j'\ldots,P_{p-1}>}$$

This rule is an exchange between two processes inside the parallel vector. Reminder : $\alpha \neq \texttt{put}$.

$$\texttt{ISOCHRONE}\frac{P_i \xrightarrow{\texttt{put}_{[\sigma_i]}} P_i' \quad \forall i \in \{0,\ldots,p-1\}}{<P_0,\ldots,P_{p-1}> \xrightarrow{T_{\sigma,\pi}} <\pi_0(P_0'),\ldots,\pi_{p-1}(P_{p-1}')>} \quad \pi_i = \sigma_{(p-1)i} \circ \ldots \circ \sigma_{0i}$$

This last rule is the semantics of the barrier. Processor i realizes the event $\texttt{put}[\sigma_i]$. σ_i is the set of the subsitutions to be sent from P_i. π_j is the set of substitutions received by P_j. To compute the cost following the BSP model, we calculate h_i^+ and h_i^- which represents the cost of the emission (resp. reception) at the process i. They are equal to the value of outgoing (incoming) communicated terms, which is calculated as follow :

$h_i^+ = \sum_{j=0}^{p-1} |\sigma_{ij}|$ and $h_i^- = \sum_{k=0}^{p-1} |\sigma_{ki}|$

If $\sigma = X_1 \leftarrow P_1, \ldots X_n \leftarrow P_n$ then $|\sigma| = \sum |P_i|$.

To cope with possible concurrent substitution into the same variable, we define priority "to the left". The cost in time, a key ingredient of the BSP theory, will be defined in section 4.2 from this mesure of communication volume.

As a final remark about BSPA's definition we observe that the ASYNC rule and its actions of the form $< \alpha @ i >$ will combine in our cost model so as to realize the following principle of parallel computation.

Remark 1 (Eagerness principle) *We assume that when a process is ready to do an action, it does it.*

Together with the idea that events can be simultaneous, this principle is the basis of all parallel performance models including BSP. The difficulty in combining it with CCS comes from the process algebra's so-called *interleaving semantics* whereby two events may commute in a parallel composition but are not explicitly simultaneous. For example $a|b = a.b + b.a$. Our solution to this dilemma is to have two parallel composition operators ($|$ and $<,,,>_p$) with similar interleaving semantics but different values in the cost model as we will see later.

2.3 Examples

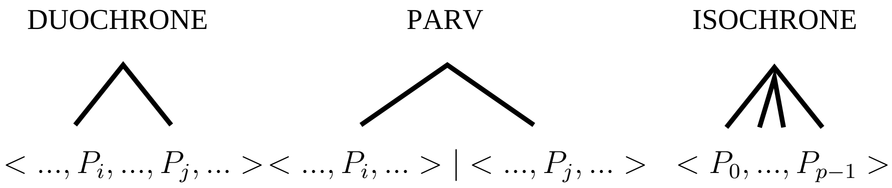

Figure 2: Main cases of communication

2.3.1 ASYNC+DUO

Here is an example of a two-processor parallel system. Processors 0 and 1 synchronize through rule DUOCHRONE.

$$\frac{a \xrightarrow{a} 0 \quad \bar{a} \xrightarrow{\bar{a}} 0}{< a, \bar{a} > \xrightarrow{\tau @ 01} < 0, 0 >}$$

2.3.2 ASYNC+PARV

Two different parallel systems, represented by terms with root $<,,>$, can synchronize as in CCS. If two synchronizations of this type are possible, they can not be simultaneous. This represents the CCS semantics we keep as part of the theory, and means that the external network serializes communications.

$$\begin{array}{ccc} < a, b, c > \mid < \bar{b}, \bar{a} > & \xrightarrow{\tau_{Net}} & < 0, b, c > \mid < \bar{b}, 0 > \\ \downarrow \tau_{Net} & & \downarrow \tau_{Net} \\ < a, 0, c > \mid < 0, \bar{a} > & \xrightarrow{\tau_{Net}} & < 0, 0, c > \mid < 0, 0 > \end{array}$$

This property of our formalism is meant to 1. perserve CCS semantics for a subset of the calculus 2. model through sequences of τ_{Net} the (very) low bandwidth of inter-system networks relative to the internal bandwidth of parallel systems.

2.3.3 ISOCHRONE with $p = 4$

This example demonstrates a communication supserstep (and hence synchronization barrier) between four processors. Processor 3 does not send any message, and processors 0 and 2 send a message to the processor 1.

Where $\pi_0 = id \circ (X \leftarrow a.0) \circ (Y \leftarrow c.0, X \leftarrow d.0) \circ id$
$\pi_0(P'_0) = \pi_0(X + Y) = d.0 + c.0$
$\pi_1 = id \circ (X \leftarrow c.0) \circ id \circ (X \leftarrow a.0)$
$\pi_1(P'_1) = \pi_1(X) = a.0$
$\pi_2 = id \circ id \circ id \circ (X \leftarrow b.0)$
$\pi_2(P'_2) = \pi_2(X) = b.0$ $\pi_3 = id \circ (Y \leftarrow b.0) \circ (X \leftarrow a.0) \circ id$
$\pi_3(P'_3) = \pi_3(X + Y) = a.0 + b.0$

$$
<\mathtt{put}\begin{bmatrix}1 \leftarrow [X \leftarrow \alpha] \\ 2 \leftarrow [X \leftarrow \beta]\end{bmatrix}.(X+Y), \mathtt{put}\begin{bmatrix}0 \leftarrow [Y \leftarrow \gamma, X \leftarrow \delta] \\ 3 \leftarrow [X \leftarrow \alpha]\end{bmatrix}.X, \mathtt{put}\begin{bmatrix}0 \leftarrow [X \leftarrow \alpha] \\ 1 \leftarrow [X \leftarrow \gamma] \\ 3 \leftarrow [Y \leftarrow \beta]\end{bmatrix}.X, \mathtt{put}[].(X+Y)>
$$

$$
\Downarrow T_{\sigma,\pi}
$$

$$
< \delta + \gamma, \alpha, \beta, \alpha + \beta >
$$

$$
\Downarrow s
$$

$$
< 0, 0, 0, 0 >
$$

Figure 3: Example: ISOCHRONE

π_1 illustrates how the different substitutions are applied and the left priority when multiple processors send substitutions to a same variable on a given destination processor.

3 Strong bisimulation

We now need to define the strong bisimulation relation for both levels of BSPA. We cite Milner from [7] and [6] for the local level. And then, we extend these definitions to the global level. They will allow us to validate transformation rules to reduced our process expressions to serial forms (explicit descriptions of the transition system), and define cost on those.

Definition 1 *[7] Let $\mathcal{F}$ be a function on binary relations $\mathcal{L} \subseteq \mathcal{P} \times \mathcal{P}$ such that $(P, Q) \in \mathcal{F}(\mathcal{L})$ if and only if, for every $\alpha \in \mathcal{A}$,*

1. *whenever $P \xrightarrow{\alpha} P'$ then $\exists Q'$ such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{L}$*
2. *whenever $Q \xrightarrow{\alpha} Q'$ then $\exists P'$ such that $P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{L}$*

Definition 2 *[7] $\mathcal{L} \subseteq \mathcal{P} \times \mathcal{P}$ is a strong bisimulation if $\mathcal{L} \subseteq \mathcal{F}(\mathcal{L})$*

Proposition 1 *[7] $\mathcal{F}$ is monotonic. $\mathcal{L} \subseteq \mathcal{L}'$ implies $\mathcal{F}(\mathcal{L}) \subseteq \mathcal{F}(\mathcal{L}')$*

Definition 3 *[7] Inverse and composition are defined as follows : $\mathcal{L}^{-1} = \{(Q, P) : P\mathcal{L}Q\}$ $\mathcal{L}\mathcal{L}' = \{(P, R) : \exists Q, P\mathcal{L}Q$ and $Q\mathcal{L}R\}$*

Proposition 2 *[7] If $\mathcal{L}_i (i \in I)$ are bisimulations, then so are $\bigcup_{i \in I} \mathcal{L}_i$, $\mathcal{L}_i^{-1}$ and $\mathcal{L}_i \mathcal{L}_j$. We write $P \sim Q$ and call P, Q bisimilar if $(P, Q) \in \mathcal{L}$ for some bisimulation $\mathcal{L}$.*

3.1 Properties of strong bisimulation for local terms

$(\mathcal{P}/\sim, +, 0)$ is an idempotent monoid.

Proposition 3 *Monoid laws[7]*

- $P + Q \sim Q + P$
- $P + P \sim P$
- $P + (Q + R) \sim (P + Q) + R$
- $P + 0 \sim P$

Proposition 4 *Static laws [7]*

- $P|Q \sim Q|P$

- $P|(Q|R) \sim (P|Q)|R$
- $P|0 \sim P$
- $\mathtt{let}\ X = P\ \mathtt{in}\ Q \sim Q[X \leftarrow P]$
- $P\backslash\alpha \sim P$ *if* $\alpha \notin \mathcal{L}(P)$
- $P\backslash\alpha\backslash\beta \sim P\backslash\beta\backslash\alpha$
- $(Q|P)\backslash\alpha \sim Q\backslash\alpha|P\backslash\alpha$ *if* $\alpha \notin \mathcal{L}(P)$ *and* $\alpha \notin \mathcal{L}(Q)$

Proposition 5 *Expansion law [7]*
Let $P \equiv (P_0|P_1|...|P_n)\backslash\beta\ \ n \geq 1$, *then*
$P \sim \sum\{\alpha_i.(P_0|...|P_i'|...|P_n)\backslash\beta : P_i \stackrel{\alpha_i}{\to} P_i', i \leq n, \alpha_i \neq \beta\}$
$\quad + \sum\{\tau.(P_0|...|P_i'|...|P_j'|...|P_n)\backslash\beta : P_i \stackrel{\alpha_i}{\to} P_i', P_j \stackrel{\alpha_j}{\to} P_j', i \leq j \leq n, \alpha_i = \bar{\alpha_j}\}$

Definition 4 *[6] An agent expression is* finite *if it contains no recursion or constants.*

Definition 5 *[6] An agent expression is* serial *if it contains neither parallel composition, nor restriction.*

Lemma 1 *[6] By the expansion law, every finite agent is equivalent to a finite serial agent.*

Milner has proposed an axiom system $\mathcal{A}_1$ for the (bisimulation) equivalence of serial terms:

1. $P + Q = Q + P$
2. $P + (Q + R) = (P + Q) + R$
3. $P + P = P$
4. $P + 0 = P$

Proposition 6 *[6] If* P, Q *are serial terms then* $P \sim Q \Leftrightarrow \mathcal{A}_1 \vdash P = Q$

3.2 Properties of strong bisimulation for global terms

Our parallel system constructor reduces to CCS parallel composition in the absence of barrier synchronizations.

Lemma 2 $< P, Q > \sim P|Q$ *if* $\mathtt{put} \notin P$ *and* $\mathtt{put} \notin Q$

The processor of a parallel system are interchangeable from the point of view of semantics.

Lemma 3 $< P_0, ...P_{p-1} > \sim < P_{\sigma(0)}, ..., P_{\sigma(p-1)} >$ *where* $\sigma \in Sym\{0, ..., (p-1)\}$ *is a permutation.*

The following laws are direct consequences of our definitions.

Proposition 7 *Monoid laws for global terms*

- $V + W \sim W + V$
- $V + V \sim V$
- $U + (V + W) \sim (U + V) + W$

- $V + 0_v \sim V$

Proposition 8 *Static laws for global terms*

- $V|W \sim W|V$
- $U|(V|W) \sim (U|V)|W$
- $V|0_v \sim V$
- $\texttt{let } X = V \texttt{ in } W \sim W[X \leftarrow V]$
- $V\backslash\alpha \sim V$ *if* $\alpha \notin \mathcal{L}(V)$
- $V\backslash\alpha\backslash\alpha \sim V\backslash\alpha$
- $V\backslash\alpha\backslash\beta \sim V\backslash\beta\backslash\alpha$
- $(W|V)\backslash\alpha \sim W\backslash\alpha|V\backslash\alpha$ *if* $\alpha \notin \mathcal{L}(V)$ *and* $\alpha \notin \mathcal{L}(W)$

Remark 2 *Non-bisimilar expressions*

- $< P_0, ..., P_{p-1} > \backslash\alpha \not\sim < P_0\backslash\alpha, ..., P_{p-1}\backslash\alpha >$
- $< ..., P_{i-1}, 0, P_{i+1}, ... > \not\sim < ..., P_{i-1}, P_{i+1}, ... >$
- $< P + Q, R > \not\sim < P, R > + < Q, R >$
- $< ... >_p \mid < ... >_p \not\sim < ... >_{2p}$
- $a. < P_0, ..., P_{p-1} > \not\sim < a.P_0, ..., a.P_{p-1} >$

Proposition 9 *Expansion law for data-parallel vectors*
Let $V \equiv < P_0, ..., P_{p-1} >)\backslash\beta$ *then*

$$
\begin{aligned}
V \sim & \sum < \alpha@i >_p .\{< ..., P_{i-1}, P'_i, P_{i+1}, ... >: \exists i P_i \xrightarrow{\alpha} P'_i, \alpha \neq \beta, \alpha \neq \texttt{put}\} \\
& + \sum < \tau@ij >_p .\{< ..., P'_i, ..., P'_j, ... >: P_i \xrightarrow{\alpha} P'_i, P_j \xrightarrow{\bar{\alpha}} P'_j, i \neq j\} \\
& + \sum T_{\sigma,\pi}.\{< P'_0, ..., P'_{p-1} >: \forall i P_i \xrightarrow{\texttt{put}[\sigma_i]} P'_i\}
\end{aligned}
$$

Proposition 10 *Expansion law for global parallel compositions (finite terms)*
Let $V \equiv (V_0|V_1|...|V_n)\backslash\beta \;\; n \geq 1$ *then*

$$
\begin{aligned}
V \sim & \sum\{\alpha.(V_0|...|V'_i|...|V_n)\backslash\beta : V_i \xrightarrow{\alpha} V'_i, \alpha \neq \beta\} \\
& + \sum\{\tau_{Net}.(V_0|...|V'_i|...|V'_j|...|V_n)\backslash\beta : V_i \xrightarrow{\alpha} V'_i, V_j \xrightarrow{\bar{\alpha}} V'_j, i \neq j\}
\end{aligned}
$$

Proof : Let $V \equiv (V_0|V_1|...|V_n)\backslash\beta$ and $W = \sum\{\alpha.(V_0|...|V'_i|...|V_n)\backslash\beta : V_i \xrightarrow{\alpha} V'_i, \alpha \neq \beta\} + \sum\{\tau_{Net}.(V_0|...|V'_i|...|V'_j|...|V_n)\backslash\beta : V_i \xrightarrow{\alpha} V'_i, V_j \xrightarrow{\bar{\alpha}} V'_j, i \neq j\}$. We must prove $V \sim W$ and proceed by induction on the length of the sums in W.

- Case of V and W of minimal size : $V = 0_V|...|\alpha.0_V|...|\bar{\alpha}.0_V|...|0_V$ and $W = \alpha.(0_V|...|\bar{\alpha}|...|0_V) + \bar{\alpha}.(0_V|...|\alpha|...|0_V) + \tau_{Net}.(0_V|...|0_V)$. Three different rules can by applied to V: $V \xrightarrow{\alpha} 0_V|...|0_V|...|\bar{\alpha}.0_V|...|0_V$, $V \xrightarrow{\bar{\alpha}} 0_V|...|\alpha.0_V|...|0_V|...|0_V$ or $V \xrightarrow{\tau_{Net}} 0_V|...|0_V|...|0_V|...|0_V$. The prefix rule can be applied to W and then lead respectively to $0_V|...|\bar{\alpha}|...|0_V$, $0_V|...|\alpha|...|0_V$ or $0_V|...|0_V$ by transitions on the corresponding events, which proves that $V \sim W$.
- Case $V \xrightarrow{\alpha} V'$. That means $\exists i : V_i \xrightarrow{\alpha} V'_i$, then $W \xrightarrow{\alpha} W'$ because $W' = \sum\{\alpha.(V_0|...|V'_i|...|V_n)\backslash\beta\} + \sum\{\tau_{Net}.(V_0|...|V'_i|...|V'_j|...|V_n)\backslash\beta : V_i \xrightarrow{\alpha} V'_i, V_j \xrightarrow{\bar{\alpha}} V'_j, i \neq j\}$

- Case $V \stackrel{\tau_{Net}}{\rightarrow} V'$ that means $\exists i \exists j \neq i : V_i \stackrel{\alpha}{\rightarrow} V_i', V_j \stackrel{\bar{\alpha}}{\rightarrow} V_j'$, then $W \stackrel{\tau_{Net}}{\rightarrow} W'$ because
  $W' = \sum\{\alpha.(V_0|...|V_i'|...|V_n)\backslash\beta : V_i \stackrel{\alpha}{\rightarrow} V_i', \alpha \neq \beta\}$
  $+ \sum\{\tau_{Net}.(V_0|...|V_i'|...|V_j'|...|V_n)\backslash\beta\}$

It is trivial to prove that for each transition on W there is a corresponding transition for V and conclude $V \sim W$.

Definition 6 *An agent expression is* finite *if it contains no recursion or constants.*

Definition 7 *An agent expression E is* serial *if it contains neither parallel composition (CCS $|$ or data-parallel $< .. >$), nor restriction.*

Lemma 4 *By the expansion laws, every finite agent is equivalent to a finite serial agent.*

Lemma 5 *A global serial term is bisimilar to a sum (≥ 1) of prefixed terms of form $\alpha.P$.*

Proof : Trivial through the syntax.

Proposition 11 *If V, W are serial finite terms then $V \sim W$ if and only if $\mathcal{A}_1 \vdash V = W$*

Proof : We observe that $U \rightarrow U' \Rightarrow |U| > |U'|$ with $U, U' \in \mathcal{E}_{v,FIN}$ a serial term (the only transition rule which does not decrease the size of terms is ISOCHRONE, not applicable to serial terms).

Correctness: we prove $\mathcal{A}_1 \vdash L = R \Rightarrow L \sim R$ by (implicit) induction on the size of the term: every axiom is correct i.e. preserves $\sim$. : Thanks to lemma 5 L and R are bisimilar to sums of terms.

1. Associativity. Let $L \equiv U + (V + W)$ and $R \equiv (U + V) + W$ which verify that $\mathcal{A}_1 \vdash L = R$. Let α be such that $L \stackrel{\alpha}{\rightarrow} L'$ and $R \stackrel{\alpha}{\rightarrow} R'$. By induction, we suppose that $\mathcal{A}_1 \vdash L' = R' \Rightarrow L' \sim R'$. $L \stackrel{\alpha}{\rightarrow} L'$ holds because either $U \stackrel{\alpha}{\rightarrow} U'$ either $V \stackrel{\alpha}{\rightarrow} V'$ or $W \stackrel{\alpha}{\rightarrow} W'$. Then $L \stackrel{\alpha}{\rightarrow} U'$ resp. $L \stackrel{\alpha}{\rightarrow} V'$ resp. $L \stackrel{\alpha}{\rightarrow} W'$ and $R \stackrel{\alpha}{\rightarrow} U'$ resp. $R \stackrel{\alpha}{\rightarrow} V'$ resp. $R \stackrel{\alpha}{\rightarrow} W'$. Trivially $\mathcal{A}_1 \vdash L' = R'$ (since L' and R' are the same term) and $L' \sim R'$. Finally, $L \sim R$.

2. Commutativity. Let $L \equiv U + V$ and $R \equiv V + U$ which verify that $\mathcal{A}_1 \vdash L = R$. Let α be such that $L \stackrel{\alpha}{\rightarrow} L'$ and $R \stackrel{\alpha}{\rightarrow} R'$. By induction, we suppose that $\mathcal{A}_1 \vdash L' = R' \Rightarrow L' \sim R'$. $L \stackrel{\alpha}{\rightarrow} L'$ holds because either $U \stackrel{\alpha}{\rightarrow} U'$ or $V \stackrel{\alpha}{\rightarrow} V'$. Then $L \stackrel{\alpha}{\rightarrow} U'$ resp. $L \stackrel{\alpha}{\rightarrow} V'$ and $R \stackrel{\alpha}{\rightarrow} U'$ resp. $R \stackrel{\alpha}{\rightarrow} V'$. Thus $\mathcal{A}_1 \vdash L' = R'$ and $L' \sim R'$. Finally, $L \sim R$.

3. Idempotence. Let $L \equiv U + U$ and $R \equiv U$ which verify that $\mathcal{A}_1 \vdash L = R$. Let α such that $L \stackrel{\alpha}{\rightarrow} L'$ and $R \stackrel{\alpha}{\rightarrow} R'$. By induction, we suppose that $\mathcal{A}_1 \vdash L' = R' \Rightarrow L' \sim R'$. If $L \stackrel{\alpha}{\rightarrow} L'$ it is because $U \stackrel{\alpha}{\rightarrow} U'$. Then $L \stackrel{\alpha}{\rightarrow} U'$ and $R \stackrel{\alpha}{\rightarrow} U'$. Thus $\mathcal{A}_1 \vdash L' = R'$ and $L' \sim R'$. Finally, $L \sim R$.

4. Unit. Let $L \equiv U + 0$ and $R \equiv U$ which verify that $\mathcal{A}_1 \vdash L = R$. Let α such that $L \stackrel{\alpha}{\rightarrow} L'$ and $R \stackrel{\alpha}{\rightarrow} R'$. By induction, we suppose that $\mathcal{A}_1 \vdash L' = R' \Rightarrow L' \sim R'$. If $L \stackrel{\alpha}{\rightarrow} L'$ it is because $U \stackrel{\alpha}{\rightarrow} U'$. Then $L \stackrel{\alpha}{\rightarrow} U'$ and $R \stackrel{\alpha}{\rightarrow} U'$. Thus $\mathcal{A}_1 \vdash L' = R'$ and $L' \sim R'$. Finally, $L \sim R$.

Completeness: We prove $L \sim R \Rightarrow \mathcal{A}_1 \vdash L = R$. Let α such that $L \stackrel{\alpha}{\rightarrow} L'$ and $R \stackrel{\alpha}{\rightarrow} R'$. By induction, we suppose that $L' \sim R' \Rightarrow \mathcal{A}_1 \vdash L' = R'$ Assuming $L \sim R$ we recall that both terms are bisimilar to a sum and consider three cases :

1. $L = \alpha.L'$ and $R = \alpha.R'$. Then $L' \sim R'$, and thanks to induction $\mathcal{A}_1 \vdash L' = R'$, thus $\mathcal{A}_1 \vdash \alpha.L' = \alpha.R'$, that is to say $\mathcal{A}_1 \vdash L = R$.

2. $L = \alpha.L'$ and $R = \sum_j^n \beta_j.R'_j$. As $L \sim R$ $\forall j \beta_j = \alpha$ and $L' \sim R'_j$. Thanks to induction $\forall j L' \sim R'_j \Rightarrow \mathcal{A}_1 \vdash L' = R'_j$. Thus $\forall j \mathcal{A}_1 \vdash \alpha.L' = \alpha.R'_j$ and $\mathcal{A}_1 \vdash L = R$

3. $L = \sum_i^n .L'_i$ and $R = \sum_j^n \beta_j.R'_j$. Remark : if $V = \sum_i^k \alpha_i V_i'^l$, $W = \sum_j^l \beta_j W_j'^l$ and $V \sim W$ then $\forall \alpha_i \exists \beta_j : \alpha_i = \beta_j$ and $V'_i \sim W'_j$, and $\forall \beta_j \exists \alpha_i : \alpha_i = \beta_j$ and $V'_i \sim W'_j$. $L \sim R$ implies $\forall i \exists j V'_i \sim W'_j \Rightarrow \forall i \exists j \mathcal{A}_1 \vdash V'_i = W'_j \Rightarrow \forall i \exists j \mathcal{A}_1 \vdash \alpha_i.V'_i = \beta_j.W'_j$ and $\forall j \exists i V'_i \sim W'_j \Rightarrow \forall j \exists i \mathcal{A}_1 \vdash V'_i = W'_j \Rightarrow \forall j \exists i \mathcal{A}_1 \vdash \alpha_i.V'_i = \beta_j.W'_j$. Thus $\mathcal{A}_1 \vdash \sum_i^k \alpha_i.V'_i = \sum_j^l \beta_j.W'_j \Rightarrow \mathcal{A}_1 \vdash V = W$

4 Cost model

Parallel programming researchers commonly refer to a parallel performance model as a *cost model* and we will use this terminology. Following BSP, we will define a cost model where execution time is related to the number of steps of local processes and the estimate of the time required for a global communication superstep. The result is a cost model which is conform to the operational semantics. Since processes are considered modulo bisimulation, it is natural to require that the cost is stable for this equivalence. The main consequence of this is that cost is oblivious to the existence of an alternate copy of any process. As for bisimulation $P+P \sim P$, this is a reasonable notion of parallel execution *time*: alternate copies of the same process do not add to the worst case time.

We have restricted our study to the CCS strong bisimulation relation $\sim$ and ignore the so-called *weak* bisimulation $\approx$. This is justified as follows in the context of a cost model (definitions refer to the rest of this section). Weak bisimulation would lead (by $\tau.P \approx P$) to $\mathcal{C}(\tau) = \mathbf{1}$ (for example, in the semi-ring [3] $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ $\mathcal{C}(\tau) = \mathbf{1} = 0$) which amounts to assigning the same cost to $\tau^{100000}.0$ and to $\tau.0$ while the first term represents a large number of internal synchronizations. That is not coherent with the use of cost as an estimate of execution time. Other motivations for dismissing weak bisimulation are given in [1] in the context of information flow analysis.

4.1 Definition of the semi-ring $\mathcal{S}$

4.1.1 Basics on semi-rings

Here are the axioms that an idempotent semi-ring $(\mathcal{S}, \oplus, \odot, \mathbf{0}, \mathbf{1})$ has to verify.

$$c \oplus d = d \oplus c \qquad c \odot (d \odot e) = (c \odot d) \odot e$$
$$c \oplus (d \oplus e) = (c \oplus d) \oplus e \qquad c \odot \mathbf{1} = \mathbf{1} \odot c = c$$
$$c \oplus c = c \qquad c \odot \mathbf{0} = \mathbf{0} \odot c = \mathbf{0}$$
$$c \oplus \mathbf{0} = c \qquad c \odot (d \oplus e) = (c \odot d) \oplus (c \odot e)$$
$$(d \oplus e) \odot c = (d \odot c) \oplus (e \odot c)$$

Here are examples of idempotent semi-rings:

- $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ which computes longest paths, that is to say worst-case time complexity.
- $(\mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0)$ which computes the earliest dead-lock time.
- $(\mathbb{B}\mathrm{ool}, \vee, \wedge, \mathrm{false}, \mathrm{true})$ which computes reachability in the transition system.
- $(\mathbb{P}(\texttt{Act}), \cup, \texttt{concat}, \emptyset, \epsilon)$ which computes trace language.

We could also introduce a semi-ring that will compute the maximal number of processors used by the process: $(\mathbb{N} \cup -\infty, \max, \lambda \mathrm{a@i.b@j} \mapsto \max(\mathrm{i},\mathrm{j}), \infty, \mathrm{a@0})$.

4.1.2 Definition of the semi-ring $\mathcal{S}$

We need to measure paths in the graphs which serve as semantics for CCS processes: the so-called *labelled transition systems* whose nodes are processes, edges transitions and edge labels are the events.

Graph (oriented)	Semi-ring properties
nodes	values are costs of sets of paths
edges	$\mathbf{1}$ is the cost of the empty path [] i.e. $\{[]\}$
labels	$x \odot y$ is the cost of a concatenation of two sets of paths x, y
paths	$\mathbf{0}$ is the cost of the empty set of paths
sets of paths	$x \oplus y$ is the cost of the union of two sets of paths x, y

The indempotecd property $x \oplus x = x$ is required by the $P + P \sim P$ law of processes and our demand that bisimulation should respect costs.

As the cost is associated to the set of execution paths, it has to represent the barriers. A cost of a BSP program is unknown until the superstep is completed. The cost can be either a vector which represents the accumultated time on each processor before the barrier. Or it can be a triple which contains the information of the cost of a barrier and two adjacent asynchronous phases: the scalars represents the cost of the barrier, and the two vectors represent the cost of asynchronous computation before/after the barrier. A special kind of semi-ring is needed for this and we will define it in two steps: from $\mathcal{S}$ to $\mathcal{S}^p$ and then to $\mathcal{S}$.

First, we define the monoid $S^p_{MULT} = (\mathcal{S}^p \cup (\mathcal{S}^p \times \mathcal{S} \times \mathcal{S}^p), \odot, \mathbf{1})$ where $\mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ and where $\odot$ applies to vectors point to point.

Remark 3 *The product sets $\mathcal{S}^p$ represents the set of p processors in a BSP machine. To simplify our presentation we assume here a single value for p. In the presence of multiple BSP machines, we would use explicit reference to multiple p values. To be exact, the index $<>_p$ should be the name and the size of a BSP machine so as to distinguish two machines of the same size.*

A triple of $\mathcal{S}^p \times \mathcal{S} \times \mathcal{S}^p$ represents (the cost of) a suffix of a computation superstep followed by (the cost of) a communication superstep, and finally (the cost of) a prefix of a computation superstep.

Definition 8 *Let* $v = \begin{pmatrix} v_0 \\ \vdots \\ v_{p-1} \end{pmatrix}$ *then* $\sum v = v_0 \oplus \ldots \oplus v_{p-1}$.

Definition 9 *We define the operation $\odot$ on this monoid. v, w, v' and w' are vectors. x and x' are scalars.*

$\odot$	$\vec{v'}$	$(\vec{v'}, x', \vec{w'})$
$\vec{v}$	$\vec{v} \odot \vec{v'}$	$(\vec{v} \odot \vec{v'}, x', \vec{w'})$
$(\vec{v}, x, \vec{w})$	$(\vec{v}, x, \vec{w} \odot \vec{v'})$	$(\vec{v}, x \odot \sum(\vec{w} \odot \vec{v'}) \odot x', \vec{w'})$

As two vectors are concatened, they are simply concatened point to point. As a triple is followed by a vector, the vector is concatened point to point to the last vector of the triple. As a vector is followed by a triple, the vector is concatened point to point to the triple's fisrt vector. And when a triple is followed by a triple, that is to say between two barriers in our model, the two central vectors are concatened and transformed into a scalar (thanks to the $\sum$ operation), to be concatened to the scalars of the triples.

Proposition 12 $\odot$ *is associative.*

Proof :

- $v_1 \odot (v_2 \odot v_3) = (v_1 \odot v_2) \odot v_3$ because of the associativity in $\mathcal{S}^p$
- $v_1 \odot (v_2 \odot (v_3, x_3, w_3))$
  $= v_1 \odot (v_2 \odot v_3, x_3, w_3)$
  $= (v_1 \odot (v_2 \odot v_3), x_3, w_3)$
  $= ((v_1 \odot v_2) \odot v_3, x_3, w_3)$
  $= (v_1 \odot v_2) \odot (v_3, x_3, w_3)$
- $v_1 \odot ((v_2, x_2, w_2) \odot v_3)$
  $= v_1 \odot (v_2, x_2, w_2 \odot v_3)$
  $= (v_1 \odot v_2, x_2, w_2 \odot v_3)$
  $= (v_1 \odot v_2, x_2, w_2) \odot v_3$
  $= (v_1 \odot (v_2, x_2, w_2)) \odot v_3$
- $v_1 \odot ((v_2, x_2, w_2) \odot (v_3, x_3, w_3))$
  $= v_1 \odot (v_2, x_2 \odot \sum(w_2 \odot v_3) \odot x_3, w_3)$
  $= (v_1 \odot v_2, x_2 \odot \sum(w_2 \odot v_3) \odot x_3, w_3)$
  $= (v_1 \odot v_2, x_2, w_2) \odot (v_3, x_3, w_3)$
  $= (v_1 \odot (v_2, x_2, w_2)) \odot (v_3, x_3, w_3)$
- $(v_1, x_1, w_1) \odot (v_2 \odot v_3)$
  $= (v_1, x_1, w_1 \odot (v_2 \odot v_3))$
  $= (v_1, x_1, (w_1 \odot v_2) \odot v_3)$
  $= ((v_1, x_1, (w_1 \odot v_2)) \odot v_3)$
  $= ((v_1, x_1, w_1) \odot v_2) \odot v_3$
- $(v_1, x_1, w_1) \odot (v_2 \odot (v_3, x_3, w_3))$
  $= (v_1, x_1, w_1) \odot (v_2 \odot v_3, x_3, w_3)$
  $= (v_1, x_1 \odot \sum(w_1 \odot (v_2 \odot v_3)) \odot x_3, w_3)$
  $= (v_1, x_1 \odot \sum((w_1 \odot v_2) \odot v_3) \odot x_3, w_3)$
  $= (v_1, x_1, w_1 \odot v_2) \odot (v_3, x_3, w_3)$
  $= ((v_1, x_1, w_1) \odot v_2) \odot (v_3, x_3, w_3)$
- $(v_1, x_1, w_1) \odot ((v_2, x_2, w_2) \odot v_3)$
  $= (v_1, x_1, w_1) \odot (v_2, x_2, w_2 \odot v_3)$
  $= (v_1, x_1 \odot \sum(w_1 \odot v_2) \odot x_2, w_2 \odot v_3)$
  $= (v_1, x_1 \odot \sum(w_1 \odot v_2) \odot x_2, w_2) \odot v_3$
  $= ((v_1, x_1, w_1) \odot (v_2, x_2, w_2)) \odot v_3$
- $(v_1, x_1, w_1) \odot ((v_2, x_2, w_2) \odot (v_3, x_3, w_3))$
  $= (v_1, x_1, w_1) \odot (v_2, x_2 \odot \sum(w_2 \odot v_3) \odot x_3, w_3)$
  $= (v_1, x_1 \odot \sum(w_1 \odot v_2) \odot (x_2 \odot \sum(w_2 \odot v_3) \odot x_3), w_3)$
  $= (v_1, (x_1 \odot \sum(w_1 \odot v_2) \odot x_2) \odot \sum(w_2 \odot v_3) \odot x_3, w_3)$
  $= ((v_1, x_1 \odot \sum(w_1 \odot v_2) \odot x_2, w_2)) \odot (v_3, x_3, w_3)$
  $= ((v_1, x_1, w_1) \odot (v_2, x_2, w_2)) \odot (v_3, x_3, w_3)$

We have thus an associative monoid because $\mathbf{1}$ is easily verified to be a unit for $\odot$.

From this associative monoid, we then build the semi-ring

$$\$ = (\mathbb{P}_{\mathrm{FIN}}(\mathrm{S}^{\mathrm{p}}_{\mathrm{MULT}}), \cup, \odot, \mathbf{0}, \mathbf{1})$$

of finite sets of elements from the previous one, where $\mathbf{0} = \emptyset$ and $\mathbf{1} = \{\mathbf{1}\}$. Union on sets is associative, commutative and idempotent. For simplicity we will often identify singleton sets with their unique elements as for $\prime$ and ∞ above.

The next step is to prove the distributivity of $\odot$ over $\oplus$ when $\odot$ is lifted to sets of values (vectors or triples), which is trivial, thanks to the property of $\cup$.

This structure is where the cost of processes will be evaluated. For a given base idempotent semi-ring $\mathcal{S}^p$ there is a unique "BSP" idempotent semi-ring $\$$.

4.2 Associating costs to processes

Our costs are evaluated on serial form processes. We assume that restriction, recursion and parallel composition have been eliminated from the given process.

We will define $\mathcal{C} : \mathcal{E}_{FIN} \to \$$ in two steps :

1. We eliminate $\backslash$ and $|$ from terms of $\mathcal{E}_{FIN}$ by applying the expansion law 5 of section 3. We also assign basic costs $\mathcal{C}(\alpha)$ to events α.

2. The remaining operators are then `let in`, ., + and 0 with variables. Property $\mathcal{C}$ is thus verified with rules 4.2.1 where $P \rhd c$ means that $\mathcal{C}(P)$ is evaluated to semi-ring element c.

Given the variability of network traffic in grid applications, the BSP constants g, and L could here be considered as random variables.

By construction, this theory of costs is able to express both the interleaving and simultaneous occurrence of events.

4.2.1 Local rules

As we apply the expansion law, we only need the rules for the Prefix, Summation and Relabelling.

$$\mathcal{C}(0) = \mathbf{0}$$

$$\texttt{ACT}\frac{P \rhd c}{\alpha.P \rhd \mathcal{C}(\alpha) \odot c}$$

$$\texttt{SUM}\frac{P \rhd c \quad P \rhd c'}{P + Q \rhd c \oplus c'}$$

$$\texttt{LET}\frac{Q[X \leftarrow P] \rhd c}{\texttt{let } X = P \texttt{ in } Q \rhd c}$$

4.2.2 Use of Milner's action system $\mathcal{A}_1$ ([6]) to prove stability for local terms

We wish to have $P \sim Q \Rightarrow \mathcal{C}(P) = \mathcal{C}(Q)$, which will only hold if the semi-ring $\mathcal{S}$ is idempotent : $P + P \sim P \Rightarrow \mathcal{C}(P) \oplus \mathcal{C}(P) = \mathcal{C}(P)$ [1].

Lemma 6 (Stability) *For P and Q serial agents, $P \sim Q \Rightarrow \mathcal{C}(P) = \mathcal{C}(Q)$*

Proof : If $P \sim Q$, $\exists P'$ and $\exists Q'$ such that P' is the standard form of P and Q' is the standard form of Q. We then have $\mathcal{A}_1 \vdash P' = Q'$. It is then sufficient to verify the equality of costs for P' and Q' through equivalence in $\mathcal{A}_1$. To simplify, $\mathcal{C}(P)$ will be noted p, $\oplus$ + and $\odot$ ..

[1] Or more accurately: if the substructure $\mathcal{C}(\$)$ is idempotent.

1. $P + Q = Q + P$

   $\mathcal{C}(P + Q) = p + q$ by rule `SUM` and $\mathcal{C}(Q + P) = q + p$ by rule `SUM`
   As $+$ is commutative $p + q = q + p$.
   Therefore $\mathcal{C}(P + Q) = \mathcal{C}(Q + P)$.

2. $P + (Q + R) = (P + Q) + R$

   $\mathcal{C}(P + (Q + R)) = p + (q + r)$ by rule `SUM` and
   $\mathcal{C}((P + Q) + R) = (p + q) + r$ by rule `SUM`
   $p + (q + r) = p + q + r = (p + q) + r$ by the associativity of $+$
   Thus $\mathcal{C}(P + (Q + R)) = \mathcal{C}((P + Q) + R)$.

3. $P + P = P$

   $\mathcal{C}(P + P) = p + p$ by rule `SUM`
   $p + p = p$ by idempotence of $+$
   Thus $\mathcal{C}(P + P) = \mathcal{C}(P)$.

4. $P + 0 = P$

   $\mathcal{C}(P + 0) = p + \mathbf{0}$ by rule `SUM`
   $p + \mathbf{0} = p$ by neutrality of $\mathbf{0}$
   Thus $\mathcal{C}(P + 0) = \mathcal{C}(P)$.

4.2.3 Global rules

$\mathcal{C}$ is now extended to a function $\not\mathcal{C}$ from all serial terms to $\not\mathcal{S}$ [4.1.2]. Serial global terms V, obtained by the expansion laws 9, 10, contain neither $< ... >$, nor $|$, but only ., +, 0 and `let in`. They may contain occurrences of $T_{\sigma,\pi}$ whose cost expresses the BSP execution time for a communication and synchronization barrier, and occurrences of asynchronous events $< \alpha@i >_p$ whose costs commute when they are not co-located (see the different cases of $\not\mathcal{C}$ definition 10 below).

$$\not\mathcal{C}(0) = \mathbf{0} \quad \text{(i.e.}\{\}\text{)}$$

$$\texttt{ACT}\frac{E \rhd c}{\alpha.E \rhd \not\mathcal{C}(\alpha) \odot c} \text{ where } \alpha \in \texttt{Act}(\mathcal{E}_v)$$

$$\texttt{SUM}\frac{V_i \rhd c \quad V_j \rhd c'}{V_i + V_j \rhd c \oplus c'}$$

$$\texttt{LET}\frac{U[X \leftarrow V] \rhd c}{\texttt{let } X = V \texttt{ in } U \rhd c}$$

Definition 10 $\not\mathcal{C}: \mathcal{E}_v \to \not\mathcal{S}$

1. *$\not\mathcal{C}(< \alpha@i >_p) = \vec{1}[i \mapsto \mathcal{C}(\alpha)]$*

2. *$\not\mathcal{C}(\alpha) = (\vec{1}, C(\alpha), \vec{1})$*

3. *$\not\mathcal{C}(T_{\sigma,\pi}) = (\vec{1}, (\sum h_i^+ \oplus \sum h_i^-)^g \odot L, \vec{1})$ where $\vec{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$, $h_i^+ = \sum_{j=0}^{p-1} |\sigma_{ij}|$ and $h_i^- = \sum_{k=0}^{p-1} |\sigma_{ki}|$ where $|\sigma|$ is the syntactic size of substitution σ and the exponent $\_\_^g$ denotes repeated $\odot$ product (without loss of generality g is an integer).*

4.2.4 Costs respect bisimulation

Lemma 7 (Stability) *Stability for global finite serial terms V and W : $V \sim W \Rightarrow ₵(V) = ₵(W)$.*

Proof : As for property 6 on local terms, $V \sim W \Rightarrow \mathcal{A}_1 \vdash V = W$, and each application of an axiom leaves ₵ unchanged because rules for $₵(0)$ and $₵(V + W)$ translate the cost to $\oplus$ which satisfies associtivity, commutativity, idempotence and has a unit (idempotent semi-ring axioms for the sum).

4.3 Reducing cost to scalars : observation

We have built $\$$ a vectorial BSP semi-ring to represent faithfully the resource consomption of processes. This semi-ring has to represent especially interleaving. Even if $\mathcal{S}$ is totally ordered like for instance $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, there is no evident order between two computation costs in $\$$. We need an observer point of view to obtain the usual cost notion: a scalar for a set of vectorial costs. To this end we define $\mathcal{O}bs$ an observation function which transforms our element of $\$$ into a scalar of $\mathcal{S}$. And then, we have tools to compare the costs. However we will note that this observation is not compositional because it acts like a barrier: in general $\mathcal{O}bs$ does not preserve $\odot$.

Definition 11 $\mathcal{O}bs : \$ \rightarrow \mathcal{S}$
$\mathcal{O}bs(\vec{v}) = \sum \vec{v}$ *Vectors are flattened.*
$\mathcal{O}bs((\vec{v}, x, \vec{w})) = \sum \vec{v} \odot x \odot \sum \vec{w}$ *Triples are reduced : the two vectors are flattened and added to the central scalar.*

And this function is then lifted to sets (elements of $\$$) as follows :
$\mathcal{O}bs(\{X\}) = \mathcal{O}bs(X)$ *where X can either be $\vec{v}$ or $(\vec{v}, x, \vec{w})$*
$\mathcal{O}bs(A \cup B) = \mathcal{O}bs(A) \oplus \mathcal{O}bs(B)$

5 Examples

In this section $₵(\alpha) = a$, $₵(\beta) = b$... We use the semi-ring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ which computes a longest path in the transition system.

5.0.1 ASYNC

$$
\begin{aligned}
& ₵(< \alpha, \beta >) \\
&= ₵((\alpha@0.\beta@1) + (\beta@1.\alpha@0)) \\
&= ₵(\alpha@0.\beta@1) \oplus ₵(\beta@1.\alpha@0) \\
&= \left(\begin{array}{c} a \\ b \end{array}\right) \oplus \left(\begin{array}{c} a \\ b \end{array}\right) \\
&= \left(\begin{array}{c} a \\ b \end{array}\right) \ \mathcal{O}bs(\left(\begin{array}{c} a \\ b \end{array}\right)) = Max(a, b).
\end{aligned}
$$

5.0.2 ISOCHRONE with $p = 4$

This example refers to the figure 5.

First we compute the π_i and their application to P'_i.
$\pi_0 = id \circ (X \leftarrow \alpha.0) \circ (Y \leftarrow \gamma.0, X \leftarrow \delta.0) \circ id$
$\pi_0(P'_0) = \pi_0(X + Y) = \delta.0 + \gamma.0$
$\pi_1 = id \circ (X \leftarrow \gamma.0) \circ id \circ (X \leftarrow \alpha.0)$
$\pi_1(P'_1) = \pi_1(X) = \alpha.0$

$$\texttt{Let } V =$$
$$\texttt{<put}\begin{bmatrix} 1 \leftarrow [X \leftarrow \alpha] \\ 2 \leftarrow [X \leftarrow \beta] \end{bmatrix}.(X+Y), \texttt{put}\begin{bmatrix} 0 \leftarrow [Y \leftarrow \gamma, X \leftarrow \delta] \\ 3 \leftarrow [X \leftarrow \alpha] \end{bmatrix}.X, \texttt{put}\begin{bmatrix} 0 \leftarrow [X \leftarrow \alpha] \\ 1 \leftarrow [X \leftarrow \gamma] \\ 3 \leftarrow [Y \leftarrow \beta] \end{bmatrix}.X, \texttt{put}[].(X+Y)\texttt{>}$$

Figure 4: Example: ISOCHRONE

$\pi_2 = id \circ id \circ id \circ (X \leftarrow \beta.0)$
$\pi_2(P'_2) = \pi_2(X) = \beta.0$ $\pi_3 = id \circ (Y \leftarrow \beta.0) \circ (X \leftarrow \alpha.0) \circ id$
$\pi_3(P'_3) = \pi_3(X+Y) = \alpha.0 + \beta.0$

Then we need to compute the different h_i. $h_i^+ = \sum_{j=0}^{p-1} |\sigma_{ij}|$ and $h_i^- = \sum_{k=0}^{p-1} |\sigma_{ki}|$
$h_0^+ = |\sigma 00| + |\sigma 01| + |\sigma 02| + |\sigma 03| = |X \leftarrow \alpha.0| + |X \leftarrow \beta.0| = |\alpha| + |\beta|$
$h_0^- = |\sigma 00| + |\sigma 10| + |\sigma 20| + |\sigma 30| = |Y \leftarrow \gamma.0, X \leftarrow \delta.0| + |X \leftarrow a.0| = |\gamma| + |\delta| + |\alpha|$
$h_1^+ = |\sigma 10| + |\sigma 11| + |\sigma 12| + |\sigma 13| = |Y \leftarrow \gamma.0, X \leftarrow \delta.0| + |X \leftarrow \alpha.0| = |\gamma| + |\delta| + |\alpha|$
$h_1^- = |\sigma 01| + |\sigma 11| + |\sigma 21| + |\sigma 31| = |X \leftarrow a.0| + |X \leftarrow \gamma.0| = |\alpha| + |\gamma|$
$h_2^+ = |\sigma 20| + |\sigma 21| + |\sigma 22| + |\sigma 23| = |X \leftarrow \alpha.0| + |X \leftarrow \gamma.0| + |Y \leftarrow \beta.0| = |\alpha| + |\gamma| + |\beta|$
$h_2^- = |\sigma 02| + |\sigma 12| + |\sigma 22| + |\sigma 32| = |X \leftarrow \beta.0| = |\beta|$
$h_3^+ = |\sigma 30| + |\sigma 31| + |\sigma 32| + |\sigma 33| = 0$
$h_3^- = |\sigma 03| + |\sigma 13| + |\sigma 23| + |\sigma 33| = |X \leftarrow \alpha.0| + |Y \leftarrow \beta.0| = |\alpha| + |\beta|$
Let $|\alpha| = |\beta| = |\gamma| = |\delta| = t$ and $a = b = c = d = w$.
$Max\ \ h_i^+ = Max(|\alpha| + |\beta|, Max(|\gamma| + |\delta| + |\alpha|, |\alpha| + |\gamma| + |\beta|)) = 3t$
$Max\ \ h_i^- = Max(|\gamma| + |\delta| + |\alpha|, Max(|\alpha| + |\gamma|, Max(|\beta|, |\alpha| + |\beta|))) = 3t$
$\mathbb{C}(T_{\sigma,\pi}) = (\vec{1}, 3tg + L, \vec{1})$

$$\mathbb{C}(s) = \begin{pmatrix} Max(d,c) \\ a \\ b \\ Max(a,b) \end{pmatrix} = \begin{pmatrix} w \\ w \\ w \\ w \end{pmatrix}$$

$$\mathbb{C}(V) = \mathbb{C}(T_{\sigma,\pi}) \odot \mathbb{C}(*) = (\vec{1}, 3tg + L, \vec{1}) \odot \begin{pmatrix} w \\ w \\ w \\ w \end{pmatrix} = (\vec{1}, 3tg + L, \begin{pmatrix} w \\ w \\ w \\ w \end{pmatrix})$$

$\mathcal{O}bs(\mathbb{C}(V)) = 3tg + L + w.$

6 Application to meta-computing: scheduling

We now have a complete cost model. We can apply it to usual scheduling problems in data-grid computing.

6.1 Serializing

A typical situation is to schedule two data-parallel processes on the same parallel machine without interference and while maximizing performance.

We outline how our model can describe this situation, leading to the possibility of choosing the most efficient scheduling solution while ensuring a coherent overlaping of the two data-parallel processes.

We will use $p = 2$, but the example is easily extensible to larger p. Assume two data-parallel applications P and Q each one possibly executed on one or two processors. The problem is to find the fastest coherent scheduling of the two processes on a two-processor system. By coherent scheduling we mean one which does not mix internal communications from one data-parallel process with communications from the other. Other properties we could require and verify are: liveness, and absence of flow of information between the two data-parallel processes.

We use the semi-ring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$. For these examples, we introduce a new notation: $< P, ..., P >$ will be abreviated $< P : i = 0, .., p-1 >$.

Let $P =< b.\mathtt{put}[].b.\mathtt{put}[] : i = 0, ..., 1 >$, $\mathcal{O}bs(\mathcal{C}(P)) = 2t_b + 2L$ and $Q =< a.\mathtt{put}[\sigma].a.\mathtt{put}[\sigma] : i = 0, ..., 1 >$, $\mathcal{O}bs(\mathcal{C}(Q)) = 2t_a + 2L + 2gh$ where σ is symmetric and $\mathcal{C}(T_{\sigma,\pi}) = gh + L$.

As the control skeleton of a parallel program, P would be considered a compute bound process while Q is a communication bound process.

We enumerate possible schedulings depending on how many processors are used by each process.

1. P and Q are distributed over both processor. If we naively use $P_I =< P_0|Q_0, P_1|Q_1 >=< b.\mathtt{put}[].b.\mathtt{put}[]|a.\mathtt{put}[\sigma].a.\mathtt{put}[\sigma] : i = 0, .., 1 >$, then P_0 may realize a barrier with Q_1 thus leading to incoherent (unwanted) communication. As a `put` can be captured by any other `put` it is useful to introduce a function $[\_,\_]$ to schedule local sub-processes. Without loss of generality, we suppose that variables in P and Q are disjoint. $[\_,\_] : Proc^2 \to Proc$ is defined as follows:
   $[A, 0] = A$ and $[0, B] = B$
   $[a.A, b.B] = a.[A, b.B] + b.[a.A, B]$ where $a, b \neq \text{put}$
   $[A + A', B] = [A, B] + [A', B]$ and $[A, B + B'] = [A, B] + [A, B']$
   $[\mathtt{put}.A, b.B] = b.[\mathtt{put}.A, B]$ and $[a.A, \mathtt{put}.B] = a.[A, \mathtt{put}.B]$
   $[\mathtt{put}[\sigma_0].A, \mathtt{put}[\sigma_1].B] = \mathtt{put}[\sigma_0].\mathtt{put}[\sigma_1].[A, B]$

   Then we can rewrite P_I in
   $P_I' =< [P_0, Q_0], [P_1, Q_1] >$
   $=<$ `let` $X = \mathtt{put}.\mathtt{put}[\sigma].0$ `in`
   `let` $Y = a.b.X + b.a.X$ `in`
   `let` $Z = \mathtt{put}[].\mathtt{put}[\sigma].Y$ `in`
   $a.b.Z + b.a.Z : i = 0, .., 1 >$

   $\mathcal{C}(P_I') = (\begin{pmatrix} t_a + t_b \\ t_a + t_b \end{pmatrix}, t_a + t_b + 4gh + 4L, \vec{1})$. $\mathcal{O}bs(\mathcal{C}(P_I')) = 2t_a + 2t_b + 4gh + 4L$.

2. P is *serialized* on one processor $< P_{ser}|Q_0, Q_1 >= b.b.\tau.b.b.\tau$ and Q is distributed over both processors. $P_{II} =< P_{ser}|Q_0, Q_1 >=$
   $< b.b.\tau.b.b.\tau|a.\mathtt{put}[\sigma].a.\mathtt{put}[\sigma], a.\mathtt{put}[\sigma].a.\mathtt{put}[\sigma] >$

   $\mathcal{O}bs(\mathcal{C}(P_{II})) = 2t_a + 4t_b + 4gh + 2L$

3. Same as the P_{II} where Q is *serialized* on one processor $Q_{ser} = a.a.\sigma.a.a.\sigma$ where c_σ is a local action realizing the same substitution (σ) as `put[`σ`]`. Its cost is ph instead of pgh (locally, the g factor costs 1). P is distributed over both processors.
   $P_{III} =< P_0, P_1|Q_{ser} >=< b.\mathtt{put}[].b.\mathtt{put}[], b.\mathtt{put}[].b.\mathtt{put}[]|a.a.c_\sigma.a.a.c_\sigma >$

   $\mathcal{O}bs(\mathcal{C}(P_{III})) = 4t_a + 2t_b + 4h + 2L$

4. Each process is *serialized* on one processor.
   $P_{IV} =< P_{ser}, Q_{ser} >= b.b.\tau.b.b.\tau, a.a.c_\sigma.a.a.c_\sigma >$

   $\mathcal{O}bs(\mathcal{C}(P_{IV})) = max(4t_b, 4t_a + 4h)$.

From the above cost estimates it is then possible to select scheduling with least execution time which depend on the numerical values of g and L. Although symbolically the minimum time must be either P_I' or P_{IV} since $P_I' \leq P_{II}$ and $P_{IV} \leq P_{III}$. It is clear from this example that solutions to large-scale versions of this problem can be computed mechanically.

6.2 Substitution passing

Another typical meta-computing problem is to distribute a process like $P =< P_0, ..., P_3 >$ on two 2-processor systems, while preventing blocking and minimizing execution time. The resulting process is the of the form $< P_0, P_1 > | < P_2, P_3 >$ and should realize a 4-process barrier by two 2-process barriers and some point-to-point synchronizations (rule `PARV`) between the two systems. A technical problem is to send messages from one system to the other via the point-to-point synchronization. This can be done by `let in` abstractions of the initial `put`[*sigma*] substitutions. But that would not faithfully represent the two 2-prcess systems because such terms are flat (without data-parallel compositions). Another possibility would be to define a “mobility” extension ([8, 2]) of our process algebra where processes could migrate and thus carry data. But this would lead to a more complex formalism and we leave it to future work where mobility would be part of the application area (e.g. pervasive computing) rather than a technical device in the formal model. We propose instead a simpler solution: to introduce a substitution passing rule which generalizes both CCS value-passing and our BSP communications.

6.2.1 Definitions

Here are listed the changes to BSPA due to the addition of substitution passing. Everything else in the algebra is unchanged with respect to sections [2,3,4].

$\mathcal{L}_V$ unchanged, but $\bar{\mathcal{A}} = \{\bar{\alpha} | \lambda \in Subst, \bar{\alpha}$ `is a local output label`$\}$ and $\bar{\mathcal{A}}_V = \bar{\mathcal{A}} \cup \{< \bar{\alpha}(\lambda)@i > | \lambda \in Subst, \bar{\alpha} \in \bar{\mathcal{A}}, 0 \leq i \leq p-1\}$

Remark : $\bar{\alpha}$ is the abbreviate for $\bar{\alpha}([])$.

$\texttt{Act}_V ::= \alpha \mid T_{\sigma,\pi} \mid < \tau@ij >_p \mid \tau_{Net}$ where $\alpha \in \mathcal{L}_V$

Remark 4 *There is no need for bindings as in value-passing CCS : $a(x)$ because the reception by a of a substitution such as $\bar{a}(\lambda)$ includes target variables.*

6.2.2 Rules

Only some global rules get some changes.

$$\texttt{RES}\frac{V \stackrel{\alpha}{\rightarrow} V'}{V\backslash\beta \stackrel{\alpha}{\rightarrow} V'\backslash\beta}(\alpha \neq \{\beta, \bar{\beta}, \bar{\beta}(\lambda)\}$$

$$\texttt{PARV}\frac{V_0 \stackrel{\alpha}{\rightarrow} V_0' \quad V_1 \stackrel{\bar{\alpha}(\lambda)}{\rightarrow} V_1'}{V_0|V_1 \stackrel{\tau_{Net}(\lambda)}{\rightarrow} \lambda(V_0')|V_1'}$$

$$\texttt{DUOCHRONE}\frac{P_i \stackrel{\alpha}{\rightarrow} P_i' \quad P_j \stackrel{\bar{\alpha}(\lambda)}{\rightarrow} P_j' \quad i \neq j}{<P_0,...,P_i,...,P_j...,P_{p-1}> \stackrel{<\tau(\lambda)@ij>_p}{\rightarrow} <P_0,...,\lambda(P_i'),...,P_j'...,P_{p-1}>}$$

6.2.3 Examples

- ASYN+PARV

$$\frac{< 0, b.X > \stackrel{<b@1>}{\rightarrow} < 0, X > \quad < \bar{b}(X \leftarrow a).0, 0 > \stackrel{<\bar{b}(X \leftarrow a)@0>}{\rightarrow} < 0, 0 >}{< 0, b.X > | < \bar{b}(X \leftarrow a).0, 0 > \stackrel{\tau_{Net}(X \leftarrow a)}{\rightarrow} < 0, a > | < 0, 0 >}$$

This example illustrates a point to point synchronization.

- DUO

$$\frac{a.X \overset{a}{\rightarrow} X \quad \bar{a}(X \leftarrow b).0 \overset{\bar{a}(X\leftarrow b)}{\rightarrow} 0}{< a.X, \bar{a}(X \leftarrow b).0 > \overset{\tau@01}{\rightarrow} < b, 0 >}$$

This exampls illustrates a synchronization within a system.

6.2.4 Bisimulation

All the proofs stay correct because the modifications involved the rules PARV and DUOCHRONE which are eliminated in serial terms. The expansion laws are essentially the same as before, with the addition of substitutions to certain actions.

6.2.5 Cost

The cost concerns only the ACT, SUM and LET rules (local and global), thus the cost rules are not to be modified. However, the cost of some transitions in which the substitutions are added, have to be defined. Thus, in definition [10] $\mathcal{C}: \mathcal{E}_v \rightarrow \$$ we add :

4. $\mathcal{C}(\tau_{Net}(\lambda)) = (\vec{1}, |\lambda|^{g}_{Net} \odot L_{Net}, \vec{1})\ L_{Net}, g_{Net}$ because such a synchronization travels between parallel systems (as over the internet) where latency and bandwidth are different than inside the parallel machines.
5. $\mathcal{C}(\tau(\lambda)@ij) = (\vec{1}, |\lambda|^{g} \odot L, \vec{1})$ where the L, g parameters are indeed those of the enclosing parallel system.

6.2.6 Application to scheduling

Let $P =<$ `put` $\begin{bmatrix} 0 \leftarrow \lambda_{i0}, \\ 1 \leftarrow \lambda_{i1}, \\ 2 \leftarrow \lambda_{i2}, \\ 3 \leftarrow \lambda_{i3} \end{bmatrix} .P_i | i = 0, .., 3 >$ be the process we want to distribute. The first step is to transform the λ_{ij} from the barrier in point to point synchronization. We then create α_{ij} and corresponding $\bar{\alpha}_{ij}(\lambda_{ij}$ on each side of the two systems. Thus the exchanges can be done as it should be in a 4-processors machine. This example is shown is figure 5.

We assume that all λ have the same cost t_λ. The cost of the usual synchronisation of a 4-process parallel machine is $(\vec{1}, 4t_\lambda g + L, \vec{1})$. The cost of the synchronization transformed in two 2-process parallel machine is $(\vec{1}, 8t_\lambda g_{Net} + 8L_{Net} + 2t_\lambda g + L, \vec{1})$. This demonstrate that in cases where g_{Net} is less than the quarter of g and L_{Net} is very small, it would be interesting to distribute.

7 Conclusion

This study highlights the fact that the CCS "interleaving" semantics can be reconciled with the data-parallel (hence BSP) notion of simultaneous actions. Moreover, a generic cost model can be designed to generalize both a standard notion of paths in CCS transition systems and the BSP notions of simultaneous parallel execution with communication and synchronization barriers.

Krishnan's distributed CCS model [4] and Rebeuf's asynchronous performance model [9] allow arbitrarily complex communications without taking advantage of barriers which prevent combinatorial explosion either in process algebraic calculations or in cost calculations. We have therefore favored the BSP point of view on parallel computation.

$$< \texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{00},\\ 1 \leftarrow \lambda_{01},\\ 2 \leftarrow \lambda_{02},\\ 3 \leftarrow \lambda_{03}\end{array}\right].P_0, \texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{10},\\ 1 \leftarrow \lambda_{11},\\ 2 \leftarrow \lambda_{12},\\ 3 \leftarrow \lambda_{13}\end{array}\right].P_1, \texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{20},\\ 1 \leftarrow \lambda_{21},\\ 2 \leftarrow \lambda_{22},\\ 3 \leftarrow \lambda_{23}\end{array}\right].P_2, \texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{30},\\ 1 \leftarrow \lambda_{31},\\ 2 \leftarrow \lambda_{32},\\ 3 \leftarrow \lambda_{33}\end{array}\right].P_3 >$$

$$\Downarrow \texttt{Transformation to two } 2 - \texttt{process machines} \Downarrow$$

$$< \bar{a}_{30}(\lambda_{30}).\bar{a}_{20}(\lambda_{20}).a_{20}.a_{30}.\texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{00},\\ 1 \leftarrow \lambda_{01}\end{array}\right].P_0, \quad \bar{a}_{13}(\lambda_{13}).\bar{a}_{12}(\lambda_{12}).a_{31}.a_{21}.\texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{10},\\ 1 \leftarrow \lambda_{11}\end{array}\right].P_1 > |$$

$$< a_{02}.a_{12}.\bar{a}_{30}(\lambda_{30}).\bar{a}_{21}(\lambda_{21}).\texttt{put}\left[\begin{array}{l}2 \leftarrow \lambda_{22},\\ 3 \leftarrow \lambda_{23}\end{array}\right].P_2, \quad a_{13}.a_{03}.\bar{a}_{31}(\lambda_{31}).\bar{a}_{30}(\lambda_{30}).\texttt{put}\left[\begin{array}{l}2 \leftarrow \lambda_{32},\\ 3 \leftarrow \lambda_{33}\end{array}\right].P_3 >$$

$$\Downarrow \texttt{Point to point synchronizations} \Downarrow$$

$$< \lambda_{20}(\lambda_{30}(\texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{00},\\ 1 \leftarrow \lambda_{01}\end{array}\right].P_0)), \lambda_{31}(\lambda_{21}(\texttt{put}\left[\begin{array}{l}0 \leftarrow \lambda_{10},\\ 1 \leftarrow \lambda_{11}\end{array}\right].P_1)) > | < \lambda_{02}(\lambda_{12}(\texttt{put}\left[\begin{array}{l}2 \leftarrow \lambda_{22},\\ 3 \leftarrow \lambda_{23}\end{array}\right].P_2)), \lambda_{13}(\lambda_{03}(\texttt{put}\left[\begin{array}{l}2 \leftarrow \lambda_{32},\\ 3 \leftarrow \lambda_{33}\end{array}\right].P_3)) >$$

$$\Downarrow \texttt{Barrier} \Downarrow$$

$$< \lambda_{20}(\lambda_{30}(\lambda_{10}(\lambda_{00}(P_0)))), \lambda_{31}(\lambda_{21}(\lambda_{11}(\lambda_{01}(P_1)))) > | < \lambda_{02}(\lambda_{12}(\lambda_{32}(\lambda_{22}(P_2)))), \lambda_{13}(\lambda_{03}(\lambda_{33}(\lambda_{23}(P_3)))) >$$

$$\Downarrow \vdots \Downarrow$$

Figure 5: Distribution on two systems

An important open technical problem is to define a formal framework to data-parallelize all terms. By this we mean the "localization" of local actions not part of any parallel vector. This ensures an explicit data placement for all processes, something not required by the theory of CCS. It remains to define the arithmetic of mixtures of "localized" actions (through the exchanges defined by rules `ASYNC`, `DUOCHRONE` and `ISOCHRONE`) and the parallel actions ($\mathtt{COM}_0$, $\mathtt{COM}_1$ and $\mathtt{COM}_2$).

Our model's application to realistic studies will require: a symbolic treatment of the width of data-parallel vectors, of individual machine identifiers and the design of unification or constraint-solving algorithms. We will then be able to apply our formalism to the desing of correct, safe and optimal scalable meta-computing processes, in particular through automatic analysis tools like model-checkers. Other extensions we are considering are mobility ([8, 2]) and its application to security properties ([1]) in the context of parallel systems. This last problem is the object of a current project between LIFO and CEA and the formalism developed here will give a realistic modelization of data-parallel distribution and performance.

References

[1] S. Anantharaman and G. Hains. A synchronous bisimulation-based approach for information flow analysis. In *Third Workshop on Automated Verification of Critical Systems: (AVOCS'03)*, April 2003.

[2] G. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *23rd annual ACM Symposium on Principles of Programming Languages (POPL'96)*, pages 371–385, St.Petersburg (FLA) USA, January 1996.

[3] M. Gondran and M. Minoux. *Graphes et Algorithmes*, chapitre 3. Eyrolles, 1985. Les Algèbres de Chemins.

[4] P. Krishnan. Distributed CCS. In *Theories of Concurrency: Unification and Extension: CONCUR-91, LNCS:527*, pages 393–407. Springer, August 1991.

[5] F. Loulergue, G. Hains, and C. Foisy. A calculus of functional BSP programs. *Science of Computer Programming*, 37(1–3):253–277, May 2000.

[6] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[7] R. Milner. Operational and algebraic semantics of concurrent processes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North-Holland, MIT-Press, 1990.

[8] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 and 41–77, September 1992.

[9] X. Rebeuf. *Un modèle de coût symbolique pour les programmes parallèles asynchrones à dépendances structurées*. Thèse de Doctorat d'Université, Université d'Orléans, décembre 2000.

[10] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103, August 1990.