A Realistic Model for True Concurrency

M. Carmen Ruiz, Fernando Cuartero, Diego Cazorla, J. José Pardo

Departamento de Informática
Escuela Politécnica de Albacete
Univ. de Castilla-La Mancha
Campus Univ. s/n. 02071. Albacete, Spain.
ph: +34 967 599200
fax +34 967 599224
{carmenr, fernando, dcazorla, jpardo}@info-ab.uclm.es

Abstract. A Timed Process Algebra which takes into account the number of resources/processors the processes have at disposal is presented. Both its syntax and its operational semantics are defined. An algorithm which allows us to estimate required time to evolve between states and which helps us with performance evaluation is also presented. A simple example is used to illustrate how a system can be specified by means of our language and how the performance evaluation algorithm works. We finish by outlining the need of our algebra and what it can be used for.

1 Introduction

It is a well-known fact that concurrency appearance involves a radical increase in the complexity of a system. Theoretical models developed to study these systems at a formal level have recently become of great importance. As a result, the first models abstract many real properties of concurrent systems which are considered non essential for them, such as durations and structure of actions, properties of communication networks, distribution of system components, number of processors and so on. Leaving these non essential properties aside, it is easier to study the behaviour of processes at a formal level due to clear design specification but, of course, these simplifications have a cost which is the loss of all the information which has been abstracted.

This fact can be observed in early formal models such as *Petri Nets* and *process algebras*, such as CSP, CCS, ACP and EPL.

Once the basic models have been studied in depth, it is necessary to extend them to study these aspects, usually quantitative ones, which have been lost in the abstraction. Extended models have been proposed to represent these aspects such as fairness [CS87, RZ92], timed models [MT90, RR88], Markovian timed models [HHK02, Hil96], probabilistic models [GJS90, NdFL95, CdFV97], interconnection network properties [Zic90, GMS94] or processes distribution [CN94].

An abstraction of a system is to consider it as a number of concurrent processes. In many real systems this abstraction will be realised by executing the processes on a shared processor where execution is switched among the processes.

For such a system, the fact that processes have to share the processor cannot be ignored, since it may influence both the time and the functional behaviour of the system. Systems of processes sharing a common processor are well described in the theory of scheduling, see e.g. [Liu00]. However, these theories generally abstract from the actual behaviour of processes and hence cannot be used to analyse functional properties of a system. On the other hand, operational models such as automata and process algebras may describe the functional behaviour of a system in detail. Today, extensions of these models with a notion of time are well-established, but they are usually based upon an assumption of indefinite computational resources being available.

Recently, some authors have studied process scheduling in the context of timed automata [AGS00, FPY02]. These approaches, however, seem focused on modelling scheduling and they have no direct way of combining their model of the process scheduling with a system model where the behaviour of processes is described in detail.

An alternative is to take a process algebraic approach in which systems, from a modelling point of view, can be specified by writing programs in a programming language. However, timed process algebras typically assume maximal parallelism, which may be interpreted as if each process had its own dedicated processor. Therefore, they cannot adequately model real-time systems where processes share a processor. Under the assumption of maximal parallelism, it may be said that the processor will always be available to a process. This means that the passing of time may be described without considering whether the processor is available or not.

The main aim of this paper is to present a process algebra which can be used in order to specify systems with real parallelism where the resources are shared by all the processes which are in execution in the system. The most common example is a multitasking system sharing one or several processors. However, the presented algebra allows specifications of more general systems. We assume any system consists of a number of processes which have to share one or more resources. By resource we mean any device needed for the running of a process. It might be a single processor, a hardware circuit, a bus, a software component and so on. The only requirement is that if there are more processes than resources then not all of them can be executed at the same moment.

The models of concurrency can be roughly divided into two groups: interleaving based models, in which the independent execution of two processes is modelled by specifying the possible interleaving of their actions; and true concurrency models, in which the causal relations between actions are represented explicitly. Our process algebra is based on the notion of true concurrency where concurrent behaviour can be distinguished from an interleaving one as follows: $a|b \neq a.b + b.a$.

As a basis of our algebra we take CSP because it is simple and yet powerful enough to express parallelism, non-determinism and synchronization, although the new features which have been introduced into it can be easily suited to other process algebras such as CCS, ACP or EPL. We have enriched CSP by extending and modifying its semantics so that it allows us to show the execution time for each action and to manage the parallelism taking into account the number of resources/processors at disposal. With these changes we achieve to express the concept of "resource consumption" which is well known to be more and more important every day and it is of huge importance in systems where the resource cost is essential.

This paper is organised as follows: First, we present the syntax of our process algebra and some basic notations. Second the operational semantics of the language is described by means of a set of rules. Building on it, we define an algorithm for time evaluation and illustrate it with a simple example of how a system can be specified and how the performance algorithm works. Then we present our conclusions and, as this paper presents work-in-progress, we conclude by outlining future work.

2 Syntax

Let $\mathcal{A}ct$ be a finite set of actions. The syntax of the language is defined by the following BNF expression:

$$P ::= \mathit{stop} \mid\ < a, \alpha > .P \mid P \ \oplus \ P \mid P \ + \ P \mid P \parallel_A P \mid P \backslash\, a \mid recX.P$$

where $A \subseteq \mathcal{A}ct$, $\mathcal{P} \in \mathit{Idf}$ (set of identifiers), $a \in \mathcal{A}ct$ and $\alpha \in \mathbb{N}$ where $\mathbb{N}$, as usual, represents the set of natural numbers.

Let us now briefly describe the informal and intuitive interpretation of these operators. Later we will provide them with an operational semantics, so the meaning for each operator and the relationship among them will have been formalised.

stop .- This represents a deadlock, i.e. no action can be executed.

Timed prefix. ($< a, \alpha > .P$).- The classical prefix operator has been enriched with information about the amount of time the action a takes in its execution (α). As usual, process $< a, \alpha > .P$, once the action a has been executed, behaves like process P.

Internal choice. ($P_1 \ \oplus \ P_2$).- This is essentially the classical internal choice operator. Given processes P_1 and P_2, $P_1 \ \oplus \ P_2$ is the process which can behave like P_1 or like P_2 as an internal decision of the system.

External choice. ($P_1 \ + \ P_2$).- This is also interpreted in the usual fashion. Given processes P_1 and P_2, $P_1 + P_2$ is the process which can behave like P_1 or like P_2 as the environment requests.

Parallel with synchronization. ($P_1 \parallel_A P_2$).- This represents the parallel execution of P_1 and P_2 synchronizing on the actions in A. We will denote by $P_1 \parallel P_2$ the parallel execution of P_1 and P_2 without synchronization, it means: $P_1 \parallel_{\{\}} P_2$.

Hiding. ($P \backslash a$).- All instances of action a in process P are hidden.

Recursion. ($recX.P$).- The classical recursion operator where occurrences of X are substituted by $recX.P$. This operator allows us to define infinite behaviour.

3 Notation

We will use the notation of multiset as follows:

$B = \{2.x_1,\ 3.x_2,\ 5.x_3\}$ is the multiset defined over the set $X = \{x_1,\ x_2,\ x_3,\ x_4\}$ by:

$$B : \{x_1,\ x_2,\ x_3,\ x_4\} \longrightarrow \mathbb{N}$$

$$B(x_1) = 2,\ B(x_2) = 3,\ B(x_3) = 5,\ B(x_4) = 0$$

Capitals B, B_1, B_2, C, C_1, C_2 will represent multisets, while A, A_1, A_2 will represent sets. Both the empty multiset and empty set will be represented by $\emptyset$ because it is clear in context.

We will also use the following operations:

- $B_1 \cup B_2$: Union of multisets.
- $B_1 \cap B_2$: Intersection of multisets.
- $B_1 - B_2$: Subtraction of multisets.
- $B_1 \subseteq B_2$: Inclusion for multisets.
- $|B|$: Number of actions in the multiset, i.e.:
  $|B| = \sum B(x), \qquad \forall x \in X.$
- $B \backslash A$: Multiset obtained removing from B the actions which belong to A, i.e.:
  $(B \backslash A)(x) = \begin{cases} 0 & \text{if } x \in A \\ B(x) & \text{otherwise} \end{cases}$
- $B_1 +_A B_2$: Multiset obtained by adding the instances of actions which do not belong to A, and taking the intersection for the ones belonging to A:
  $(B_1 +_A B_2)(x) = \begin{cases} B_1(x) \cup B_2(x) & \text{if } x \notin A \\ B_1(x) \cap B_2(x) & \text{otherwise} \end{cases}$
  Note that $B_1 +_A B_2 = ((B_1 \cup B_2)\backslash A) \cup ((B_1 \cap B_2) \cap A)$.
- $B_1 \lhd_A B_2$: Multiset that consists of actions in B_1 which belong to A and are not in B_2:
  $(B_1 \lhd_A B_2)(x) = \begin{cases} B_1(x) & \text{if } x \in A \cap (B_1 - B_2) \\ 0 & \text{otherwise} \end{cases}$

We call a bag of actions B as the multiset defined over the elements of Act.

Definition 1. *(Bag of timed actions) We define a bag of timed actions, TB, as a pair whose elements are defined in the following way*

$$(b, \alpha) \textit{ such that } b \in B \textit{ and } \alpha \in \mathbb{N}$$

where α is the duration of each action in the bag. The execution of each action belonging to the same bag takes the same amount of time to complete.

4 Operational semantics

Operational Semantics provides language operators with a meaning and accurate interpretation by describing how one process is able to turn into another. This evolution is shown by a labelled transition system.

Definition 2. *(Transition)*
A transition is a tuple (P, Q, TB) *which we denote as:*

$$P \stackrel{B,\alpha}{\longrightarrow} Q$$

and its meaning is the usual one, i.e., process P *executes actions belonging to the bag* B *and then it behaves like process* Q*. The execution of each action belonging to the same bag takes the same amount of time to complete, here* α*.*

In Tables 1, 2, 3, 4 and 5 we can find the rules for our language which we are going to explain in a intuitive way.

$$\textbf{R1)}\quad \frac{}{(a,\alpha).P \xrightarrow{\{a\},\alpha} P}$$

$$\textbf{R2a)}\quad \frac{P \xrightarrow{B,\alpha} Q}{P \oplus R \xrightarrow{B,\alpha} Q} \qquad \textbf{R2b)}\quad \frac{P \xrightarrow{B,\alpha} Q}{R \oplus P \xrightarrow{B,\alpha} Q}$$

$$\textbf{R3a)}\quad \frac{P \xrightarrow{B,\alpha} P'}{P + Q \xrightarrow{B,\alpha} P'} \qquad \textbf{R3b)}\quad \frac{Q \xrightarrow{B,\alpha} Q'}{P + Q \xrightarrow{B,\alpha} Q'}$$

Table 1. Operational Semantics: Timed Prefix, Internal and External Choice

We assume $P,\ Q,\ P',\ Q' \in Idf$; $a, b \in \mathcal{A}ct$; $\alpha_i \in \mathbb{N}$; $B_i \in \mathcal{P}(\mathcal{A}ct)$ and $A_i \subseteq \mathcal{A}ct$.

Rule $R1$ states that given a process P and an action a with an execution time α, firstly action a is executed and then it behaves like process P.

Rules $R2$ and $R3$ are the classical ones for the internal and external choice operators.

In following rules we have extended the Timed Prefix operator for bags of timed actions in the following way:

$$< B, \alpha > .P$$

which is not a syntactic process, but it is obtained as a result of the evolution of other processes and its intuitive description is that the process $< B, \alpha > .P$, once the actions of the bag B have been executed, behaves like process P.

The group of rules $R4$ defines how the system develops due to the parallel operator without synchronization. This evolution is also defined in rule $R5$ where semantics of the parallel operator with synchronization is established. If the set

R4a)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q'}{P \parallel Q \xrightarrow{B_1, \alpha_1} P' \parallel Q}$$

R4b)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q'}{P \parallel Q \xrightarrow{B_2, \alpha_2} P \parallel Q'}$$

R4c)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q' \;\wedge\; |B_1| + |B_2| \leq \mathcal{N} \;\wedge\; \alpha_1 < \alpha_2}{P \parallel Q \xrightarrow{B_1 \cup B_2, \alpha_1} P' \parallel (B_2, \alpha_2 - \alpha_1).Q'}$$

R4d)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q' \;\wedge\; |B_1| + |B_2| \leq \mathcal{N} \;\wedge\; \alpha_1 > \alpha_2}{P \parallel Q \xrightarrow{B_1 \cup B_2, \alpha_2} (B_1, \alpha_1 - \alpha_2).P' \parallel Q'}$$

R4e)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q' \;\wedge\; |B_1| + |B_2| \leq \mathcal{N} \;\wedge\; \alpha_1 = \alpha_2}{P \parallel Q \xrightarrow{B_1 \cup B_2, \alpha_1} P' \parallel Q'}$$

R4f)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q' \;\wedge\; |B_1| + |B_2| > \mathcal{N} \;\wedge\; \alpha_1 < \alpha_2}{P \parallel Q \xrightarrow{C, \alpha_1} (B_1 - C_1, \alpha_1).P' \parallel (B_2, \alpha_2 - \alpha_1).(B_2 - C_2, \alpha_1).Q'}$$

R4g)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q' \;\wedge\; |B_1| + |B_2| > \mathcal{N} \;\wedge\; \alpha_1 > \alpha_2}{P \parallel Q \xrightarrow{C, \alpha_1} (B_1, \alpha_1 - \alpha_2).(B_1 - C_1, \alpha_2).P' \parallel (B_2 - C_2, \alpha_2).Q'}$$

R4h)
$$\frac{P \xrightarrow{B_1, \alpha_1} P' \;\wedge\; Q \xrightarrow{B_2, \alpha_2} Q' \;\wedge\; |B_1| + |B_2| > \mathcal{N} \;\wedge\; \alpha_1 = \alpha_2}{P \parallel Q \xrightarrow{C, \alpha_1} (B_1 - C_1, \alpha_1).P' \parallel (B_2 - C_2, \alpha_1).Q'}$$

Where

$C = C_1 \cup C_2$, with $C_1 \subseteq B_1$, $C_2 \subseteq B_2$ and $|C| \leq \mathcal{N}$

$\mathcal{N}$ is a constant which bounds the number of resources at disposal

Table 2. Operational Semantics: Parallelism with synchronization

of synchronization is empty this rule defines the same behaviour as rule $R4$. However, we have chosen to include it in order to achieve a better understanding.

In these rules we consider $m_1 = |B_1|$, $m_2 = |B_2|$ and $\mathcal{N}$ representing the maximum number of threads/processes which can be executed in parallel. Rules $R4c$, $R4d$ and $R4e$ take into account the case in which the addition of actions of the bags belonging to the processes involved is smaller than the maximum number of threads which can be executed in parallel, i.e., $m_1 + m_2 \leq \mathcal{N}$. In this assumption, all the actions (from both bags) are executed for $min(\alpha_1, \alpha_2)$,

$$\textbf{R5a)}\quad \frac{P \xrightarrow{B_1,\alpha_1} P' \ \wedge\ Q \xrightarrow{B_2,\alpha_2} Q' \ \wedge\ \alpha_1 = \alpha_2}{P \parallel_A Q \xrightarrow{B,\alpha_1} (B_1 \lhd_A B_2, \alpha_1).P' \parallel_A (B_2 \lhd_A B_1, \alpha_2).Q'}$$

Where

$B = B_1 +_A B_2 \quad and \quad |B| \leq \mathcal{N}$

$$\textbf{R5b)}\quad \frac{P \xrightarrow{B_1,\alpha_1} P' \ \wedge\ Q \xrightarrow{B_2,\alpha_2} Q' \ \wedge\ \alpha_1 = \alpha_2}{P \parallel_A Q \xrightarrow{C,\alpha_1} (B_1 - C_1 - A_1, \alpha_1).P' \parallel_A (B_2 - C_2 - A_1, \alpha_2).Q'}$$

Where

$\Delta = \{\ C = C_1 \ \cup\ C_2 \ \cup\ A_1\}$ with

$C_1 \subseteq B_1 \backslash A \qquad A_1 \subseteq (B_1 \cap B_2) \cap A$

$C_2 \subseteq B_2 \backslash A \qquad |C| \leq \mathcal{N}$

Table 3. Operational Semantics: Parallelism with synchronization and $\alpha_1 = \alpha_2$

$$\textbf{R5c)}\quad \frac{P \xrightarrow{B_1,\alpha_1} P' \ \wedge\ Q \xrightarrow{B_2,\alpha_2} Q' \ \wedge\ \alpha_1 > \alpha_2}{P \parallel_A Q \xrightarrow{C,\alpha_1} (C_1 \cup A_1, \alpha_1 - \alpha_2).(B_1 - C_1 - A_1, \alpha_1).P' \parallel_A (A_1, \alpha_1 - \alpha_2).(B_2 - C_2 - A_1, \alpha_2).Q'}$$

Where

$\Delta = \{\ C = C_1 \ \cup\ C_2 \ \cup\ A_1\}$ with

$C_1 \subseteq B_1 \backslash A \qquad A_1 \subseteq (B_1 \cap B_2) \cap A$

$C_2 \subseteq B_2 \backslash A \qquad |C| \leq \mathcal{N}$

Table 4. Operational Semantics: Parallelism with synchronization and $\alpha_1 \neq \alpha_2$

$$\textbf{R6a)}\quad \frac{P \xrightarrow{B,\alpha} Q \quad B \in \mathcal{P}(\mathcal{A}ct)\ a \notin B}{P\backslash a \xrightarrow{B,\alpha} Q\backslash a} \qquad \textbf{R6b)}\quad \frac{P \xrightarrow{\{a\},\alpha_1} Q \ \wedge\ Q\backslash a \xrightarrow{B,\alpha_2} R}{P\backslash a \xrightarrow{B,\alpha_1+\alpha_2} R}$$

$$\textbf{R7a)}\quad \frac{P\{recX.P/X\} \xrightarrow{\{a\},\alpha} P'}{recX.P \xrightarrow{\{a\},\alpha} P'}$$

Table 5. Operational Semantics: Hiding and Recursion

then the process whose action bag has the shortest time finishes the execution of its actions in parallel while the other still has to execute its actions for the remaining time. When the time of both bags is equal, all the actions are executed and both processes finish their actions in parallel.

In rules $R4f$, $R4g$ and $R4h$ semantics for parallel operator without synchronization is captured when the addition of the actions of the bags belonging to the processes involved is bigger than the maximum number of threads which can be executed in parallel, i.e.,

$$B = B_1 \cup B_2 \quad \text{with} \quad |B| > \mathcal{N}$$

We define Δ representing the set of all the multisets which can be obtained by taking actions from the bags of both processes (B_1 and B_2) and whose cardinal is less or equal than $\mathcal{N}$, i.e.:

$$\Delta = \{\ C = C_1 \cup C_2, \ \text{ with } \ C_1 \subseteq B_1, \ \ C_2 \subseteq B_2 \ \text{ and } \ |C| \leq \mathcal{N}\}$$

then, for each $C \in \Delta$ we can consider that the transitions

$$P \xrightarrow{B_1, \alpha_1} P'$$
$$Q \xrightarrow{B_2, \alpha_2} Q'$$

may be split into the following two transitions:

$$P \xrightarrow{C_1, \alpha_1} P'' \xrightarrow{B_1 - C_1, \alpha_1} P'$$
$$Q \xrightarrow{C_2, \alpha_2} Q'' \xrightarrow{B_2 - C_2, \alpha_2} Q'$$

where firstly the system executes all the C_i that make up the multiset C, which has been randomly chosen from those in Δ and then, in subsequent steps, the rest of actions from B_i will be executed.

Following up this idea, in rules $R4f$, $R4g$ and $R4h$ firstly actions of C (randomly chosen among the ones in $C \in \Delta$) are executed for a time $min(\alpha_1, \alpha_2)$, and then each process has to execute the rest of its bag. The process with greater time also has to finish the actions that it executed in parallel.

Rules $R5$ state behaviour of the parallel operator with synchronization. In this first model of our algebra, we only allow actions to synchronize if they have the same execution time, i.e., $\alpha_1 = \alpha_2$. Here, we have also decided to write a different rule ($R5a$) for the case in which the sum of actions to execute is smaller than the number of threads which can be executed in parallel. However, we must note that rule $R5b$ captures all the possibilities and rule $R5a$ has been written just for the sake of clarity.

In rule $R5a$ we execute a bag (B) consisting of no synchronization actions of bags B_1 and B_2 and just one copy of synchronization actions belonging to both bags. After B is executed, each process still has to execute those synchronization actions in its bag which are not contained in the bag of the process in parallel.

For rule $R5b$ we need to define Δ as follows:

$\Delta = \{\ C = C_1 \cup C_2 \cup A_1$ with $C_1 \subseteq B_1 \backslash A,\ \ C_2 \subseteq B_2 \backslash A,\ \ A_1 \subseteq (B_1 \cap B_2) \cap A$ and $|C| \leq \mathcal{N}\}$

i.e., bags with no synchronization actions of B_1 and B_2, and synchronization actions belonging to the bags of both processes.

Randomly one of these bags is chosen and executed leaving remaining actions until later.

Rule $R5c$ extends the parallel operator with synchronization to deal with synchronization actions with different execution times. In this case, we have decided that the action with the smallest execution time has to wait for the other action to finish.

Semantics for the hiding operator is established by the set of rules $R6$, where how a process can abstract the behaviour of any action is defined. And the last one, rule $R7$ captures the classical semantics for recursion in the usual fashion which helps us to define infinite processes.

5 Performance evaluation

With the formal resource-aware model that we have just defined, the timed characteristics of the system have been captured. Now we are concerned with defining an algorithm which allows us to estimate the required time to evolve between states. So, starting from the operational semantics we can make a transition graph for each process to help us in the performance evaluation.

Let T_{Si} be the expected time to evolve from *initial state* S_0 to S_i and, α_{ji} the label of the edge connecting S_j with S_i as shown in Figure 1.

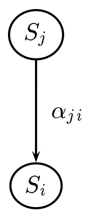

Fig. 1. Transition graph

Then $\forall j \in In(S_i), T_{Si} = Min(T_{Sj} + \alpha_{ji})$

where $In(S_i)$ is the set of predecessor nodes of S_i (it means S_i has an incoming edge from each node in $In(S_i)$).

Therefore, the required time to evolve from the state S_0 to the final one (S_F) can be computed by executing the presented recursive equation.

The transition graph built for each process is a directed one and it has weights (always a positive number) in its edges meaning the amount of time required to evolve between the nodes connected. As a result, the presented equation can be computed by the following algorithm.

We assume that our graph G is represented by an adjacency matrix where $cost[i][j]$ is the weight of the edge (i, j) and the weights of edges which do not belong to the graph are settled to ∞. We number the nodes of the graph from 0 to $n-1$.

Let S be the set of nodes for which the shortest time required to evolve from initial node to them have been calculated.

```
MinTime(int v, int cost[][SIZE], int n, int dist[]) {
// v: initial node
// cost: adjacency matrix with weights
// n: number of nodes in the graph
int u, minimum;
bool S[SIZE];

for (int i=0; i<=n-1; i++) {
    S[i] = false; dist[i] = cost[v][i];
}
S[v] = true; dist[v] = 0;
for (int num=1; num<=n-1; num++) {
// Choose u from among nodes not in S
// such that dist[u] is minimum.
    minimum = big_number;
    for (int i=1; i<=n-1; i++)
      if ((S[i]==false) && (dist[i]<minimum))
            minimum=dist[i]; u=i;
    S[u]=true;
    for (int w=1; w<=n; w++) // Update distances
        if ((S[w]==false) && (dist[w]>dist[u]+cost[u][w]))
            dist[w] = dist[u] + cost[u][w];
    }
}
```

6 Example: Sum of Matrices.

In this section, we present an example of specification using our language. We also prove that the algorithm which has been presented to estimate the required time to reach the final state in a model, works accurately because the times which have been obtained with it, match up exactly with the ones obtained in an experimental way. In this example, we have used a system which performs the addition of two matrix.

We assume, without loss of generality, that we have the same number of threads as rows (columns) in the matrices to sum. This system can be modelled by the presented algebra in the following way:

$$\text{AddMatriz} \equiv P_1||P_2||...||P_i$$

$$P_i \equiv (rd_i, 4).(add_i, 1).(wr_i, 2)$$

where rd_i is an action meaning "read a row", add_i is "add a row" and wr_i is "write a row".

It is not easy to depict the transition graph in a readable way due to the number of states grows considerably when the number of rows increases. So we have chosen to add two matrices with just three rows in a system with two processors ($\mathcal{N} = 2$). In Figure 2 the transition graph we obtain is shown, although taking into account the symmetry in the three possible paths from the initial node, just one of them has been depicted. For the sake of clarity, we have always chosen the biggest number of actions we can execute simultaneously in each moment.

There are several paths in which the required time to do this addition is minimum, in the transition graph in Figure 2 we have marked with dotted lines one of them and the meaning of the nodes in it is as follows:

$N_0 = (rd_1, 4).(add_1, 1).(wr_1, 2)||(rd_2, 4).(add_2, 1).(wr_2, 2)||(rd_3, 4).(add_3, 1).(wr_3, 2)$
$N_1 = (add_1, 1).(wr_1, 2)||(add_2, 1).(wr_2, 2)||(rd_3, 4).(add_3, 1).(wr_3, 2)$
$N_2 = (wr_1, 2)||(wr_2, 2)||(rd_3, 4).(add_3, 1).(wr_3, 2)$
$N_3 = (add_1, 1).(wr_1, 2)||(wr_2, 2)||(rd_3, 3).(add_3, 1).(wr_3, 2)$
$N_9 = (add_1, 1).(wr_1, 2)||(rd_3, 1).(add_3, 1).(wr_3, 2)$
$N_{15} = (wr_1, 2)||(add_3, 1).(wr_3, 2)$
$N_{21} = (wr_1, 1)||(wr_3, 2)$
$N_{23} = (wr_3, 1)$
$N_{26} = STOP$

We observe that the minimum required time to do this addition of matrices is eleven units of time while if we were to work in a sequential way or with interleaving we would have needed twenty-one units of time.

The same example of the addition of matrices has been developed for various matrix sizes and different number of processors, obtaining the results displayed in Table 6 where we can observe that the presented model represents accurately the expected behaviour of the system, decreasing the execution time because of the increase in the number of available processors until the optimum number of processors is reached.

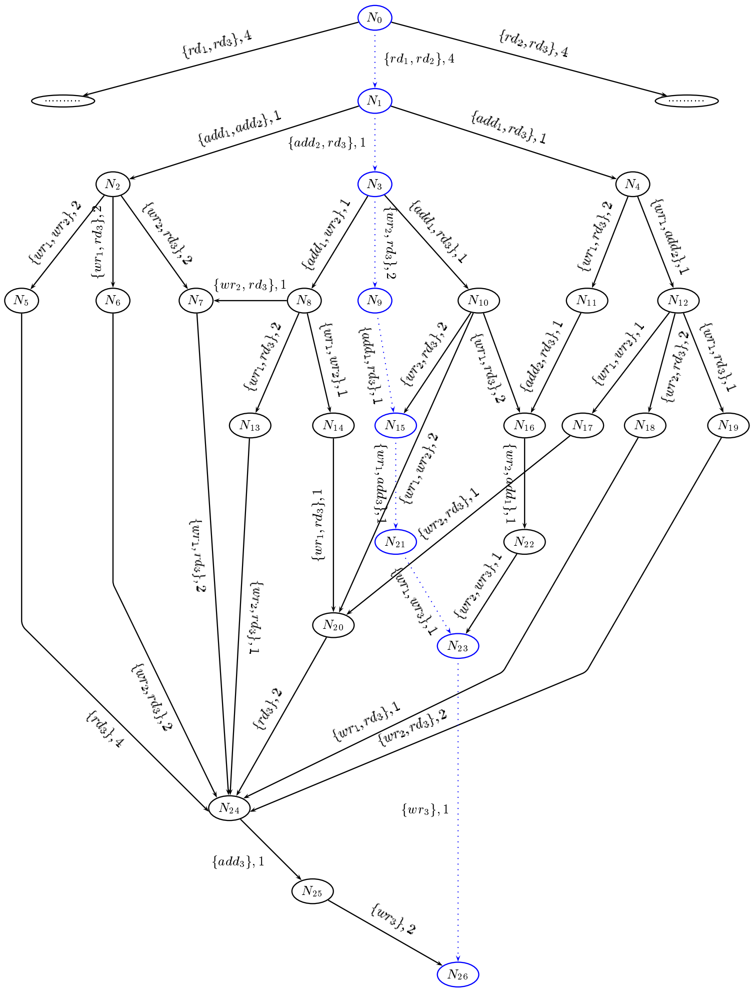

Fig. 2. Transition graph for the addition of two matrices 3x3 with N=2

7 Conclusions

In this paper we have defined an algebraic language which adds the execution time for each action and takes into account the number of resources which have processes at disposal. This algebra can be used for two different tasks; on one hand, starting from a known number of resources we are able to describe system behaviour at a relatively low level (e.g. matrices addition). On the other, if we start with some specification, we can find an appropriate number of resources (processors) for processes such that desired behaviour is preserved. This is particularly important for systems where the price of resources is essential.

All process algebra extensions which take into account action durations have to handle a number of resources. But in literature only two approaches are

Matrix size	Processors	T. Min	T. Max
3	2	11	21
3	3	7	21
4	2	14	28
4	3	10	28
4	4	7	28
6	2	21	42
6	3	14	42
6	4	11	42
6	6	7	42
10	2	35	70
10	3	24	70
10	4	18	70
10	5	14	70
10	6	12	70
10	7	11	70
10	8	11	70
10	9	11	70
10	10	7	70

Table 6. Sum of two matrices with various matrix sizes and different number of processors

known. It is supposed that processes have at disposal as many resources as they need (full parallelism) or they have just one resource/processor at disposal.

In the presented language we can model both cases by putting the value of variable N to a large number (full parallelism) and by putting N = 1 (interleaving).

8 Future Work

Our next goal (maybe challenge) of our future work is to link our resource-aware model to a tool for standard timed models making the analysis amenable. If it were impossible (or too hard) we will consider to design a new tool.

Another goal is to model a real system with the presented algebraic language. For this aim, we are considering some algorithms for image and video compression based on Wavelet methods. Compression of visual data is a computationally expensive task where large volumes of data have to be processed, therefore compression algorithms can greatly benefit from the use of parallel systems. In this setting, our algebraic language can be used to considerably reduce the required time to check new compression schemes (because it takes place at an early stage in the design) and it may also be used to find the most appropriate number of needed processors for optimum performance. A Wavelet-based parallel algorithm will be used as an example of the use of our algebra.

Our model suffers from the *state explosion problem* but we are working with parallelism, so this problem is less significant than in interleaving models due to the fact that parallelism leads to the sum of component states, rather than to their product (as in interleaving). Nevertheless, the problem remains, so we think that it may be useful to include some kind of scheduling in our model which allows the reduction of this state explosion.

References

[AGS00] K. Altisen, G. Gossler, and J. Sifakis. Scheduling-a methodology for the construccion of scheduled systems. In *Lecture Notes in COmputer Science*, pages 106–120, 2000.

[CdFV97] F. Cuartero, D. de Frutos, and V. Valero. A sound and complete proof system for probabilistic processes. In *ARTS'96, LNCS 1231*, 1997.

[CN94] Flavio Corradini and Rocco De Nicola. Distribution and locality of concurrent systems. In *ICALP'94*, pages 154–165, Berlin, 1994. Springer-Verlag.

[CS87] G. Costa and C. Stirling. Weak and strong fairness in CCS. *Information and Computation*, 73:207–244, 1987.

[FPY02] E. Fersman, P. Petterson, and Wang Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS2002)LNCS 2280*, pages 67–82, 2002.

[GJS90] A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of Working Conference on Programming Concepts and Methods, IFIP TC 2*, Sea of Galilee, Israel, 1990.

[GMS94] D.P. Gruska and A. Maggiolo-Schettini. Timed network semantics for communicating processes. In *Proceedings of CSP 93*, pages 125–143. Warsaw University Press, 1994.

[HHK02] H. Hermanns, U. Herzog, and J-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1–2):43–86, 2002.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[Liu00] Jane W.S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR'90, LNCS 458*, pages 401–415, Amsterdam, 1990.

[NdFL95] M. Núñez, D. de Frutos, and L. Llana. Acceptance trees for probabilistic processes. In *CONCUR'95, LNCS 962*, pages 249–263, 1995.

[RR88] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[RZ92] J.J.M.M. Rutten and J.I. Zucker. A semantics approach to fairness. *Fundamenta Informaticae*, XVI:1–38, 1992.

[Zic90] J.J. Zic. Some thoughts on communication system performance specification. In *Proceedings of the Open Distributed Processing Workshop*, Sydney, 1990.