Performance Evaluation of Process Partitioning Using Probabilistic Model Checking

Saddek Bensalem[1], Borzoo Bonakdarpour[2], Marius Bozga[1], Doron Peled[3], and Jean Quilbeuf[1]

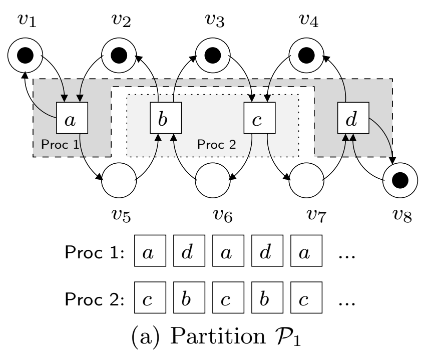
[1] UJF-Grenoble 1 / CNRS VERIMAG UMR 5104, France
{saddek.bensalem,marius.bozga,jean.quilbeuf}@imag.fr
[2] School of Computer Science, University of Waterloo, Canada
borzoo@cs.uwaterloo.ca
[3] Department of Computer Science, Bar Ilan University, Israel
doron.peled@gmail.com

Abstract. Consider the problem of *partitioning* a number of concurrent interacting processes into a smaller number of physical processors. The performance and efficiency of such a system critically depends on the tasks that the processes perform and the partitioning scheme. Although empirical measurements have been extensively used a posteriori to assess the success of partitioning, the results only focus on a subset of possible executions and cannot be generalized. In this paper, we propose a probabilistic state exploration method to evaluate a priori the efficiency of a set of partitions in terms of the speedup they achieve for a given model. Our experiments show that our method is quite effective in identifying partitions that result in better levels of parallelism.

Keywords: Concurrent programming, Scheduling, Speedup, Efficiency, Parallel programming, Formal methods.

1 Introduction

In concurrent programming, a program consists of multiple interacting *processes* that may run in parallel. Processes can potentially reside on different physical *processors* in order to speedup execution. Typically, there are far more processes than actual available physical processors. *Partitioning* is the problem of assigning processes to processors, so that the best level of parallelism and, ideally, maximum *speedup* is achieved. This problem is known to be notoriously challenging in the context of concurrent systems, as an intelligent scheduler must be able to make predictions about the state and temporal execution of interactions among processes. The research activities that deal with this problem range over a wide spectrum: from theoretical parallel and distributed algorithms and analytical performance analysis methods to empirical and experimental approaches. In all these activities, one measures the executions of different processes and utilization of processors of the system over time. Rules of thumb that involve static analysis of the system can be useful but of limited effect, since it is the dynamic



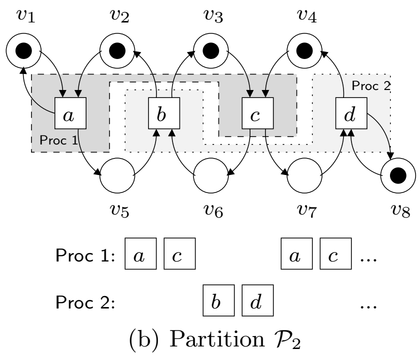


Fig. 1. An example of the effect of partitioning on concurrent scheduling

behavior that actually affects the efficiency. Empirical approaches provide us with an insight about specific dynamic behavior of the system under analysis, but typically they only focus on a very small subset of possible executions and cannot be generalized.

The focus of this paper is on evaluating and comparing partitions for concurrent programs by using state exploration techniques. Consider the abstract model of a concurrent program in Figure 1(a) running on two processors Proc 1 and Proc 2. The model has Boolean variables $v_1 \cdots v_8$, where the values of $v_1 \cdots v_4$ and v_8 are initially true and the rest are false. The transitions of this program are labeled by a, b, c, and d. For example, transition a updates the value of variables as follows: $v_2 = \mathit{false}$, $v_5 = \mathit{true}$, and the value of v_1 does not change. We assume that transitions that share variables cannot execute simultaneously (e.g., a and b). A *partition* maps transitions to processors. Let $\mathcal{P}_1$ be the partition, such that processor Proc 1 executes transitions a and d, and processor Proc 2 executes transitions b and c. Since transitions a and c are enabled in the initial state, Proc 1 and Proc 2 can execute them concurrently. After this execution, the values of variables are updated as follows: $v_2 = v_3 = v_4 = \mathit{false}$, $v_5 = v_6 = v_7 = \mathit{true}$. In this state, transitions b and d are enabled and, hence, Proc 1 and Proc 2 can concurrently execute d and b. Thus, both processors are always busy executing.

Now, consider a different scenario, where transitions are mapped to processors according to partition $\mathcal{P}_2$ (see Figure 1(b)). In the initial state, a and c are enabled, but Proc 1 can only execute one of them. If Proc 1 executes transition a, the system reaches a state from where only transition c is enabled. Since this is the only choice of execution, Proc 1 executes c and the system reaches a state where b and d are enabled. Since these transitions are both handled by Proc 2, they are executed sequentially. Thus, during that execution, partition $\mathcal{P}_2$ allows utilizing only half of the resources, that is one processor over two.

We propose an effective and relatively inexpensive test for assessing a priori the efficiency of concurrent execution of a system deployed according to a given partition. The task needs to be based on a simple but affordable analysis, and based on existing tools. We rule out solutions that involve long simulations. Observing the system for a long period, over actual runs, would indeed give a good answer to the problem, but may be sometimes unfeasible within the system development deadlines. Instead, we would like to use quick deep analysis, based on push-button techniques, such as model checking.

We first define a *maximal concurrency* model that is used to assess the run-time behavior of the system under different partitions. This concurrency model consists of a set of processors running in parallel and a partition of the transitions into these processors. We keep track of the current execution time at each processor. The execution of a transition may thus need the coordination and interaction of several processors. This requires the corresponding processors to be available. Thus, the execution of a transition in our maximal concurrency model involves first waiting for a time where all involved processors are available, and then consuming the time required to execute the transition in each one of them. Obviously, the different mapping of transitions into processors makes a big impact on execution time.

Our method analyzes the value of variables that keep track of execution time for each processor. We provide a probabilistic analysis, assuming that different nondeterministic and concurrent choices are selected randomly. The analysis relies on comparing execution time and the probability of speedup for finite prefixes of execution sequences according to a partition. We aim at keeping the complexity of conducting the analysis as low as possible. This is in particular challenging, as any analysis that takes into account the performance needs to, in some sense, sum up behavior parameters (e.g., some measurement of the duration of the execution) over time. This can potentially explode the size of the state space very quickly. Our solution is to provide an analysis for a limited execution sequence, say, of length k. As starting this analysis from only the initial state would be biased (perhaps the system always performs first the same constant initialization sequence), our method analyzes execution prefixes from the entire reachable state space of the system. This means that we take into account the probability of reaching some state, and, from there, the probability of speedup of executing a sequence of length k.

By experimenting with various values of k, we can tune between precision and additional complexity. More formally, one needs to identify the answer to the following question: with what probability can random executions according to one partition be faster by a factor of f. We, in particular, use the probabilistic model checker PRISM [10] to solve this problem. Our experiments clearly show the benefit of using our technique to assess the efficiency of different partitions.

Organization. In Section 2, we present our computation and concurrency model. Section 3 is dedicated to our method for comparing the performance of two given partitions. We present our case study in Section 4. Related work is discussed in Section 5. Finally, in Section 6, we make the concluding remarks and discuss future work.

2 Computation Model

2.1 Transition Systems

Definition 1. *A* transition system $\mathcal{T}$ *is a tuple* $\langle V, S, T, \iota \rangle$*, where*

- V *is a finite set of finite-domain* variables.
- S *is the set of* states *that are valuations of the variables* V*,*
- T *is a set of* transitions. *With each transition* $\tau \in T$*, we have:*
  - *An* enabling condition $en_\tau : S \rightarrow \{true, false\}$ *over the variables* V *that must be true for the transition* τ *to execute.*
  - *A* transformation $f_\tau : S \rightarrow S$ *that modifies the current state on execution of* τ*.*
- $\iota \in S$ *is the* initial state*, that is the initial valuation of the variables.* □

For example, the model in Figure 1 has Boolean variables $v_1 \cdots v_8$ and four transitions a, b, c, and d. The initial state is where $v_1 = v_2 = v_3 = v_4 = v_8 = true$ and $v_5 = v_6 = v_7 = false$. Moreover, en_a is $v_1 = v_2 = true$. In Section 1, we described how the transitions update the state of the program.

Definition 2. *An* execution σ *is a maximal sequence of states* $\sigma = s_0 s_1 s_2 \ldots$*, such that*

- $s_0 = \iota$.
- *For each* $0 \leq i < |\sigma|$*, there exists some* $\tau \in T$*, such that* $en_\tau(s_i)$ *holds and* $s_{i+1} = f_\tau(s_i)$*.* □

We choose to use simple transition systems, as they can already illustrate our approach without the need to use complicated details. In particular, we abstract away processes; concurrency between transitions is allowed when they use disjoint set of variables. In fact, when the system is divided logically into processes, there is no need to use a more complicated model: the program counter of each process is an additional variable that makes all the transitions of a single process interdependent.

2.2 Maximal Concurrency Model

Given a transition system $\mathcal{T} = \langle V, S, T, \iota \rangle$, in order to reason about concurrency and speedup, we need to take the execution of transitions and their duration into account. Let P be a fixed finite set of *processors*. The mapping of the system $\mathcal{T}$ on the processors P is specified through a partition $\mathcal{P}$ of the set $V \cup T$. Each class in $\mathcal{P}$ corresponds to a set of variables and/or transitions handled by a given processor in P. Thus, for a variable $v \in V$ (respectively, transition $\tau \in T$), $\mathcal{P}(v)$ (respectively, $\mathcal{P}(\tau)$) returns a processor in P. Specifying the mapping of the variables explicitly allows us to determine which processor is used when accessing the variables.

Since computing en_τ and f_τ, where $\tau \in T$, involves dealing with a set of variables, execution of τ will engage a *set* of processors. Formally, given a partition $\mathcal{P}$, if transition τ is associated with variables V_τ, then the set of processors engaged in executing τ is

$$P_\tau^{\mathcal{P}} = \{p \in P \mid p = \mathcal{P}(\tau) \vee \exists v \in V_\tau : p = \mathcal{P}(v)\}$$

We assume that executing a transition τ engages each processor in $P_\tau^{\mathcal{P}}$ for some (not necessarily equal) constant time. Given a processor $p \in P_\tau^{\mathcal{P}}$, we call this time the *duration* of transition τ on processor p, and denote it by $p_\tau^{\mathcal{P}}$. Furthermore, in order to analyze and keep track of the duration for which a processor is still engaged in a transition, we introduce a *history (time) variable* $t_p^{\mathcal{P}}$ for each processor $p \in P$.

In order to measure execution time of processors for $\mathcal{T}$, we augment executions of $\mathcal{T}$ with history variables.

Definition 3. *Let ρ be a finite segment (suffix of a prefix) of an execution sequence σ and $\mathcal{P}$ be a partition. The* augmented execution *of ρ according to partition $\mathcal{P}$ includes updates to the variables $t_p^{\mathcal{P}}$ for each processor $p \in P$ as follows:*

- *In initial state ι, $t_p^{\mathcal{P}} = 0$, for all $p \in P$.*
- *Executing τ involves the following updates:*
  - *We let $c = \max\{t_p^{\mathcal{P}} \mid p \in P_\tau^{\mathcal{P}}\}$; i.e., the maximum value among the value of history variables of processes associated with τ.*
  - *We let $t_p^{\mathcal{P}} = c + p_\tau^{\mathcal{P}}$, for each process $p \in P_\tau^{\mathcal{P}}$; i.e., after synchronization, the history variable $t_p^{\mathcal{P}}$ is updated, as $p_\tau^{\mathcal{P}}$ time units elapsed in the local time of process p.* □

Figure 2 shows the effect of executing a transition that involves processors p_1, p_3, and p_4 on the value of history variable for each processor. Notice that first, the value of all variables are updated with the maximum time value, and then, the time durations of each processor are added, respectively. The augmented execution of any two transitions involving a common processor is sequentialized through the history variable of the common processor. In particular, two transitions involving a common variable v are never executed concurrently, since they both involve processor $\mathcal{P}(v)$.

3 Evaluating the Effect of Partitions on Speedup

3.1 The Notion and Metrics for Execution Speedup

Our main goal is to compare the execution time of each processor through analyzing history variables under different partitions. The comparison relies on the following definitions of respectively execution time, sequential execution time, and speedup, for a finite prefix of execution sequences according to a partition.

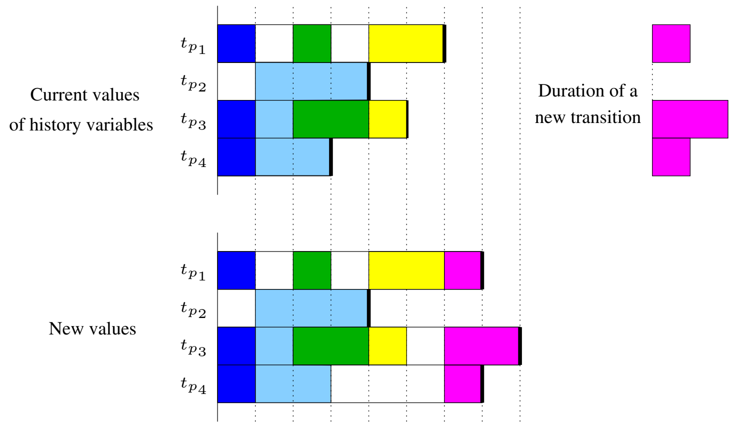


Fig. 2. The effect of a new transition involving processors $\{p_1, p_3, p_4\}$ on history variables

Definition 4. *Given a set P of processors, a partition $\mathcal{P}$, and a finite execution prefix ρ,*

- *the* execution time *of ρ is $exec_{\mathcal{P}}(\rho) = \max\{t_p^{\mathcal{P}} \mid p \in P\}$, where $t_p^{\mathcal{P}}$ are obtained from augmented execution of ρ according to the partition $\mathcal{P}$,*
- *the* sequential execution time *is*

$$seqexec_{\mathcal{P}}(\rho) = \sum_{\tau \in T} \#(\rho, \tau) \cdot \max\{p_\tau^{\mathcal{P}} \mid p \in P_\tau^{\mathcal{P}}\}$$

  where $\#(\rho, \tau)$ denotes the number of times τ is executed in ρ,
- *the* speedup *achieved by partition $\mathcal{P}$ when executing ρ is the ratio*

$$speedup_{\mathcal{P}}(\rho) = \frac{seqexec_{\mathcal{P}}(\rho)}{exec_{\mathcal{P}}(\rho)}$$

□

In order to provide a detailed analysis, one needs information about the timing duration of transitions. This information is based on the processor partition and the actual protocol used to provide the correct interactions between all the elements (in particular the participating processors) involved in its execution. Moreover, to combine all results obtained on different execution sequences, we need a probability distribution on the occurrences of such sequences. In particular, this can be obtained from a probability distribution of execution of different transitions of the system, whenever several of them are simultaneously enabled. In case a probability distribution of simultaneously enabled transitions is not given, then one can assume identical probability of execution for all of them. If the probability distribution μ on finite prefixes of execution ρ is given, then we can answer the following questions:

(i) What is the probability distribution of $\mathit{speedup}_{\mathcal{P}}^N$, that is, the speedup achieved by $\mathcal{P}$ on execution sequences of length N? This distribution can be in fact explicitly computed as follows, given any possible value of the speedup s :

$$\mathbb{P}\left[\mathit{speedup}_{\mathcal{P}}^N \geq s\right] = \sum \left\{\mu(\rho) \mid \mathit{length}(\rho) = N \wedge \mathit{speedup}_{\mathcal{P}}(\rho) \geq s\right\}$$

(ii) What is the expected mean value of $\mathit{speedup}_{\mathcal{P}}^N$, that is, the average speedup value obtained on all sequences of length N? Again, this value can be computed by direct summation as follows:

$$\mathbb{E}\left[\mathit{speedup}_{\mathcal{P}}^N\right] = \sum \left\{\mu(\rho) \cdot \mathit{speedup}_{\mathcal{P}}(\rho) \mid \mathit{length}(\rho) = N\right\}$$

(iii) What is the probability that partition $\mathcal{P}_1$ is better than $\mathcal{P}_2$ by at least, for instance, 20% on sequences of length N? The answer can be directly computed as:

$$\mathbb{P}\left[\frac{\mathit{speedup}_{\mathcal{P}_1}^N}{\mathit{speedup}_{\mathcal{P}_2}^N} \geq 1.2\right] = \sum \left\{\mu(\rho) \mid \mathit{length}(\rho) = N \wedge \frac{\mathit{speedup}_{\mathcal{P}_1}(\rho)}{\mathit{speedup}_{\mathcal{P}_2}(\rho)} \geq 1.2\right\}$$

We note the following about our analysis method:

1. Probabilities on executing transitions can be given as input. Otherwise, the method assumes that all transitions enabled at some state are executed with equal probability.
2. Execution duration of transitions can also be given as input in terms of time units. Otherwise, we assume that the duration of a transition is 1 time unit for each participating processor. Thus, the sequential execution time $\mathit{seqexec}_{\mathcal{P}}(\rho)$ of any finite prefix ρ is equal to the length of the prefix $\mathit{length}(\rho)$ and, hence, independent of the partition $\mathcal{P}$. In this case, we can use the speedup as a performance indicator. That is, a greater speedup means equivalently smaller (faster) execution time for a partition.
3. Complete analysis is expensive. Timing analysis explodes the state space and makes even a finite state system, potentially, an infinite one. We will, thus, analyze limited segments of executions. To avoid biasing the measurement, we will not always start the analysis from the initial state, but from any reachable state. Furthermore, we will weigh these measurements according to the probability to reach and execute such finite fragments.

3.2 Our Assessment Solution

Because of the high complexity of timing analysis for arbitrarily long sequences, we choose to check only executions of limited size k, with $k \ll N$ and depending on the amount of time and memory available, e.g., running $k = 10$ execution steps. However, a naive implementation of this idea would be quite biased. For example, an ATM system may start with verifying the PIN code, not revealing within the first few transitions executed a whole lot about the nature of the rest of the execution.

For this reason, we check the behavior of sequences of k transitions from *all* reachable states, based on the calculated probability (under the stated assumptions in Subsection 3.1) of reaching that state. We start building augmented execution prefixes from that point. Let zero be the proposition that states that the value of all history variables are zero. For an arbitrary expression η encoding the property of interest, we compute the probability of temporal properties of the form

$$\phi(k, \eta) \equiv \Diamond(\mathsf{zero} \wedge \bigcirc^k \eta)$$

where $\bigcirc^k$ is the application of the next-time operator $\bigcirc$ in temporal logic k times.

Using specific η properties, we can compute approximate answers to questions (i)-(iii) mentioned in Subsection 3.1. For example, to answer questions (i) and (ii) for a fixed partition $\mathcal{P}$, we consider $\eta \equiv exec_{\mathcal{P}} \leq \frac{k}{s}$ (where s is a given possible value of the speedup) and approximate

$$\mathbb{P}\left[speedup_{\mathcal{P}}^{N} \geq s\right] \approx \mathbb{P}\left[\phi(k, exec_{\mathcal{P}} \leq \tfrac{k}{s})\right]$$

$$\mathbb{E}\left[speedup_{\mathcal{P}}^{N}\right] \approx \textstyle\sum_{j=1}^{k} \frac{k}{j} \cdot \mathbb{P}\left[\phi(k, exec_{\mathcal{P}} = j)\right]$$

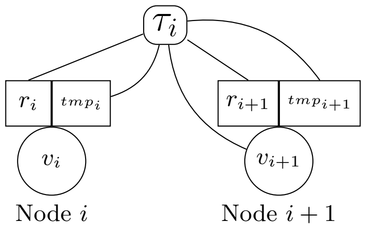
Likewise, the average speedup value is approximated as the weighted average of all potential speedups $\{\frac{k}{j} \mid 1 \leq j \leq k\}$ that can be achieved on sequences of length k. To answer question (iii), we must consider execution of two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ simultaneously. Accordingly, the two partitions will have different durations for transitions. In order to make a comparison between partitions, we allow using multiple history variables and constants per a transition system. Thus, we may use two sets of processors P_1 and P_2, two sets of history variables and two sets of duration constants per partition. In this case, we consider

$$\eta \equiv \frac{exec_{\mathcal{P}_2}}{exec_{\mathcal{P}_1}} \geq 1.2$$

and approximate:

$$\mathbb{P}\left[\frac{speedup_{\mathcal{P}_1}^{N}}{speedup_{\mathcal{P}_2}^{N}} \geq 1.2\right] \approx \mathbb{P}\left[\phi\left(k, \frac{exec_{\mathcal{P}_2}}{exec_{\mathcal{P}_1}} \geq 1.2\right)\right]$$

In order to efficiently construct augmented prefixes for k steps at each state, we provide *two copies* of the transitions: one copy that does not count (i.e., no augmented executions), before zero becomes true. Then, a transition that causes zero to hold, exactly once, occurs, it sets up a flag and the second copy of the transitions starts to apply augmented executions. This replication of the transitions guarantees that the analyzed property does not count the time in doing a reachability analysis, but rather constrain the counting to the k last steps. The transition that guarantees zero has its own probability, and on the



$$f_{\tau_i} : \begin{cases} v_{i+1} = min(v_{i+1}, tmp_i) \\ tmp_{i+1} = max(v_{i+1}, tmp_i) \\ r_{i+1} = true \\ r_i = false \end{cases}$$

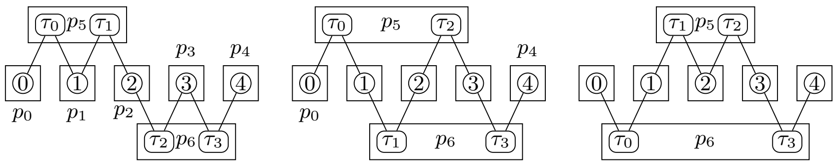
Fig. 3. Detail of a propagation transition

face of it, one needs to normalize the probability of $\phi(k, \eta)$ with the probability to execute this transition. However, the probability of skipping this transition forever is zero. Hence no actual normalization is needed.

4 Case Study : Sorting Chain

Our case study to evaluate our technique is a chain of nodes that sorts values through a propagation mechanism. Values to sort enter the chain through the leftmost node. Each node compares the incoming value on its left with its current value, keeps the smaller one and propagates the greater one to its right. This propagation scheme sorts the values in ascending order from left to right.

We model the node i as a set of variables (v_i, tmp_i, r_i), where v_i is the current value stored in the node, tmp_i is the value to be propagated to the right, and r_i is a Boolean variable that is true whenever the node has to propagate a value to the right. Propagation of a value from node i to node $i+1$ is modeled through the transition τ_i, as shown in Figure 3. The enabling condition is that the node i has a value to propagate and $i+1$ has not. Formally, $en_{\tau_i} = r_i \wedge \neg r_{i+1}$. The effect of the propagation can be described as follows. The propagation transitions compare the current value of $i+1$ with the value propagated by i. The smaller one becomes the new value of $i+1$ while the greater one is propagated by $i+1$. The variables r_{i+1} and r_i are updated to indicate that i has propagated its value and $i+1$ has now a value to propagate. Note that transitions τ_i and τ_{i+1} cannot execute simultaneously since both access variables of node $i+1$.



(a) Neighbor pair partition. (b) Distant pair partition. (c) Concentric pair partition.

Fig. 4. Different partitions for sorting chain of 5 nodes

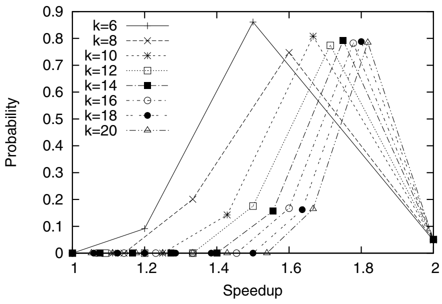


Fig. 5. Approximations of speedup distribution, for the model with 5 nodes using the neighbor pair partition

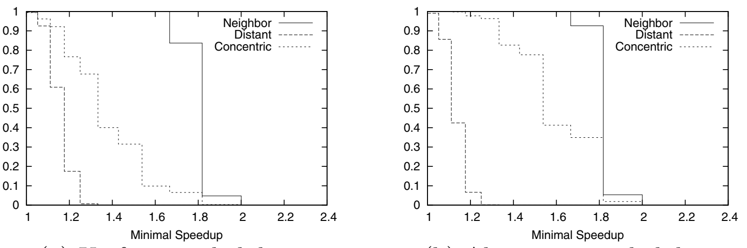
We assume that each node i is maintained by its own private processor p_i. Then, we consider additional support processors for executing transitions. Execution of a transition implies its support processor and the two processors hosting the corresponding nodes. For simplicity, we assume that the number of transitions is even (and, hence, the number of variables is odd). We consider the following partitions:

- Neighbor pairs ($\mathcal{P}_1$): we add one processor for each pair (τ_{2i}, τ_{2i+1}), where $i \in \{0, \ldots, \frac{n}{2}\}$. With 5 variables, we have p_5 for executing τ_0 and τ_1, p_6 for executing τ_2 and τ_3, as depicted in Figure 4(a).
- Distant pairs ($\mathcal{P}_2$): we add one processor for each pair of transitions $(\tau_i,\tau_{i+\frac{n}{2}})$, where $i \in \{0, \ldots, \frac{n}{2}\}$. With 5 variables, transitions τ_0, τ_2 are executed by p_5 and transitions τ_1, τ_3 are executed by p_6, as depicted in Figure 4(b).
- Concentric pairs ($\mathcal{P}_3$): we add one processor for each pair of transitions $(\tau_i,\tau_{n-(i+1)})$, where $i \in \{0, \ldots, \frac{n}{2} - 1\}$. With 5 variables, transitions τ_0, τ_3 are executed by p_5 and transitions τ_1, τ_2 are executed by p_6, as depicted in Figure 4(c).

4.1 Approximating Speedup

Our first experiment shows how the approximation behaves when increasing the length k of the considered execution sequences. We compute an approximation of the distribution of $\mathit{speedup}_{\mathcal{P}_1}$ for different values of k. This is obtained by checking with PRISM [10] the probability of $\phi(k, \mathit{speedup}_{\mathcal{P}_1} = s)$, where $s \in \{\frac{k}{j} \mid 1 \leq j \leq k\}$.

Figure 5 shows the speedup distribution for different values of k for the neighbor pairs partition $\mathcal{P}_1$. The lines are added for readability, the actual results are the points that correspond to the possible values of speedup. When k increases, there are only three speedup values with a strictly positive probability. Namely,



(a) Uniform probability. (b) Alternative probability.

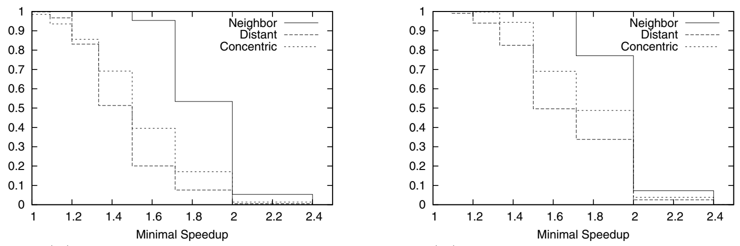
Fig. 6. Complementary cumulative distribution function of the speedup for the three proposed partitions, for the chain of 5 nodes

these are 2, $\frac{k}{\frac{k}{2}+1}$ and $\frac{k}{\frac{k}{2}+2}$. To obtain a speedup of 2, each support processor is used at every time slot. A speedup of $\frac{k}{\frac{k}{2}+1}$ is obtained when one of the processors is unused during one time slot. Similarly, a speedup of $\frac{k}{\frac{k}{2}+2}$ is obtained when there are two time slots where one processor is unused. Thus, in that example $\mathit{speedup}_{\mathcal{P}_1}^N$ converges toward 2 as N goes to ∞, which is visible in the successive distributions obtained.

By checking the probability of $\phi(k, \mathit{speedup}_{\mathcal{P}} \leq s)$, we obtain an approximation of the complementary cumulative distribution function for the speedup, given a partition $\mathcal{P}$. Since, the probabilities on executing transition can be given as input, we compare here the uniform probability and an alternative probability. In the latter, the probability of executing a given transition is higher, if it was enabled and not executed for a greater number of steps. To compute the probability of $\phi(k, \mathit{speedup}_{\mathcal{P}} \leq s)$, we take $k = 20$ for the chain of 5 nodes. The time for PRISM to build the model and compute the probabilities is about 20 minutes on a 3GHz dual-core machine running Debian Linux. For the chain of 7 nodes, we take $k = 12$. The time for computing the probabilities is then about 9 hours.

The obtained functions are shown in Figure 6, for the chain of 5 nodes and Figure 7 for the chain of 7 nodes. Consider the chain of 5 with uniform probability (Figure 6(a)). The probability of "the speedup is 1.4 or greater" is 0 for the partition $\mathcal{P}_2$ (distant pair partition), roughly 0.4 for the partition $\mathcal{P}_3$ (concentric pair partition), and 1 for the partition $\mathcal{P}_1$ (neighbor pair partition). For the chain of 5 nodes, the speedup is at most 2, since there are only 2 support processors for transitions. For the chain of 7 nodes, it ranges between 1 and 3.

The alternative probability yields a higher average speedup than uniform probability. Indeed, if two transitions can be executed in parallel, executing one of them increases the probability of executing the other one immediately after, thus enforcing parallelism. Table 1 summarizes the results by giving the average speedup for each configuration. This gives us the first method for assessing the efficiency of partitions. Note that the probabilities or the average can be computed independently for each partition.



(a) Uniform probability. (b) Alternative probability.

Fig. 7. Complementary cumulative distribution function of the speedup for the three proposed partitions, for the chain of 7 nodes

4.2 Comparing Partitions

In order to compare more precisely two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$, we compare their speedup by considering $\frac{speedup_{\mathcal{P}_1}}{speedup_{\mathcal{P}_2}}$. By checking the probability of the property

$$\phi\left(k, \frac{speedup_{\mathcal{P}_1}}{speedup_{\mathcal{P}_2}} \geq \alpha\right)$$

for the possible speedup ratios α, we obtain again a complementary cumulative distribution function. Figure 8(a) and Figure 8(b) compare partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ for the chains of 5 and 7 nodes. Note that equation

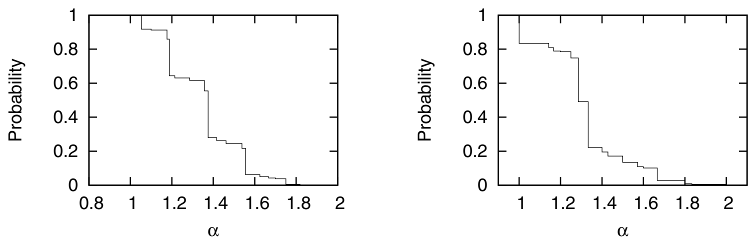
$$\mathbb{P}\left[\frac{speedup_{\mathcal{P}_1}}{speedup_{\mathcal{P}_2}} \geq 1\right] = 1$$

means that partition $\mathcal{P}_1$ cannot exhibit a smaller speedup than the partition $\mathcal{P}_2$. Furthermore, with probability 0.6, the speedup of partition $\mathcal{P}_1$ is 20% better than the speedup of partition $\mathcal{P}_2$, and with probability 0.2 it is 40% better. These results come from the fact that $\mathcal{P}_1$ preserves the intrinsic parallelism of the model, whereas $\mathcal{P}_2$ prevents some interactions to execute in parallel. Here, partition $\mathcal{P}_1$ is clearly better than partition $\mathcal{P}_2$.

Figures 9(a) and 9(b) compare $\mathcal{P}_2$ and $\mathcal{P}_3$, for the models of 5 and 7 nodes. In this case, both partitions are restricting the intrinsic parallelism of the model, but none of them is always better than the other. For the chain of 5 nodes,

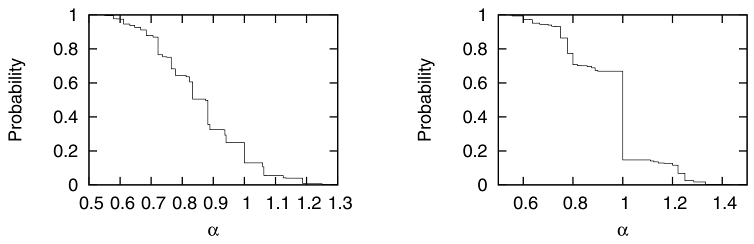
	Uniform Probability			Alternative Probability		
Number of nodes	Partition $\mathcal{P}_1$	Partition $\mathcal{P}_2$	Partition $\mathcal{P}_3$	Partition $\mathcal{P}_1$	Partition $\mathcal{P}_2$	Partition $\mathcal{P}_3$
5	1.80	1.16	1.38	1.82	1.13	1.60
7	1.88	1.46	1.56	1.96	1.67	1.79

Table 1. Average speedup



(a) Chain of 5 nodes. (b) Chain of 7 nodes.

Fig. 8. Probability of $\frac{speedup_{\mathcal{P}_1}}{speedup_{\mathcal{P}_2}} \geq \alpha$.



(a) Chain of 5 nodes. (b) Chain of 7 nodes.

Fig. 9. Probability of $\frac{speedup_{\mathcal{P}_2}}{speedup_{\mathcal{P}_3}} \geq \alpha$

$$\mathbb{P}\left[\frac{speedup_{\mathcal{P}_1}}{speedup_{\mathcal{P}_2}} \geq 0.5\right] = 1$$

since both speedups are between 1 and 2. Likewise,

$$\mathbb{P}\left[\frac{speedup_{\mathcal{P}_1}}{speedup_{\mathcal{P}_2}} \geq 2\right] = 0$$

However, the probability that $\mathcal{P}_2$ strictly outperforms $\mathcal{P}_3$ ($\alpha > 1$) is less that 0.2, meaning that the probability that $\mathcal{P}_3$ outperforms or performs as well as $\mathcal{P}_2$ is more than 0.8. Hence, $\mathcal{P}_3$ seems apriori better than $\mathcal{P}_2$.

5 Related Work

The (max, +) discrete event systems theory [7] provides conditions about the existence of asymptotic values for throughput, growth rates, cycle time, etc, as well as analytic methods for their computation. Nonetheless, these results are usually restricted to particular classes of systems. As an example tightly related to our work, one can mention the study of the asymptotic growth rate for heaps of pieces in Tetris games, as presented in [9]. Using our computational model, this problem is equivalent to the computation of the limit/asymptotic speedup for transition systems where transitions are continuously enabled and fire according to a uniform probability. For this particular problem, the $(\max, +)$ systems theory guarantees the existence of the limit speedup. However, the computation of this limit, known as the Lyapunov exponent, is to the best of our knowledge still an open problem actively studied within the $(\max, +)$ community.

Schedulability analysis using verification has been addressed in different contexts. For example, in [1], the authors model jobshop and precedence task scheduling using timed automata. The schedulability problem is then reduced to reachability problem in timed automata. On the contrary, the focus of our work is not on schedulability analysis, but rather on evaluating and comparing two existing partitions using quantitative analysis. Likewise, the work in [8,11] discusses partitioning and execution of concurrent Petri nets, but fall short in evaluation of partitions.

The work in [4,3,5] proposes transformations for automated implementation of concurrent (e.g., multi-threaded, multi-process, or distributed) systems from a component-based abstract model. One input to the transformation algorithms is a partition scheme that maps concurrent components and schedulers to processors and/or machines. Although the focus of the work in [4,3,5] is on automating the implementation of a concurrent system, it falls short on evaluating different partition schemes to choose the one that results in the best speedup for the concurrent implementation. The evaluation technique presented in this paper can assist in choosing a better input partition to automatically implement concurrent programs.

Finally, most approaches based on classic performance evaluation (e.g., in [6,2]) are limited to specific network topologies. On the contrary, our technique is independent of topology and it benefits from the push-button model checking technology. This makes the tedious task of performance evaluation for different topologies much easier.

6 Conclusion

In this paper, we studied the problem of evaluating and comparing two given *partitions*, where each transition and variable of a transition system is mapped to a processor and the system allows some concurrency among its transitions. We defined a notion of *maximal concurrency* that is used to assess the runtime behavior of the system under different partitions. Our notion of maximal concurrency model consists of a set of processors, a partition, a variable per processor that keeps track of the current execution time at the processor, and a constant, which describes the duration of transition.

We developed an effective and relatively inexpensive test for assessing and comparing a priori the efficiency of two partitions. Our method on observing the system for a long period, over actual runs, or empirical studies. It analyzes the value of variables that keep track of execution time for each processor. The analysis relies on comparing execution time, sequential execution time, and the probability of speedup, for finite prefixes of execution sequences of some constant size k according to a partition. We utilized a probabilistic model checker to answer the abstract question: with what probability can random executions according to one partition be faster by a factor of f. Our experiments on popular distributed algorithms clearly showed the effectiveness of our technique to assess the efficiency of different partitions.

For future work, we are planning to design methods that automatically generate partitions that are likely to perform well. Another interesting research direction is to incorporate other parameters, such as power consumption (e.g., in sensor networks), network load, etc in assessing the efficiency of partitions.

Acknowledgments. This research was supported in part by Canada NSERC Discovery Grant 418396-2012 and NSERC Strategic Grant 430575-2012.

References

1. Abdeddaïm, Y., Asarin, E., Maler, O.: Scheduling with timed automata. Theoretical Computer Science 354(2), 272–300 (2006)
2. Bianchi, G.: Performance analysis of the IEEE 802.11 distributed coordination function. IEEE Journal on Selected Areas in Communications 18, 535–547 (2000)
3. Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: Automated conflict-free distributed implementation of component-based models. In: IEEE Symposium on Industrial Embedded Systems (SIES), pp. 108–117 (2010)
4. Bonakdarpour, B., Bozga, M., Jaber, M., Quilbeuf, J., Sifakis, J.: A framework for automated distributed implementation of component-based models. Journal on Distributed Computing (DC) 25(1), 383–409 (2012)
5. Bonakdarpour, B., Bozga, M., Quilbeuf, J.: Automated distributed implementation of component-based models with priorities. In: ACM International Conference on Embedded Software (EMSOFT), pp. 59–68 (2011)
6. Cao, M., Ma, W., Zhang, Q., Wang, X., Zhu, W.: Modelling and performance analysis of the distributed scheduler in IEEE 802.16 mesh mode. In: ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pp. 78–89 (2005)
7. Olsder, G.J., Bacelli, F., Cohen, G., Quadrat, J.P.: Synchronization and Linearity. Wiley (1992)
8. Ferscha, A.: Concurrent execution of timed Petri nets. In: Winter Simulation Conference, pp. 229–236 (1994)
9. Gaubert, S.: Methods and applications of (max,+) linear algebra. Technical Report 3088, INRIA (January 1997)
10. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
11. Cortadella, J., Kondratyev, A., Lavagno, L., Passerone, C., Watanabe, Y.: Quasi-static scheduling of independent tasksfor reactive systems. In: Esparza, J., Lakos, C.A. (eds.) ICATPN 2002. LNCS, vol. 2360, pp. 208–227. Springer, Heidelberg (2002)