Predicting Performance in the Presence of Stochastic Information

Natalia López, Manuel Núñez, Fernando Rubio

Dpto. Sistemas Informáticos y Programación
Facultad de Informática
Universidad Complutense de Madrid
28040-Madrid, Spain
{natalia,mn,fernando}@sip.ucm.es

Abstract. Predicting the cost of a given implementation of a skeleton is a key issue when dealing with skeleton-based parallel languages. Unfortunately, this is not a trivial task when the granularity is not fixed. In this paper we present a methodology to provide cost models in case the granularity of each process is given by a random variable. We will compute, for a given implementation, a random variable indicating the *time* associated with its performance. Thus, we will have not a unique time but information such as "with probability p the time will be less than or equal to t". We will consider a simple process algebraic language to express this stochastic-time information. In order to illustrate our framework, we will present two skeletons and we will show how the cost models associated with possible implementations are computed.

1 Introduction

During the last decade many parallel languages based on skeletons have been developed and several mature implementations can be used nowadays (see e.g. [15, 1, 14, 10, 4]). A skeleton [3] is a *parallel problem solving scheme* applicable to certain families of problems. For example, the divide & conquer schema can be abstracted in a single skeleton. The specific functions to be performed in the nodes of the process topology are abstracted as parameters. For instance, in order to parallelize merge-sort it is enough to specify this algorithm in terms of the divide & conquer skeleton. Then, the actual parallel implementation will be delegated to the underlying skeleton.

One of the main characteristics of skeletons is that it should be possible to predict the efficiency of each implementation. This can be done by providing a *cost model* together with each implementation. A cost model is just a formula stating the predicted performance time of the algorithm (see e.g. [7] for a survey on the subject). To build this formula, the implementor has to consider all the activities which take place in the *critical path* of the algorithm. This includes the initial sequential actions needed to put at work all the processors of the parallel machine, the maximum of the individual times needed by the processors, and the final sequential actions, which take place between finishing the last subtask and delivering the final result. Cost models are parameterized by some constants that may depend either on the problem to be solved, on the underlying parallel architecture, or on the runtime system being used.

In previous work [16], simple cost models for the Eden language [8] were introduced. In this first approach, there were limitations that diminished the applicability of them. One of the main restrictions, common to many other cost models, is that the performance of the different tasks is estimated as a fix amount of time. However, this restriction is not realistic in the general case. In this paper we will consider that possible different performances are

* Work supported in part by the MCYT project TIC 2003-07848-C02-01 and the JCCLM project PAC-03-001.

estimated by using random variables. That is, instead of using fixed times, we will have random variables indicating that with a certain probability the performance time is less than or equal to a certain time. Another drawback of our previous cost models is that they were defined by considering a given distribution of tasks among processors. In this paper we consider *all* possible distributions and we assign probabilities to them. By combining the random variables associated with tasks and the distributions among processors, we will compute a unique random variable expressing the *time* needed to complete the whole computation. In particular, we may take the expected value (mean) of this random variable as a simple estimation of this time. Moreover, if we combine this value with the variance of the random variable we can apply well-known results from probability theory to estimate, with a certain confidence level, the time of performance.

The relevant information, concerning time of performance, of the possible implementations of a skeleton will be specified by using a simple stochastic process algebra. Thus, we use a formal language where the performance details of an implementation can be appropriately described in terms of times expected for each task (given by random variables) and the parallel composition of these tasks. Given the fact that, for the study of performance, we may (partially) abstract communications, we do not need a language featuring all the characteristics of usual process algebras. In fact, we consider a simple subset of the language given in [13], where we also presented a translation mechanism from processes described in the formal language into Eden programs. Thus, by using this integrated approach we can work both at the process algebraic and at the Eden levels.

The rest of the paper is organized as follows. In the next section, we present our preliminary approach to predicting the cost of Eden skeletons, showing its limitations. Then, in Section 3 we introduce the formalism that we use in the next section to specify performance information of implementations. As we indicated before, we will consider a very simple model where stochastic-time information (in terms of random variables) can be appropriately specified. In order to illustrate our methodology, in Section 4 we show how our framework can deal with parallel implementations of the map and divide & conquer skeletons. Finally, in Section 5 we present our conclusions and some lines for future work.

2 First Approach to Cost Models in Eden

In this section we briefly review previous work on cost models for skeletons in the parallel functional language Eden. First, we recall the basic constructions of the language. Next, we show how these cost models were defined.

2.1 Basic Constructions of Eden

Eden[1] extends the lazy functional language Haskell [17] by using syntactic constructions that explicitly define processes. This language includes a new expression `process x -> e` to define a *process abstraction* having the variable `x` as input and the expression `e` as output. Process abstractions can be compared to functions. In this similarity, the main difference is that processes, when instantiated, may be executed in parallel.

Given a process abstraction, a *process instantiation* is achieved by using the predefined infix operator `#`. Each time an expression `e1 # e2` is evaluated, the instantiating process will be responsible for evaluating and sending `e2`, while a new process is created to evaluate the application `(e1 e2)`. We will refer to the latter as the *child* process and to the owner of the instantiation expression as the *parent* process.

[1] The Eden compiler can be freely downloaded from the main web page of the project: `http://www.mathematik.uni-marburg.de/inf/eden`.

Problem dependent parameters	
N	Number of tasks of the input
t_f	sequential CPU time for function f
nwI	number of words of input message going to a child
nwO	number of words of output message coming from a child
Runtime system dependent parameters	
t_{create}	CPU time in a parent processor to create a child process
$t_{\#}$	CPU time in a child processor to create a new process
Architecture dependent parameters	
P	Number of processors
δ	latency of a message, from start sending to start receiving
λ	start-up fixed CPU cost for sending or receiving a message
β	per-word CPU cost for sending or receiving a message
Abbreviations	
$t_{unpackI} = \lambda + \beta.nwI$	CPU time for unpacking an input
$t_{unpackO} = \lambda + \beta.nwO$	CPU time for unpacking an output
$t_{packI} = \lambda + \beta.nwI$	CPU time for packing an input
$t_{packO} = \lambda + \beta.nwO$	CPU time for packing an output

Fig. 1. Parameters of the cost models

2.2 Simple Skeletons and Cost Models in Eden

Eden is a higher-order language where processes are explicitly handled. Thus, this language allows both to implement skeletons and to use them inside the same language. A detailed description of our library, including implementations of a wide repository of skeletons (map, divide & conquer, pipe, map & reduce, iterUntil, etc) can be found in [18], while a practical comparative study about the efficiency of the language can be found in [9].

In order to illustrate how skeletons are implemented in Eden, let us consider a simple example: A parallel map. This skeleton can be easily implemented by creating a new process for each element of the input list. The Eden implementation is as follows

```
par_map f xs = [pf # x | x <- xs ] `using` spine
  where pf = process x -> f x
```

where `` `using` spine `` forces the eager evaluation of the list of instantiations. In [16] simple cost models for some implementations of skeletons in Eden were presented. However, these cost models were somehow restricted. For instance, in the case of the previous implementation, the cost model assumed that the granularity of all the tasks is equal and that there is a equitable distribution of processes among processors. In fact, processes are distributed in a round-robin fashion. With these assumptions, and considering the parameters shown in Figure 1, the cost model is as follows:

$$\begin{aligned} t_{par\_map} &= L_{init} + t_{processor} + L_{final} \\ L_{init} &= P(t_{create} + t_{packI}) + \delta \\ L_{final} &= \delta + t_{unpackO} \\ t_{processor} &= \lceil \tfrac{N}{P} \rceil (t_{\#} + t_{unpackI} + t_f + t_{packO}) \end{aligned}$$

The first formula describes the critical path determining the parallel time. Before the last worker starts computing, P processes must be created and P messages must be packed in the parent. After the last worker finishes, an output message must arrive at the parent (hence the δ latency) and then it must be unpacked. The most heavily loaded processor will receive $\lceil \frac{N}{P} \rceil$ tasks. We are assuming that the remaining management costs do not belong to the critical path.

However, we think that the underlying assumptions can be improved. A useful alternative that allows us to improve the reliability of the cost model consists in using as parameters two probability distribution functions, or equivalently two random variables. First, we need to formalize the distribution of processes into processors. Second, we also need probability distribution functions for the granularity of processes. The first distribution is known, as it depends only on the runtime system[2]. However, the second distribution depends on the concrete problem. For instance, Dirac distributions can be used for processes with a known fixed granularity, while Poisson distributions can be used for processes whose behavior depends on the reception of data. In order to modify the existing cost models, we only need to reconsider how both the value of t_f and the maximum number of processes assigned to a processor are defined. These values will not be fixed numbers anymore but, as it was previously indicated, they will be given by random variables. Finally, we can *forget* most of the information provided by the cost model by considering only the expected value and the variance of the computed random variable. In this case, by applying well-known results on statistics, we can provide a *confidence interval* for the performance of the implementation. Thus, given a probability p, we will give an interval such that the real value of t_f belongs to this interval with probability higher than p.

3 Basic Notions on Statistics and Process Algebras

In this section we introduce the main concepts on statistics (e.g. random variables, probability distribution functions, expected value and variance) that we need in the next section. We also present the process algebra that we use to specify the performance of implementations. In contrast to the typical uses of process algebras, the aim of this one is just to compute a theoretical cost model. For that reason, it will be used only to specify the time needed to execute the processes, so that the actual actions performed by processes can be omitted. This time could be given by fixed amounts of time, but (as we have already explained in the previous sections) it is more realistic to assume that the time consumed by a certain process will not always be the same, as it will depend on certain factors that may vary in different executions. In this paper we will consider that this time is described by means of a random variable. By combining all the random variables describing the performance of each of the tasks we will be able to provide information such as "with probability p the execution time is less than or equal to t."

3.1 Preliminaries on Statistics

Next we introduce some concepts on random variables. We will consider that the sample space (that is, the domain of random variables) is given by the set of real numbers $\mathbb{R}$.

Definition 1. Let ξ be a random variable. We define its *probability distribution function*, F_ξ, as the function $F_\xi : \mathbb{R} \to [0,1]$ such as $F_\xi(x) = \mathrm{P}(\xi \leq x)$, where $\mathrm{P}(\xi \leq x)$ is the probability that ξ assumes values less than or equal to x.

Let ξ_1, ξ_2 be random variables. We say that ξ_1 and ξ_2 are *identically distributed*, denoted by $\xi_1 = \xi_2$, if for any $x \in \mathbb{R}$ we have $F_{\xi_1}(x) = F_{\xi_2}(x)$.

Let ξ be a random variable. We write $\mathtt{E}[\xi]$ to denote the *expected value* of ξ. We write $\mathtt{V}[\xi]$ to denote the *variance* of ξ.

We consider a *distinguished* random variable. By *unit* we denote a random variable such that $F_{unit}(t) = 1$ for any $t \geq 0$, that is, *unit* follows the *Dirac* distribution in 0. □

[2] In the case of Eden, the runtime system allows the user to choose between two strategies: Distributing processes in a round-robin fashion or distributing them randomly.

Given the fact that in our setting random variables are associated with time distributions, they will only take positive values in $\mathbb{R}^+$. That is, given a random variable ξ we have $F_\xi(t) = 0$, for any $t < 0$. We will call this type of random variables *temporal random variables*.

In order to compute performance of implementations we will have to put together the performance of the corresponding tasks. In the next definition we introduce some operators to produce these combined random variables.

Definition 2. Let $\xi_1, \ldots, \xi_n$ be independent random variables. We denote by $\sum \xi_i$ a random variable distributed as the sum of these random variables. In particular, we consider that $n \cdot \xi$, with $n \in \mathbb{N}$, is a shorthand for the addition of n independent random variables identically distributed as ξ.

Let $\xi_1, \ldots, \xi_n$ be independent random variables. We denote by $\max\{\xi_i\}$ a random variable distributed as the maximum of these random variables.

Let $p_1, \ldots, p_n \in (0, 1]$ be such that $\sum p_i = 1$, and $\xi_1, \ldots, \xi_n$ be independent random variables. We define the *weighted addition* of $(\xi_1, \ldots, \xi_n)$ with probabilities $(p_1, \ldots, p_n)$, denoted by $\bigoplus\{(\xi_i, p_i) \mid 1 \leq i \leq n\}$, as the random variable whose probability distribution function is the following:

$$F_{\bigoplus(\xi_i, p_i)}(t) = \sum_{i=1}^{n} p_i \cdot F_{\xi_i}(t)$$

The sum of random variables will be used to denote performance associated with the sequential execution of tasks. Thus, if we have to (sequentially) perform two tasks having performances given by ξ_1 and ξ_2 then the random variable $\xi = \xi_1 + \xi_2$ defines the global performance of the implementation. Let us note that for any temporal random variable ξ we have that $\xi + \mathit{unit} = \xi$. The maximum of random variables will be used to combine tasks that are performed in parallel. Thus, if we have to perform (in parallel) two tasks having performances given by ξ_1 and ξ_2 then the random variable $\xi = \max\{\xi_1, \xi_2\}$ defines the global performance of the implementation. Besides, *unit* is also a neutral element of this operation, that is, for any temporal random variable ξ we have that $\max\{\xi, \mathit{unit}\} = \xi$. The operator $\bigoplus(\xi_i, p_i)$ defines a random variable that results when we consider several alternatives, each of them with different probabilities of being performed, and each of them with different random variables defining their execution times.

We finish this presentation of basic concepts on random variables by recalling *Tchebyshev inequality* (see e.g. [6] for more details). This result allows us to *bound* the current value of a random variable.

Lemma 1. Let ξ be a random variable and let $\epsilon > 0$. Then, we have that $\mathrm{P}(\mid \xi - \mathtt{E}[\xi] \mid > \epsilon) \leq \frac{\mathtt{V}[\xi]}{\epsilon^2}$. Alternatively, we have $\mathrm{P}(\mid \xi - \mathtt{E}[\xi] \mid \leq \epsilon) \geq 1 - \frac{\mathtt{V}[\xi]}{\epsilon^2}$. □

Thus, we have that with a certain probability, the actual value of a random variable will belong to the interval $(\mathtt{E}[\xi] - \epsilon, \mathtt{E}[\xi] + \epsilon)$.

3.2 A simple Stochastic Process Algebra

Next we present the language that we use to describe the performance of the considered implementations.

Definition 3. Let $\mathcal{V}$ be a set of temporal random variables and $Id_\mathcal{P}$ be a set of process variables. The set of processes, denoted by $\mathcal{P}$, is given by the following EBNF-expression:

$$P ::= \mathtt{stop} \mid \xi \mid \; \|^n P_i \mid P \gg P \mid X := P$$

where $\xi \in \mathcal{V}$, $X \in Id_\mathcal{P}$, and $n > 0$. □

The term `stop` denotes a terminated process. A process as $P = \xi$, with $\xi \in \mathcal{V}$, indicates that the performance of P is given by the random variable ξ. Specifically, P will last at most t units of time with probability $P(\xi \leq t)$. The term $\|^n P_i$ represents a parallel composition of n processes P_i. If $n = 1$ then $\|^n P_i$ represents a single process running sequentially. The process $P \gg Q$ represents the sequential composition of two processes. Thus, $P \gg Q$ behaves as P until it finishes; afterwards it behaves as Q. Finally, $X := P$ denotes the definition of a (possibly recursive) process.

In order to avoid side effects we will suppose that all the random variables appearing in the definition of a process are independent. In particular, the same random variable cannot appear twice in the definition of a process. Note that this restriction does not imply that we cannot have identically distributed random variables (as far as they have different names). Anyway, for the sake of convenience, we will sometimes use the same random variable several times in a single process. For example, two occurrences of the same random variable ξ is a shorthand to indicate that these two stochastic-time values are given by two independent random variables, ψ_1 and ψ_2, both of them being identically distributed as ξ.

The next natural step in the presentation of a process algebra is the definition of an operational semantics. However, in our setting we do not need to provide one since we are interested only in the global performance of a process. These quantities (i.e. the corresponding random variables) will be computed by studying the structural definition of processes. Let us remark that, given the simplicity of our language, we do not need the usual complicated mechanisms to deal with general distributions in the context of stochastic process algebras (see e.g. [5, 11, 2]). Actually, the underlying operational model is similar to that used in [12] but interpreting different alternative branches as parallel composition (instead of choice).

4 Cost Models in Terms of Random Variables

In this section we show how cost models including stochastic information can be computed. Our cost models take into account the number of available processors. Thus, we need to consider how processes are distributed among processors.

Definition 4. Let P be a process, n the number of tasks generated by P and m the number of available processors. Let $g : [1, n] \longrightarrow [1, m]$ be a total function that assigns each task to each processor. We denote by $\mathcal{T}(P, g)$ the random variable expressing the performance of P for the distribution of tasks given by g.

Besides, we define *the number of task assigned to processor* k by g, denoted by $y_{g,k}$, as

$$y_{g,k} = |\{i \mid g(i) = k, 1 \leq i \leq n\}|$$

□

Next we illustrate our methodology by means of an example: A parallel map. In addition, we will sketch how we can apply these ideas to a cost model for a divide & conquer skeleton expressed in terms of the parallel map.

4.1 Parallel Map

We consider an implementation that creates a process to evaluate each of the elements of the input list. That is, if the input list has n elements, we will create n processes. We will also consider that the time needed by each task to be performed is governed by a certain probability distribution function. Thus, the implementation will be given as a parallel composition of a certain number of processes whose execution time is given by random variables identically distributed.

Definition 5. Let us consider the following Eden implementation of the map skeleton:

```
par_map f xs = [pf # x | x <- xs ] `using` spine
     where pf = process x -> f x
```

Let n be the length of `xs` and let $\xi \in \mathcal{V}$ be the random variable specifying the performance of each task. The implementation of this skeleton in our stochastic process algebra, denoted by `parmap` (ξ, n), is given by the process $\|^n P_i$, where for any i we have $P_i = \xi_i$, with $\xi_i \in \mathcal{V}$, being the random variables $\xi_1, \ldots, \xi_n$ independent and identically distributed as ξ. □

Given the previous implementation (whose temporal information is given by the process `parmap` (ξ, n)) we may consider several random variables indicating its performance under different considerations. Regarding the general case, we will consider two particular situations. First, we will give a cost model for a given distribution of processes. Second, we will consider the method that distributes tasks among processors randomly but in an equiprobable way. That is, the probability for each process to be assigned to a given processor is the same.

Lemma 2. Let $n, m \in \mathbb{N}$ and $\xi \in \mathcal{V}$. Let us consider the process $P =$ `parmap` (ξ, n) and the total function $g : [1, n] \longrightarrow [1, m]$ that assigns each task to each processor. We have

$$\begin{aligned} \mathcal{T}(P, g) &= \max\{y_{g,k} \cdot \xi \mid 1 \leq k \leq m\} \\ &= \max\{y_{g,k} \mid 1 \leq k \leq m\} \cdot \xi \end{aligned}$$

□

The independence of the involved random variables allows us to easily compute both the expected value and the variance of the random variable associated with the performance of our implementation.

Proposition 1. Let $n, m \in \mathbb{N}$, and $\xi \in \mathcal{V}$. Let $g : [1, n] \longrightarrow [1, m]$ be the total function that assigns each task to each processor. Finally, let us consider the process $P =$ `parmap` (ξ, n). The expected value of $\mathcal{T}(P, g)$ is given by the following formula:

$$\mathtt{E}[\mathcal{T}(P, g)] = \max\{y_{g,k} \mid 1 \leq k \leq m\} \cdot \mathtt{E}[\xi]$$

while the variance is computed as follows:

$$\mathtt{V}[\mathcal{T}(P, g)] = (\max\{y_{g,k} \mid 1 \leq k \leq m\})^2 \cdot \mathtt{V}[\xi]$$

Proof: By applying the definition of expected value we have

$$\mathtt{E}[\mathcal{T}(P, g)] = \mathtt{E}[\max\{y_{g,k} \mid 1 \leq k \leq m\} \cdot \xi]$$

Let us consider $m_g = \max\{y_{g,k} \mid 1 \leq k \leq m\}$. We have that $m_g \cdot \xi = \sum_{i=1}^{m_g} \xi$. That is, the sum of m_g independent random variables identically distributed. As they are independent, $\mathtt{E}[m_g \cdot \xi] = m_g \mathtt{E}[\xi]$.

In the case of the variance, the reasoning is done in a similar way. By its definition, we have that $\mathtt{V}[\mathcal{T}(P, g)] = \mathtt{V}[\max\{y_{g,k} \mid 1 \leq k \leq m\} \cdot \xi]$ while $\mathtt{V}[m_g \cdot \xi] = m_g^2 \, \mathtt{V}[\xi]$. □

Before, we have assumed that tasks are distributed among processors by means of a given function. Let us suppose now that the concrete definition of this function is unknown. To make a study of the time used to perform P, we will consider that each processor is *equiprobable*. That is, if we have m processors then the probability of a generated task to be associated with a given processor is always equal to $\frac{1}{m}$.[3] Then, the probability of each assignment of tasks to processors is given by a *multinomial distribution.*

[3] Note that this is one of the distribution strategies used in Eden.

Definition 6. Let $n, m \in \mathbb{N}$. We will denote by C_n^m the set

$$C_n^m = \left\{ (y_1, \ldots, y_m) \;\middle|\; \forall\, 1 \leq k \leq m :\; y_k \in \mathbb{N} \;\wedge\; \sum y_k = n \right\}$$

Given $\overline{y} \in C_n^m$ we denote the maximum component of $\overline{y}$ by $\max(\overline{y})$.

A probability distribution function F follows a *multinomial distribution* if the following conditions hold: (1) F is a discrete distribution; (2) the outcomes are discrete; (3) F is a generalization of the binomial distribution from only 2 outcomes to k outcomes; (4) individual trials are independent; and (5) outcomes are mutually exclusive and all inclusive.

Let $\overline{y} \in C_n^m$. The probability distribution function of a multinomial distribution for m equiprobable outcomes, denoted by $Prob(\overline{y} \mid n, \frac{1}{m})$, is defined as

$$Prob\left(\overline{y} \mid n, \frac{1}{m}\right) = \frac{n!}{y_1! \cdot y_2! \cdots y_m!} \cdot \left(\frac{1}{m}\right)^n$$

Let P be a process generating n tasks and m be the number of available processors. We denote by $\mathcal{T}(P, C_n^m)$ the random variable expressing the performance of P for the distribution of tasks following a multinomial distribution. □

The set C_n^m contains all the possibilities of distributing n elements among m components. As we have previously commented, the random distribution of tasks among processors, in an equiprobable way, generates a multinomial distribution. That is, the characteristics shown in the previous definition are fulfilled. First, the distribution must be discrete as well as the outcomes (that is, each processor will receive a discrete number of processes). Let us note that if we have 2 processors then we get a binomial distribution. Moreover, when assigning a new task to a processor we do not take into consideration the previous assignments. Finally, a process cannot be assigned to two different processors (outcomes are mutually exclusive) and every task has to be assigned to a processor (all inclusive). Let us remark that, owing to this fact, if we know the number of processes assigned to $m - 1$ processors then we will also know the number of processes associated with the remaining processor.

Taking into account the previous *distribution* of tasks to processors and combining it with the random variables associated with the corresponding tasks we obtain the following result.

Lemma 3. Let $n, m \in \mathbb{N}$ and $\xi \in \mathcal{V}$. Let us consider the process $P = \texttt{parmap}\ (\xi, n)$. We have

$$\mathcal{T}(P, C_n^m) = \bigoplus_{\overline{y} \in C_n^m} \left\{ (p, \xi') \;\middle|\; p = Prob\left(\overline{y} \mid n, \tfrac{1}{m}\right) \;\wedge\; \xi' = \max(\overline{y}) \cdot \xi \right\}$$

Besides, the probability of performing the process P before a certain time t has passed is given by the following formula:

$$Prob(P, t) = \sum_{\overline{y} \in C_n^m} Prob\left(\overline{y} \mid n, \frac{1}{m}\right) \cdot P(\max(\overline{y}) \cdot \xi \leq t)$$

□

As in the case where the distribution of processes to processors was known, we can compute both the expected value and the variance of the previous random variable. Once we have these values we may apply again Tchebyshev inequality to give a confidence interval for the value of the random variable.

Proposition 2. Let $n, m \in \mathbb{N}$ and $\xi \in \mathcal{V}$. Let us consider the process $P = \texttt{parmap}\ (\xi, n)$. Let ψ be distributed as $\mathcal{T}(P, C_n^m)$, that is

$$\psi = \bigoplus_{\overline{y^i} \in C_n^m} \left\{ (p_i, \xi_i) \;\middle|\; p_i = Prob\left(\overline{y^i} \mid n, \tfrac{1}{m}\right) \ \wedge\ \xi_i = \max(\overline{y^i}) \cdot \xi \right\}$$

The expected value of ψ is:

$$\mathtt{E}[\psi] = \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot m_i \cdot \mathtt{E}[\xi]$$

while the variance of ψ is:

$$\mathtt{V}[\psi] = \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot m_i^2 \cdot \mathtt{E}[\xi^2] - \left(\sum_{(p_i, m_i) \in PC_n^m} p_i \cdot m_i \cdot \mathtt{E}[\xi] \right)^2$$

where

$$PC_n^m = \left\{ (p_i, m_i) \;\middle|\; p_i = Prob\left(\overline{y^i} \mid n, \tfrac{1}{m}\right) \ \wedge\ \overline{y^i} \in C_n^m \ \wedge\ m_i = \max(\overline{y^i}) \right\}$$

Proof: The expected value of a random variable is defined by the formula $\mathtt{E}[\psi] = \int_{\mathbb{R}^+} t \cdot f_\psi(t) dt$, where f_ψ is the mass/density probability function associated with ψ. In the discrete case, function f_ψ is defined as $\sum_{(p_i, m_i) \in PC_n^m} p_i \cdot f_{\xi_i}$. Thus, applying the definition of expected value to this function, we obtain:

$$\begin{aligned}
\mathtt{E}[\psi] &= \int_{\mathbb{R}^+} t \cdot \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot f_{\xi_i}(t)\ dt \\
&= \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot \int_{\mathbb{R}^+} t \cdot f_{\xi_i}(t)\ dt \\
&= \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot \mathtt{E}[\xi_i]
\end{aligned}$$

Besides, the expected value of each random variable ξ_i is given by $\mathtt{E}[\xi_i] = m_i \cdot \xi$. Thus, we conclude

$$\mathtt{E}[\psi] = \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot m_i \cdot \mathtt{E}[\xi]$$

The variance of ψ is given by the expression $\mathtt{V}[\psi] = \mathtt{E}[\psi^2] - (\mathtt{E}[\psi])^2$, where $\mathtt{E}[\psi^2]$ is as follows:

$$\begin{aligned}
\mathtt{E}[\psi^2] &= \int_{\mathbb{R}^+} t^2 \cdot f_\psi(t)\ dt \\
&= \int_{\mathbb{R}^+} t^2 \cdot \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot f_{\xi_i}(t)\ dt \\
&= \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot \int_{\mathbb{R}^+} t^2 \cdot f_{\xi_i}(t)\ dt \\
&= \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot \mathtt{E}[\xi_i^2]
\end{aligned}$$

Besides, the expected value of ξ_i^2 is given by $\mathtt{E}[\xi_i^2] = m_i^2 \cdot \mathtt{E}[\xi^2]$ as each ξ_i is the sum of m_i independent random variables identically distributed as ξ. Thus,

$$\mathtt{E}[\psi^2] = \sum_{(p_i, m_i) \in PC_n^m} p_i \cdot m_i^2 \cdot \mathtt{E}[\xi^2]$$

Finally, by including the values obtained for $\mathtt{E}[\psi_i^2]$ and $\mathtt{E}[\psi]$ in the definition of $\mathtt{V}[\psi]$ we get the desired result. □

4.2 Parallel Divide & Conquer

Let us now briefly sketch how to deal with the divide & conquer skeleton. We will use an implementation where a parameter `n` is used to determine how many levels of the divide & conquer tree are to be performed in parallel. Thus, if the value `n` is `0` means that the problem is to be solved sequentially, while the value `n` is greater than `0` means that each of the subproblems is to be solved by using an independent process. The Eden implementation is as follows:

```
par_dc 0 trivial solve split combine x
  = seq_dc trivial solve split combine x
par_dc n trivial solve split combine x
  | trivial x   = solve x
  | otherwise   = combine x c
 where c = par_map (par_dc (n-1) trivial solve split combine) (split x)
```

As in the previous case we will specify the time performance of this implementation in our process algebra, assuming that the depth of the tree of processes will be given by a value n. The main structure of each (recursive) process consists of the sequential composition of three steps: *splitting* the input problem into subproblems, *solving* in parallel each of the subproblems (recursive call), and *combining* the results of the subproblems.

In order to present the performance details of the implementation we will take advantage of our previous study for parallel map.

Definition 7. Let $\xi, \texttt{split}, \texttt{combine} \in \mathcal{V}$ be random variables expressing the performance time for sequential execution, splitting, and combining results, respectively, $n \in \mathbb{N}$ be the depth of the tree of processes, and $k \in \mathbb{N}$ the number of subprocesses generated by each recursive call. The implementation of the divide & conquer skeleton in our stochastic process algebra, denoted by $\texttt{d\&c}(\xi, \texttt{split}, \texttt{combine}, n, k)$ is given by the process P_n recursively defined as:

$$\begin{aligned} P_0 &:= \xi \\ \underset{i>0}{P_i} &:= \texttt{split} \gg \texttt{parmap}(P_{i-1}, k) \gg \texttt{combine} \end{aligned}$$

□

Let us remark that, for the sake of clarity, in the previous definition we have ignored the fact that a process P_n can be assigned to solve a trivial task. In that case, the actual implementation does not need to create any subprocess. However, when computing the costs we will assume that this (faster) case is not frequent.

In order to compute the cost of the implementation we can unfold the previous recursive definition, so that the n levels of the tree of processes appear explicitly. By doing so, we can compute the global cost as the sum of three steps. In the first step, the whole tree of processes is created. Then, all the sequential processes are solved. Finally, the combination of results take place.

Lemma 4. Let $\xi, \texttt{split}, \texttt{combine} \in \mathcal{V}$ be random variables and $n, k \in \mathbb{N}$. Let us consider $\texttt{d\&c}(\xi, \texttt{split}, \texttt{combine}, n, k)$. Finally, let $g : [1, k^n + 2 \cdot \sum_{i=1}^{n-1} k^i] \longrightarrow [1, m]$ be the total function that assigns each task to each processor. We have

$$\begin{aligned} \mathcal{T}(P, g) = \mathcal{T}(\texttt{parmap}(\texttt{split}, \textstyle\sum_{i=1}^{n-1} k^i), g) + \\ \mathcal{T}(\texttt{parmap}(\xi, k^n), g) + \\ \mathcal{T}(\texttt{parmap}(\texttt{combine}, \textstyle\sum_{i=1}^{n-1} k^i), g) \end{aligned}$$

□

Given the fact that the random variables associated with each sequential step are independent, the expected value of the random variable will be obtained by adding the expected values of each step. Analogously, the variance of the sum of several independent random variables is also obtained by adding the corresponding variances. Finally, let us recall that in this case we can also use Tchebyshev inequality to obtain a confidence interval.

5 Conclusions and Future Work

In this paper we have shown how cost models for implementations of skeletons can be defined in terms of random variables. By using this approach we do not get a unique performance time but a random variable indicating, for each time t, the probability with which the implementation has finished before time t. In order to illustrate our approach we have presented the application of our method to the parallel map skeleton and we have sketched how cost models for the divide & conquer skeleton can be computed.

As future work we plan to apply our methodology to more complex skeletons. A preliminary study with such skeletons (e.g. we have already performed the study for a pipeline) has shown that increasing complexity of skeletons does not extraordinarily increase the difficulty of the performance models.

References

1. G. H. Botorog and H. Kuchen. Efficient high-level parallel programming. *Theoretical Computer Science*, 196(1–2):71–107, 1998.
2. M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.
3. M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computations*. MIT Press, 1989. Research Monographs in Parallel and Distributed Computing.
4. M. Cole. Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. *Parallel Computing*, 30:389–406, 2004.
5. P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. General purpose discrete-event simulation using ♠. In *6th International Workshop on Process Algebra and Performance Modelling*, pages 85–102, 1998.
6. W. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, 1973.
7. M. Hamdan. *A Combinational Framework for Parallel Programming Using Algorithmic Skeletons*. PhD thesis, Department of Computing and Electrical Engineering. Heriot-Watt University, 2000.
8. U. Klusik, R. Loogen, S. Priebe, and F. Rubio. Implementation skeletons in Eden: Low-effort parallel programming. In *Implementation of Functional Languages, IFL'00, LNCS 2011*, pages 71–88. Springer, 2001.
9. H.W. Loidl, F. Rubio, N. Scaife, K. Hammond, S. Horiguchi, U. Klusik, R. Loogen, G. Michaelson, R. Peña, S. Priebe, A. Rebón Portillo, and P. Trinder. Comparing parallel functional languages: Programming and performance. *Higher-Order and Symbolic Computation*, 16(3):203–251, 2003.
10. R. Loogen, Y. Ortega-Mallén, R. Peña, S. Priebe, and F. Rubio. Parallelism abstractions in Eden. In F. A. Rabhi and S. Gorlatch, editors, *Patterns and Skeletons for Parallel and Distributed Computing*, pages 95–128. Springer-Verlag, 2002.
11. N. López and M. Núñez. `NMSPA`: A non-markovian model for stochastic processes. In *International Workshop on Distributed System Validation and Verification (DSVV'2000)*, pages 33–40, 2000.

12. N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *CONCUR 2001, LNCS 2154*, pages 321–335. Springer, 2001.

13. N. López, M. Núñez, and F. Rubio. Stochastic process algebras meet Eden. In *Integrated Formal Methods 2002, LNCS 2335*, pages 29–48. Springer, 2002.

14. G. Michaelson, N. Scaife, P. Bristow, and P. King. Nested algorithmic skeletons from higher order functions. *Parallel Algorithms and Applications special issue on High Level Models and Languages for Parallel Processing*, 16(2-3):181–206, 2001.

15. S. Pelagatti. *Structured Development of Parallel Programs*. Taylor and Francis, 1998.

16. R. Peña and F. Rubio. Parallel functional programming at two levels of abstraction. In *Principles and Practice of Declarative Programming, PPDP'01*, pages 187–198. ACM Press, 2001.

17. S. L. Peyton Jones and J. Hughes. Report on the programming language Haskell 98. Technical report, February 1999.

18. F. Rubio. *Programación Funcional Paralela Eficiente en Eden*. PhD thesis, Dpto. Sistemas Informáticos y Programación. Universidad Complutense de Madrid (Spain), November 2001. In Spanish.