Reprinted from

PARALLEL
COMPUTING

Parallel Computing 20 (1994) 1583–1603

Performance analysis of parallel programs based on model calculations

Georg Fleischmann [a,*], Matthias Gente [b], Fridolin Hofmann [b], Gunter Bolch [b]

[a] *Academy of Media Arts, Dept. of Computer Science / Audio-Visual Media, Peter-Welter-Platz 2, D-50676 Cologne, Germany*

[b] *University of Erlangen, Dept. of Computer Science IV, Martensstraße 1, D-91058 Erlangen, Germany*

Received 21 May 1993

N·H
ELSEVIER

Parallel Computing

Editorial policy. *Parallel Computing* is an international journal presenting the theory and use of parallel computer systems, including vector, pipeline, array, fifth and future generation computers and neural computers. Within this context the journal covers all aspects of high-speed computing.

Its primary objectives are:

- publication of new research results pertaining to the development and application of parallel computer systems;
- dissemination of information on new developments and events in this field.

The editorial policy and the technical content of the journal are the responsibility of the Editors and the Editorial board. The journal is self-supporting from subscription income and contains a minimum amount of advertisements. Advertisements are subject to the prior approval of the Editors.

Scope. *Parallel Computing* features original research work, tutorial and review articles as well as accounts on practical experience with and techniques for the use of parallel computers. Contributions can cover:

- algorithm design for all types of parallel computers;
- all aspects of the application of parallel computers, including those to AI, CAD/CAM, databases, information retrieval, etc. in industrial, business and office environments;
- the impact of high-speed computation on current research, as well as the advancement of knowledge;
- software engineering aspects relating to parallel computing;
- software for parallel computer systems including programming languages, operating systems, utilities, libraries, etc.;
- networking technology for support of high-speed computing via centralised file servers, output servers, interactive access, etc.;
- taxonomy, models and architectural trends of parallel processing;
- general hardware (architecture) concepts, new technologies enabling the realisation of such new concepts as well as details of commercially available systems;
- performance measurement results on state-of-the-art systems;
- peripheral devices for parallel (super-)computers.

(∞) The paper used in this publication meets the requirements of ANSI/NISO 239.48-1992 (Permanence of Paper).

0167-8191/94/$07.00

Printed in The Netherlands

ELSEVIER

Parallel Computing 20 (1994) 1583–1603

PARALLEL COMPUTING

Performance analysis of parallel programs based on model calculations

Georg Fleischmann [a,*], Matthias Gente [b], Fridolin Hofmann [b], Gunter Bolch [b]

[a] *Academy of Media Arts, Dept. of Computer Science / Audio-Visual Media, Peter-Welter-Platz 2, D-50676 Cologne, Germany*

[b] *University of Erlangen, Dept. of Computer Science IV, Martensstraße 1, D-91058 Erlangen, Germany*

Received 21 May 1993

Abstract

Modeling parallel programs can help the developer during the design phase of a particular implementation on the one hand and on the other hand provide principle insights which are needed to establish design principles for the development of parallel programs. It is important that the model used provides a sufficient but not too detailed representation of the parallel program.

Precedence graphs provide an easy to understand representation of the structure of parallel programs. Information about the behavior of parallel programs during execution can be obtained by assigning distributions of the execution times of the subtasks to the nodes of these graphs.

In this paper we introduce truncated *θ-exponential polynomials* as a class of distributions suitable for the modeling of task run-times of massively parallel programs. As opposed to classical exponential polynomials known from literature, truncated θ-exponential polynomials allow the representation of distributions with finite domain and are therefore appropriate for the analysis of massively parallel systems.

We present approximation formulas which allow the approximate calculation of the total execution time for large models in particular. We provide formulas for the parallel execution of a large number of subtasks. We also give a method to estimate the total execution time of *regular* graphs, which are also very important in the field of modeling parallel programs.

* Corresponding author.

SSDI 0167-8191(94)00049-G

Keywords: Parallel computing; Performance analysis; Modeling; Precedence graph; Exponential polynomial; Massively parallel system; Extreme value distribution; Static scheduler; Dynamic scheduler; Regular structure

1. Introduction

The primary reason for employing parallel user computations is to reduce the overall processing time of problems which require a large amount of computation. Therefore, the processing time of parallel programs is an important criterion when choosing a particular implementation. During the development phase of a parallel program, information about the execution times can only be obtained from calculations based on models. Another reason for modeling parallel programs is to develop principle insights, which are needed to establish design methodologies for the development of parallel programs.

The analytical investigations presented in this paper were part of the SUPRENUM project. Such analytical investigations enable the quantitative performance analysis already during the design phase of multiprocessor systems like the SUPRENUM. For this reason a part of the SUPRENUM project was devoted to the modeling and performance analysis of parallel systems.

In this paper we deal with the performance prediction of large parallel programs. We introduce a new class of distributions for the representation of task run-times which is especially appropriate for the modeling of massively parallel systems. In Section 4 we present approximation formulas for massively parallel systems and in Section 5 approximation formulas for the mean execution time of iterative structures are developed.

Precedence graphs are the most appropriate tool to use when representing the functional structure of parallel programs. This is especially the case when one is interested in the principle investigations which are performed in Section 4 and Section 5 of this paper. The nodes of precedence graphs represent the subtasks of a parallel program and the edges represent the precedence relations between these subtasks. Precedence graphs are very popular because of their comprehensibility and are used by many authors [1–5].

The exact execution times of the subtasks of the distributions of these execution times must be known in order to be able to represent the behavior during program execution with precedence graphs. We will show in Section 3 that phase-type distributions such as exponential, Erlang or hyperexponential distributions, which are often used in performance evaluation, are not appropriate for the representation of typical task run-times. This is due to the fact that typical task run-times are characterized by a small coefficient of variation and thus need a large number of phases to be described by phase-type distributions. The class of distributions presented in this paper makes not only the efficient representation of typical task run-times possible, but also the guarantee of an upper limit for the execution time. This latter property is especially important in the modeling of massively parallel programs, as we will show in Section 4.1.

In Section 4 we investigate a fundamental structure in the field of parallel computing: The parallel execution of independent subtasks of a parallel program. We present approximation formulas for the case in which the number of available processors is unlimited. These formulas are based on the limit distributions of θ-exponential polynomials and truncated θ-exponential polynomials. When the number of processors is limited we have to define a strategy according to which the tasks are assigned to the available processors. We distinguish between static and dynamic scheduling and derive approximation formulas for both cases. Using these formulas we show that the dynamic scheduler causes shorter execution times when communication times are disregarded but may cause longer times when communication has to be taken into account.

Often a parallel computation consists of stages in which the tasks can be executed concurrently. Between the tasks of one stage there is no communication or synchronization and the precedence relations between two successive stages obey the same pattern. In Section 5 we define the class of *regular* graphs, which is used to represent this kind of computations. The investigation of numerous examples leads to the assumption that the expectation of the total execution time of such graphs grows by a fixed value whenever a new stage is added. Using this *stationarity* of regular graphs it is possible to analyze large task structures by the extrapolation of results for small graphs.

2. Modeling of parallel programs

Modeling techniques for the performance evaluation of parallel systems can be devided into four main classes: Fork-join queueing networks, timed Petri nets, precedence graphs and combinations of two or more of the aforementioned classes (see Fig. 1).

Fork-join queueing networks are classical queueing models extended by additional constructs used to model splitting (fork construct) of one task into several subtasks which can be computed concurrently and to model merging (join construct) of multiple subtasks into one [6,7]. Fork-join queueing models are mainly used for the analysis of parallel tasks, which are repeatedly processed by the same hardware.

With timed Petri nets [8] the possibility of a detailed representation of the structure of a parallel, program can be provided, as well as a detailed description of the hardware which is used to process the program. The main disadvantage of Petri nets is that even small problems lead to large and complex models. Petri nets are very important for the modeling of hardware and operating system properties but precedence graphs are more appropriate when the point of interest lies in the properties of application programs.

Precedence graphs – referred to below as task-structures – are easy to understand and are thus widely used for modeling parallel programs [1–5,9,10]. The nodes of precedence graphs represent the subtasks of a parallel program. The

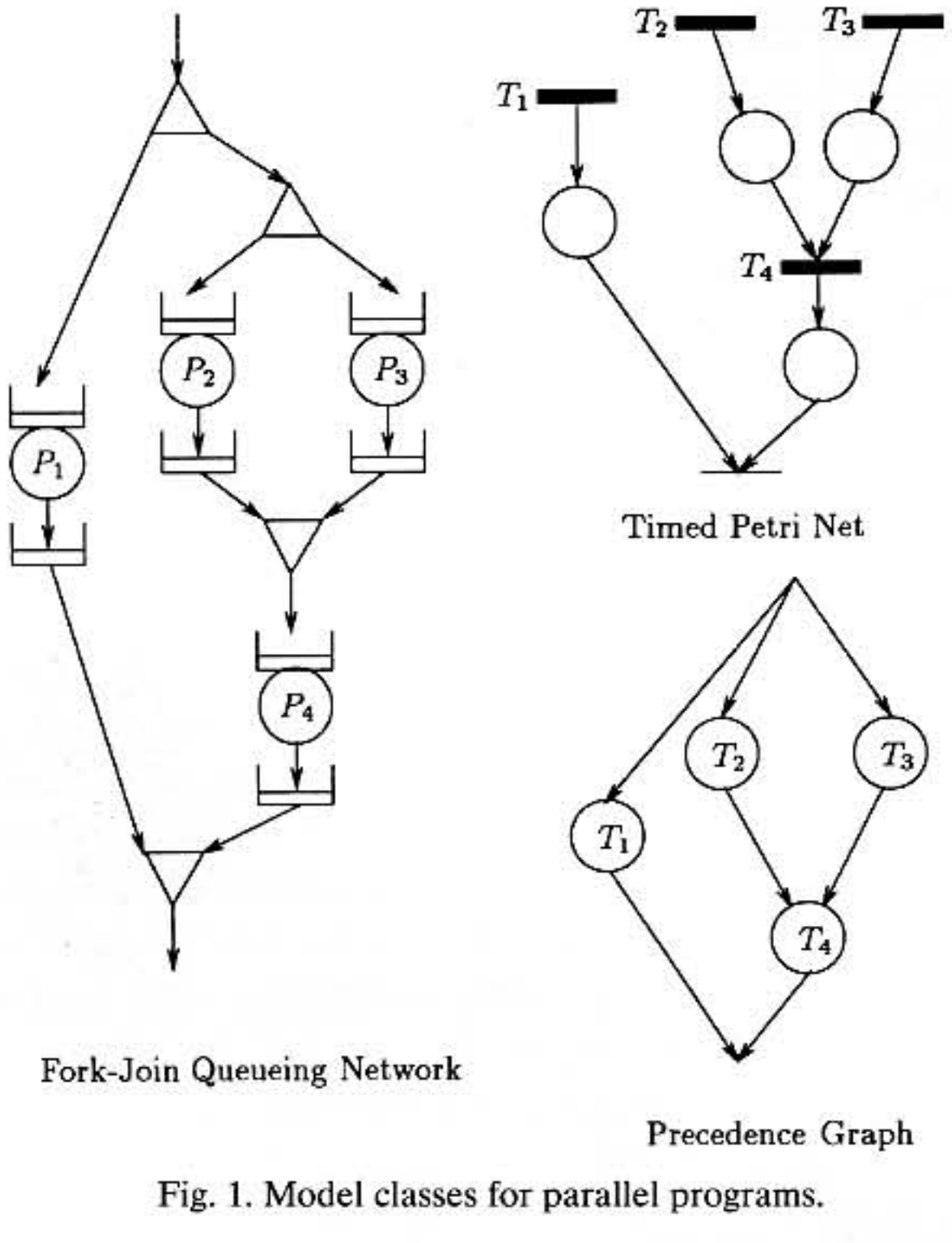


Fig. 1. Model classes for parallel programs.

dependencies between the subtasks are modelled by the edges of the graph. System constraints such as a finite number of processors or communication requirements can be taken into account. A simple example is shown in Fig. 2.

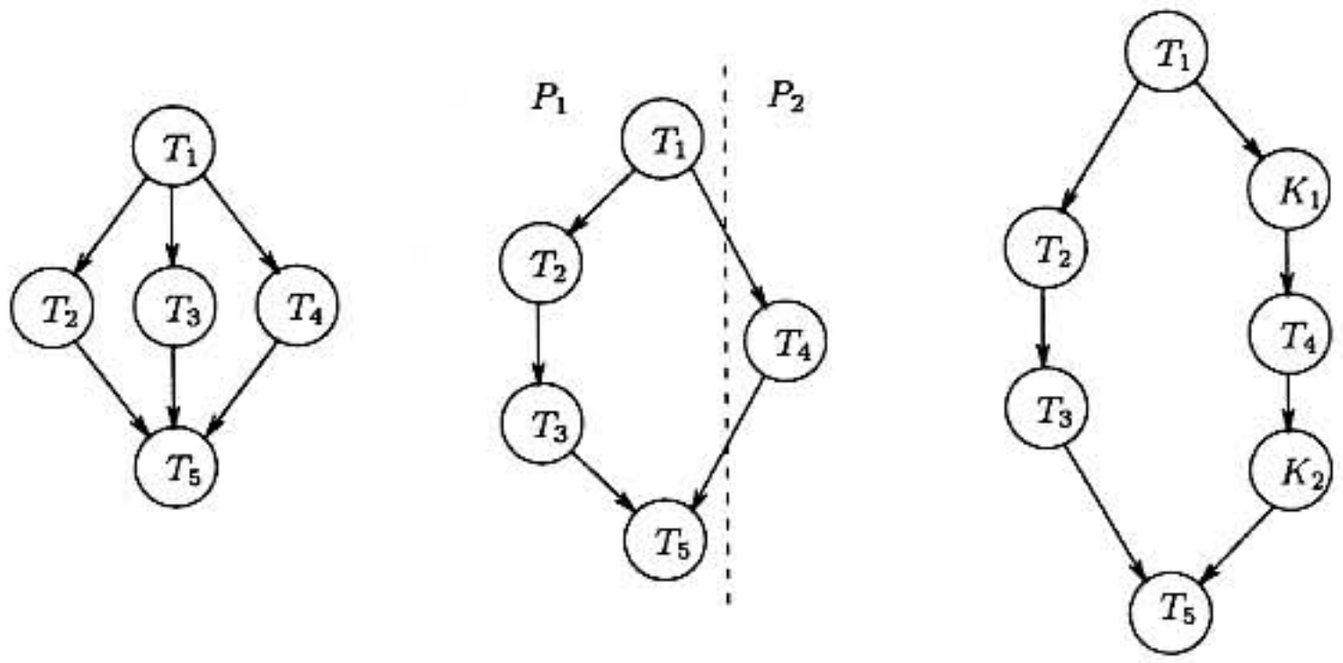


Fig. 2. Examples of precedence graphs.

Graph (a) in Fig. 2 is the model of the parallel algorithm. Graph (b) is obtained by assigning the subtasks to the processors P_1 and P_2. The communication requests are modelled by the nodes K_1 and K_2 in graph (c). If random variables are assigned to the subtask execution-times, then the total execution time of the program can be calculated. It corresponds to the time required to traverse a directed acyclic graph.

In the performance evaluation tool *GIANT* [1] several exact analysis methods have been provided which can be used to predict the total execution time of small to medium sized task-structures. But the size of models which can be analyzed by exact methods is limited due to memory constraints and numerical inaccuracies. To predict the behavior of systems with a large number of subtasks we need approximation formulas. Such formulas are presented in Sections 4 and 5.

GIANT can be used to investigate both series-parallel graphs and more general, non series-parallel graphs. The advantage of series-parallel graphs is that the time needed to analyze them is linear or polynomial [11]. The analytical operations of convolution, multiplication, addition, integration, and differentiation, applied to the distribution function, are sufficient to compute the total execution time. If T_1, $T_2,\ldots,T_n$ are nodes of a directed acyclic graph with independent random variables X_1, $X_2,\ldots,X_n$ and distribution functions F_{X_1}, $F_{X_2},\ldots,F_{X_n}$, then:

(a) The value X_{sum} obtained from the serial arrangement of T_1, $T_2,\ldots,T_n$ is $X_{\text{sum}} = \sum_{i=1}^{n} X_i$ with the distribution function

$$F_{sum}(t) = \int_0^t \left(\otimes_{i=1}^{n} \frac{\mathrm{d}}{\mathrm{d}x} F_{X_i}(x) \right) \mathrm{d}x$$

where $\otimes$ is the convolution operator.

(b) The value $X_{\max}$ of the parallel arrangement of T_1, $T_2,\ldots,T_n$ is $X_{\max} = \max_{1 \le i \le n}\{X_i\}$ with the distribution function $F_{\max}(t) = \prod_{i=1}^{n} F_{X_i}(t)$.

Classes of functions which are closed under these operations, and are thus particularly suitable for modeling distributions of execution times using computers, are presented in Section 3.

If pure graph models are used it is possible only to model static scheduling mechanisms and communication strategies. The appropriate parameters are included in the model on which *GIANT* is based so that dynamic allocation strategies can also be described [12].

3. Modeling the execution times of parallel programs

In this section we describe classes of continuous distribution functions for the description of the run-times of parallel programs. These classes of distribution

[1] Graphoriented Interactive ANalysis Tool.

functions make the computation of the total execution times of graph models in a semi-symbolic form on a computer possible. Our goal is a class of distribution functions which is particularly suitable for the representation of typical task run-times and which makes the efficient calculation of the total execution time of parallel programs possible.

Sahner and Trivedi introduced the class of exponential polynomials $EP(t) = \sum_{i=1}^{n} a_i t^{k_i} e^{-\lambda_i t}$ with $a_i \in C$, $\lambda_i \in C$, $k_i \in N$ to model task execution times in series-parallel task graphs [4,11]. Exponential polynomials can be represented by their parameter triples (a_i, k_i, λ_i). Phase-type distributions, such as exponential, Erlang or hyperexponential distributions, which are often used in the field of performance evaluation of hardware architectures and operating systems, can be represented by exponential polynomials. For example the distribution function $F_\lambda(t) = 1 - e^{-\lambda t}$ of an exponential distribution can be described by the two parameter triples (1, 0, 0) and (1, 0, λ). Because exponential polynomials are closed under the operations which must be performed to analyze series-parallel graphs, the results of these operations can also be represented by a set of parameter-triples. This latter fact makes the calculation of the total execution time of task-graphs in a semi-symbolic form on a computer possible. We demonstrate this with the following example, in which the distribution function $F_{\max}(t) = 1 - e^{-\lambda_1 t} - e^{-\lambda_2 t} + e^{-(\lambda_1+\lambda_2)t}$ of the maximum of two exponential distributions with distribution functions $F_{\lambda_1}(t)$ and $F_{\lambda_2}(t)$ is computed:

$F_{\lambda_1}(t)$	$F_{\lambda_2}(t)$	$F_{\max}(t)$
(1, 0, 0)	(1, 0, 0)	(1, 0, 0)
(1, 0, λ_1)	(1, 0, λ_2)	(−1, 0, λ_1)
		(−1, 0, λ_2)
		(1, 0, $\lambda_1 + \lambda_2$)

Even this small example indicates that the number of parameters increases drastically when the operations are performed. Task execution times typically have distributions with small coefficients of variation which can only be represented by exponential polynomials with a large number of parameter triples. The effect of performing the analysis operations on such exponential polynomials is disastrous because the number of parameters increases drastically.

The class of θ-exponential polynomials introduced in the next section makes the representation of typical task run-times by a small number of parameters possible, and is thus more appropriate in the modeling of parallel programs than pure exponential polynomials.

The task execution-times of parallel programs are limited. Exponential polynomials and θ-exponential polynomials are distributions with infinite domains and thus cannot represent this observed behavior. In Section 3.2 we introduce the class of truncated θ-exponential polynomials which have finite domains. We will show in Section 4.1 that only distributions with finite domains are appropriate for the modeling of massively parallel programs.

3.1. θ-exponential polynomials

The extension of the class of exponential polynomials mentioned in the previous section is presented in greater detail in this section. A function $H: R_0^+ \to R_0^+$ of the form

$$H(t) = \sum_{i=1}^{n} H_i(t) \quad \text{with} \quad H_i(t) = \begin{cases} a_i(t-\theta_i)^{k_i} e^{-\lambda_i(t-\theta_i)} & \text{for} \quad t \geq \theta_i \\ 0 & \text{for} \quad t < \theta_i \end{cases}$$

with $a_i \in R$, $\lambda_i \in R_0^+$, $k_i \in N_0$, $\theta_i \in R$ is called a θ-exponential polynomial. The parameter θ_i is called the deterministic part, a_i the coefficient, λ_i the rate, and k_i the stage.

If the parameters $c_i := a_i k_i! / \lambda_i^{k_i+1}$ are all positive and the θ_i are all zero, then we obtain the well-known representation of generalised Erlang density functions, sometimes referred to in literature as mixed Erlang distributions, e.g. in [13]. In general the parameters c_i can be any real number. If the value c_i is outside the interval [0,1] it can no longer be interpreted as a branching probability. Nevertheless θ-exponential polynomials can be represented by the well-known phase diagrams, as shown in Fig. 3, in which ○ represents an exponential phase and □ represents a deterministic part.

The class of θ-exponential polynomials is obtained by translating the terms of exponential polynomials by an amount θ_i. The translation of the individual terms ensures that program execution times occuring in practice can be modelled using a small number of stages. In addition, the deterministic parts θ_i can represent minimum execution times of subtasks. The linear combination of the terms using real weights c_i guarantees that the class of functions is closed under the operations of analysis. The closure properties are shown in [14].

3.2. Truncated θ-exponential polynomials

A consequence of modeling execution times by distributions with infinite domains is the lack of an upper bound for the execution times. However, this

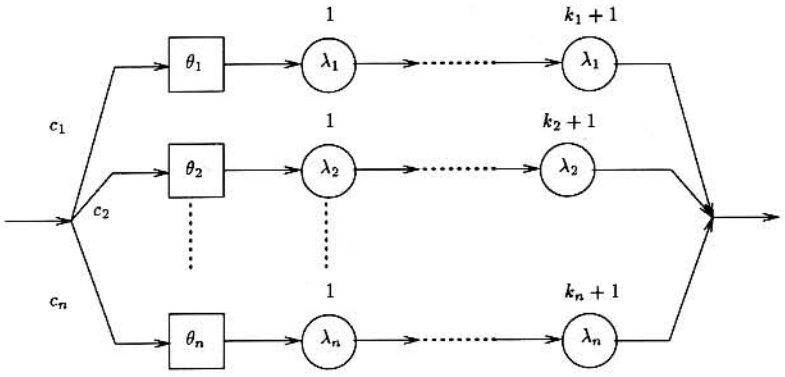


Fig. 3. Representation of θ-exponential polynomials by phase diagram.

contradicts the observed behavior of programs. It follows that, when modeling program execution times, it is more realistic to use distributions with finite domains than distributions with infinite domains. It is shown in Section 4.1 that the choice of the distribution class has a decisive influence on the distribution function of the execution time of massively parallel programs.

The θ-exponential polynomials discussed in Section 3.1 have the following useful properties; they are closed under the operations of analysis and every empirical distribution can be approximated to an arbitrary degree of accuracy by a θ-exponential polynomial. It can easily be shown that truncated θ-exponential polynomials also have these properties. Truncated θ-exponential polynomials (H_ω) are functions of the following form:

$$H_\omega(t) = \xi \sum_{i=1}^{n} H_i(t),\ H_i(t) = \begin{cases} 0 & \text{if } t < \theta_i \\ a_i (t-\theta_i)^{k_i} e^{-\lambda_i (t-\theta_i)} & \text{if } \theta_i \le t \le \omega \\ \dfrac{1}{n\xi} & \text{if } t > \omega \end{cases}$$

$$\text{with } \xi := \frac{1}{\sum_{i=1}^{n} H_i(\omega)},\ a_i \in R,\ \lambda_i \in R_0^+,\quad k_i \in N_0,\quad \theta_i \in R \quad \text{and} \quad \forall i : \omega > \theta_i.$$

4. Investigation of massively parallel systems

A typical task-structure in the field of parallel processing is the iterative execution of independent subtasks. A new step of the iteration cannot be begun

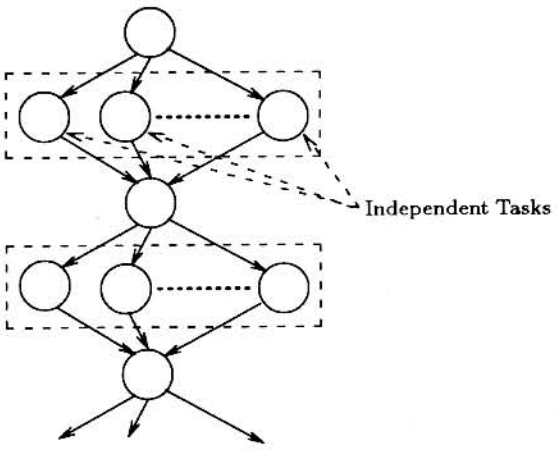


Fig. 4. Fundamental structure of parallel computations.

until *all* tasks of the previous step have been completed, as shown in Fig. 4 (barrier synchronization). Such structures occur in the solution of linear equations of partial differential equations, in search and optimization problems, in simulation, etc. [15].

The core of this fundamental structure, the parallel processing of n independent subtasks, is investigated in this section. In Fig. 4, each row of the graph represents one such composed task. There are n subtasks to be executed and there are no dependencies between them. The distribution function of the execution time is the same for all subtasks and processors. In Section 4.1 we study the case in which the number of processors is not restricted while in Section 4.2 different scheduling strategies are investigated.

4.1. Unlimited number of processors

In this section we investigate the execution of independent subtasks of parallel programs under the assumption that there are sufficient processors for all subtasks to be executed in parallel. From a statistical point of view the problem lies in determining the distribution function of the random variable $X_{\max} := \max\{X_i, \ldots, X_n\}$ where X_i are identically distributed random variables with distribution function $F(t)$. Let

$$F_{\max}(t) = (F(t))^n$$

be the distribution function of $X_{\max}$.

It is relatively easy to compute exactly the distribution function $F_{\max}(t)$ for small values of n. As the number of subtasks increases, the amount of work involved in calculating the overall distribution function and the moments greatly increases when modeling execution-times of subtasks with phase-type distributions. However, using results from extreme value theory (see e.g. [16]) it is possible to derive easily used approximation formulas for $F_{\max}(t)$ and for the mean value.

It is shown in Sections 4.1.1 and 4.1.2, by comparing the limit distributions of θ-exponential polynomials and truncated θ-exponential polynomials, that the choice of distribution class for the execution time of subtasks has a decisive influence on the distribution of the overall execution time of massively parallel structures. This is the case even when the initial distributions barely differ in the shape of their densities.

4.1.1. θ-exponential polynomials

The limit distribution of θ-exponential polynomials is the Gumbel distribution (see [14]). Using this fact, and results from [16] and [17], the following approximation formulas can be derived for mean μ_n and variance σ_n^2 of the maximum of n random variables, if their distribution functions are θ-exponential polynomials:

$$\mu_n = \beta_n + \alpha_n \gamma, \quad \sigma_n^2 = \alpha_n^2 \delta \tag{1}$$

Here γ is the Euler constant $\gamma = \lim_{n\to\infty}\gamma_n \approx 0.577216$, $\gamma_n = \sum_{i=1}^{n}(1/i) - \ln n$ and $\delta = \pi^2/6$. The normalizing constants α_n and β_n are given by [2]:

$$\beta_n = \left(\frac{1}{1-F}\right)^{-1}(n) \tag{2}$$

$$\alpha_n = \left(\frac{1-F}{F'}\right)(\beta_n) \tag{3}$$

The values α_n and β_n cannot be exactly determined analytically for a general θ-exponential polynomial. Rearranging Eq. (2) as:

$$(1-F)(\beta_n) - \frac{1}{n} = 0$$

leads to the problem of determining the zeroes of θ-exponential polynomials. Well-known numerical methods can be used to determine the values of β_n. In [14] an approximate formula for β_n for general θ-exponential polynomials is given. α_n can then easily be calculated when (3) is used. Here we give some simpler approximate equations for special cases of θ-exponential polynomials:

Exponential distribution:

$$\alpha_n = \frac{1}{\lambda} \qquad \beta_n = \frac{\ln n}{\lambda} \tag{4}$$

Erlang-k distribution:

$$\alpha_n = \frac{1}{\lambda} \qquad \beta_n = \frac{1}{\lambda}(\ln n + k \ln\ln n) \tag{5}$$

θ-exponential polynomials with $\forall(i = 1,\ldots,n): \theta_i = \theta$:

$$\alpha_n = \frac{1}{\lambda^{(\min)}} \quad \beta_n = \frac{1}{\lambda^{(\min)}}(\ln n + k^{(\max)} \ln\ln n) + \theta \tag{6}$$

The estimation functions 4–6 have been tested for particular distributions to get an idea of the convergence behavior of the exact results and the asymptotic approximations. Simulation results are also presented for comparison.

In Fig. 5, μ_n is an upper boundary for the expectation of the maximum of n independent random variables with identical distributions. It can be seen that this formula provides a very good estimate of the mean value for small n ($n < 10$). This boundary, which is independent of the distribution, is presented, e.g. in [18].

Fig. 5 shows that the estimated values for the expectation agree quite closely with the values obtained by simulation when the numerical values of the parameters α_n and β_n have been obtained by calculating the zeroes of the exponential

[2] $[1/(1-F)]^{-1}$ denotes the inverse function and not the reciprocal of $1/(1-F)$ here.

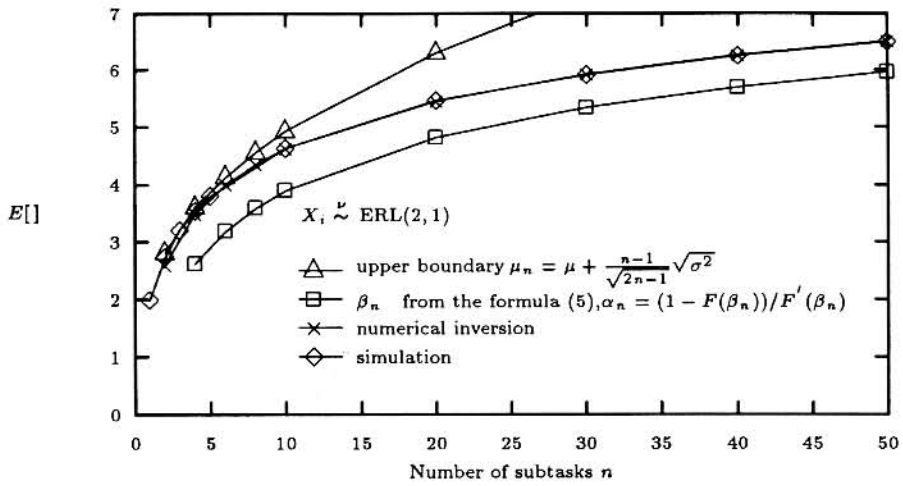


Fig. 5. Mean value of the total execution time for a parallel structure.

polynomial. If the distribution is Erlang, the parameters α_n and β_n can be estimated successfully using formula (5).

4.1.2. Truncated θ-exponential polynomials

We discuss below a phenomenon which must be taken into account when modeling execution times of massively parallel computations ($n > 100$). In this case the use of distributions with finite domains is more realistic than the use of distributions with infinite domains when modeling program execution times. The truncated θ-exponential polynomials were presented in Section 3 as an example of a class of distributions with finite domains. The limit distribution of the truncated θ-exponential polynomials is the Weibull distribution (see [14]). It is important that the asymptotic mean and variance of distributions, which have a Weibull distribution as their limit distribution, are given by:

$$\mu_\infty = x_0$$

$$\sigma_\infty^2 = 0,$$

where x_0 is the right-hand end-point $x_0 = \sup\{y : F(y) < 1\}$ of a distribution with distribution function F. Interesting consequences for the modeling and analysis of parallel programs follow from the fact that different extreme value distributions occur, depending on the domain of the initial distribution. We use the class of truncated θ-exponential polynomials to demonstrate the consequences more clearly.

Figs. 6, 7 and 8 show the nature of the convergence of the maxima of independent random variables with identical distributions. Fig. 6 shows the densities of the maximum of random variables with Erlang density function $f_{ERL(2,1)}$. Truncating $f_{ERL(2,1)}$ at ω leads to the density function $f_\omega(t) = I_{[0,\omega]}(t)(1/F_{ERL(2,1)}(\omega))f_{ERL(2,1)}(t)$, where $F_{ERL(2,1)}$ is the distribution function corresponding to $f_{ERL(2,1)}$. $I_{[0,\omega]}(t)$ is the indicator function with $I_{[0,\omega]}(t) = 1$ if $t \in [0, \omega]$ and

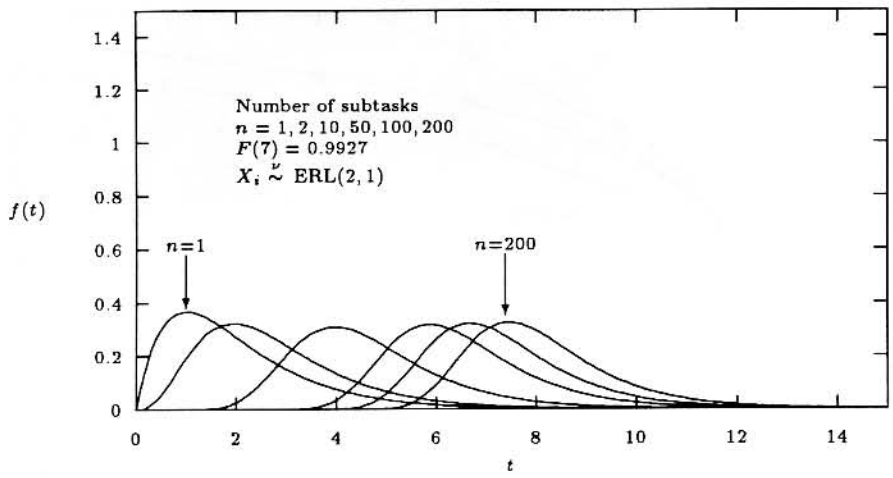


Fig. 6. Parallel composition of distributions with infinite domain.

$I_{[0,\omega]} = 0$ else. Fig. 7 shows the densities of the maximum of random variables with density f_ω with $\omega = 7$. It is particularly interesting that $F_{ERL(2,1)}(7) = 0.9927$; i.e. in the graphical representations of the density functions $f(t)$ and $f_\omega(t)$ the two curves are very close. Fig. 8 shows that the resulting density functions $f_{X,erl}$ and $f_{X,\omega}$ begin to diverge even for small values of n. This leads to the result, important in modeling, that, although they are nearly equal at the beginning, the distribution functions of the total execution time approach different extreme value distributions, depending on the distribution class. In general it is true that the use of distributions with infinite domains is acceptable when the subtasks are executed

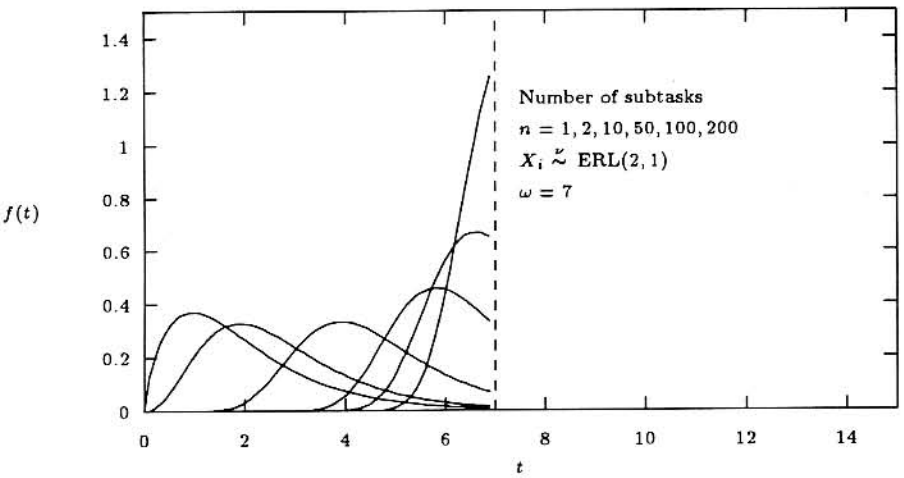


Fig. 7. Parallel composition of distributions with finite domain.

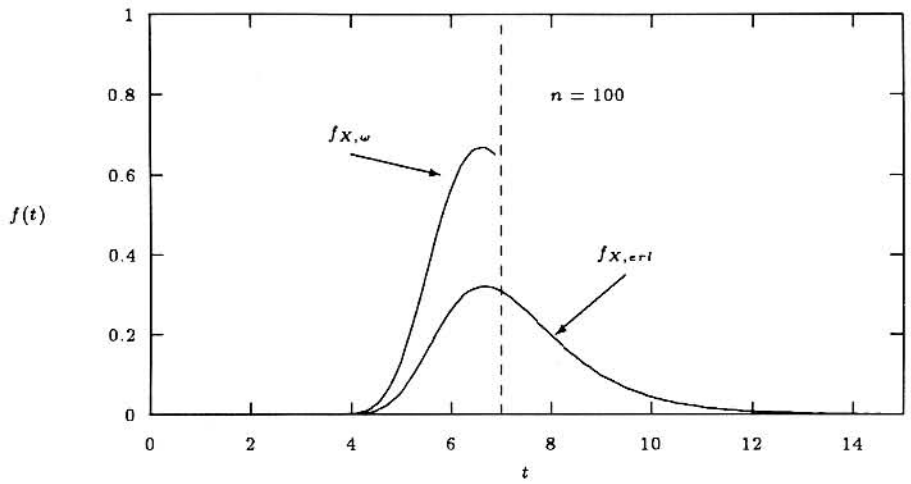


Fig. 8. $f_{X,erl}$ $(x_0 = \infty)$ versus $F_{X,\omega}$ $(x_0 = 7)$.

serially, but the assignment of the right-hand end-point $x_0 = \sup\{y : F(y) < 1\}$ has a decisive effect on the result in massively parallel structures ($n \gg 100$) and is thus of crucial importance in the modeling.

4.2. Limited number of processors

In this section we focus on the case in which the number of processors is less than the number of subtasks which can be computed in parallel. Thus the execution time is not only determined by the task structure but also by the scheduling strategy. We compare static and dynamic scheduling with or without respect to communication overhead. Proofs for the formulas given in this section can be found in [14].

4.2.1. Static scheduling without communication overhead

We assume that the execution times of all tasks are independent, identically distributed random variables. Thus the tasks are distributed equally among all processors and therefore we can use the task structure in Fig. 9 as a model. Each column in (b) corresponds to the execution of all the tasks on one processor.

When the value n/p increases the distribution of the execution time for one column tends to a normal distribution according to the Central Limit Theorem. The total execution time of the task structure tends to a Gumbel distribution (see [16] and [14]) and from that we can derive the following approximation formulas for the mean execution time $\mu_{n,p}$ and the variance $\sigma^2_{n,p}$:

$$\mu_{n,p} = \frac{n}{p}\mu + \sqrt{\frac{n}{p}\sigma^2}\left(\sqrt{2\ln p} - \frac{1}{2}\left(\frac{\ln\ln p + \ln 4\pi}{\sqrt{2\ln p}}\right) + \frac{1}{\sqrt{2\ln p}}\gamma\right)$$

$$\sigma^2_{n,p} = \frac{\pi^2}{12}\frac{n}{p\ln p}\sigma^2, \tag{7}$$

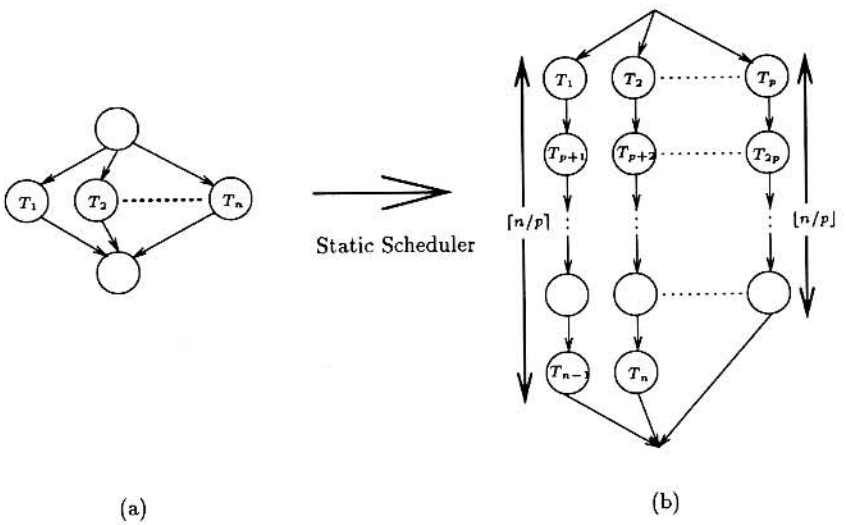


Fig. 9. Model for static scheduling.

where μ is the mean and σ^2 is the variance of the execution time of one subtask. The approximation for the mean value is also derived in [19]. (7) cannot be used when the number of processors p is large, because then the number of columns in Fig. 9 is also large. As shown in Section 4.1 we then have to take into account that the task execution-times are limited.

4.2.2. Dynamic scheduling without communication overhead

In this section we assume that the tasks are assigned to the processors according to a dynamic scheduling strategy during and not before program execution. When a processor becomes available it is assigned the next task which has not already been processed. Assuming that the task execution-times are all distributed exponentially we can easily derive the following formula for the total execution time:

$$X_{dyn} = \sum_{i=1}^{n-p} \min\left\{X_1^{(i)}, X_2^{(i)}, \ldots, X_p^{(i)}\right\} + \max\{X_1, X_2, \ldots, X_p\}$$

and

$$\mu_{n,p} = \frac{1}{\lambda}\left(\frac{n-p}{p} + \sum_{i=1}^{p}\frac{1}{i}\right) \qquad \sigma_{n,p}^2 = \frac{1}{\lambda^2}\left(\frac{n-p}{p^2} + \sum_{i=1}^{p}\frac{1}{i^2}\right)$$

This exact result only holds for exponentially distributed run-times of the subtasks. If the task execution times are not distributed exponentially we can give the following bounds for the total execution time:

$$X_{lower} = \frac{1}{p}\sum X_i \le X_{dyn} \quad X_{upper} = \frac{1}{p}\sum X_i + \max\left\{X_1^{(i)}, X_2^{(i)}, \ldots, X_{p-1}^{(i)}\right\} \ge X_{dyn}$$

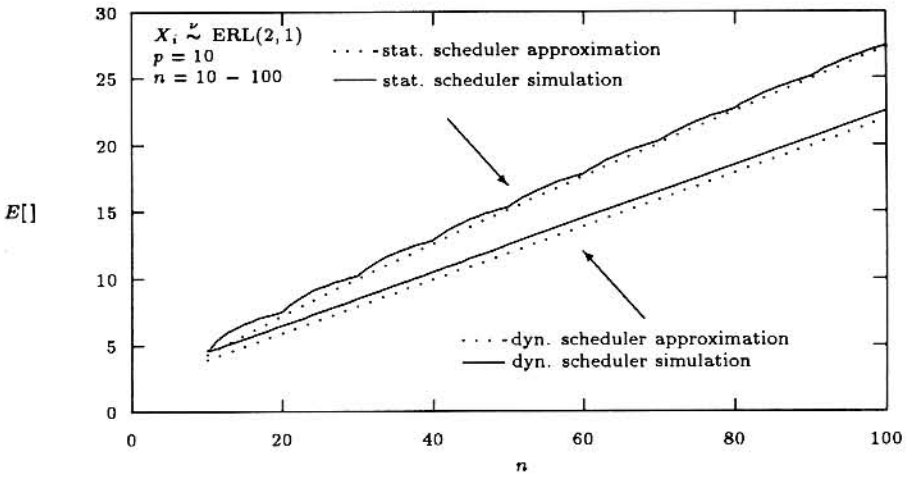


Fig. 10. Mean value: Static and dynamic scheduling.

Taking expectations we get:

$$E[X_{lower}] = \frac{n\mu}{p} \le E[X_{dyn}] \le \frac{n\mu}{p} + E\left[\max\left\{X_1^{(i)}, X_2^{(i)}, \ldots, X_{p-1}^{(i)}\right\}\right]$$
$$= E[X_{upper}]$$

The variances of X_{lower} and X_{upper} are calculated as follows:

$$\sigma^2_{lower} = \frac{n}{p^2}\sigma^2 \qquad \sigma^2_{upper} = \frac{n}{p^2} + Var\left[\max\left\{X_1^{(i)}, X_2^{(i)}, \ldots, X_{p-1}^{(i)}\right\}\right]$$

X_{ideal} in defined as that point of time at which a single processor with the same performance as all p processors together would complete the task. If we assume that at this point of time half of the processors have completed their work and the remaining processors have completed half of their last subtask we get the approximation:

$$\mu_{dyn} = \frac{n\mu}{p} + E\left[\max\left\{\frac{X_1^{(i)}}{2}, \ldots, \frac{X_{\lfloor (p-1)/2 \rfloor}}{2}\right\}\right]$$

Fig. 10 shows the mean execution times for different numbers of subtasks. There are 10 processors and the execution times of the subtasks are Erlang-2 distributed with parameter $\lambda = 1$. This figure indicates the following assumptions:

- The mean value of the execution time grows linearly with the number of subtasks by a factor $1/p$. This is valid for static scheduling as well as dynamic scheduling.
- The variances grow by a factor $1/p^2$ in the case of dynamic scheduling and with factor $1/(p \ln p)$ in the case of static scheduling.

Fig. 10 also indicates that a dynamic scheduler causes shorter execution times compared to a static scheduler when communication times are disregarded.

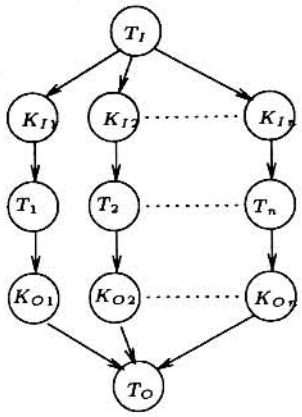


Fig. 11. Graph model representation of asynchronous communication.

4.3. Taking into account communication times

In this section we want to take into account the overhead involved with communication. We focus on the case of asynchronous communication. This means that the processors can perform other tasks during the communication between two subtasks. Fig. 10 indicates that a dynamic scheduler causes shorter execution times than a static scheduler if we ignore communication overhead. When using dynamic scheduling we do not know on which processor a task is going to be executed and therefore cannot start the communication before the processor is assigned. This means that the processor has to wait for the communication in the case of dynamic scheduling. In Fig. 11 the communication times are represented by the communication tasks $K_{I1},\ldots,K_{In}$ and $K_{O1},\ldots,K_{On}$. In the case of a static scheduler we can insert new edges to represent the scheduling strategy (Fig. 11). This is not possible for dynamic scheduling.

We assume that $p \ll n$. In the case of a dynamic scheduler we can add the communication times to the execution times of the subtasks and thus get the following approximation formula:

$$X_{dyn} = \frac{1}{p}\sum_{i=1}^{n}(K_{Ii} + X_i + K_{Oi})$$

In the case of a static scheduler the communication for subtask j will usually be completed before subtask i is finished. Therefore we get the following approximation formula:

$$X_{stat} = \max\{K_{I1} + X_1 + \ldots + X_{n/p} + K_{O1},\ldots,$$
$$K_{Ip} + X_{n-p+1} + \ldots + X_n + X_{Op}\}$$

where X_i is the run time of task T_i, K_{Ii} is the communication time between task T_I and task T_i and K_{Oi} is the communication time between tasks T_i and T_O. Fig. 12

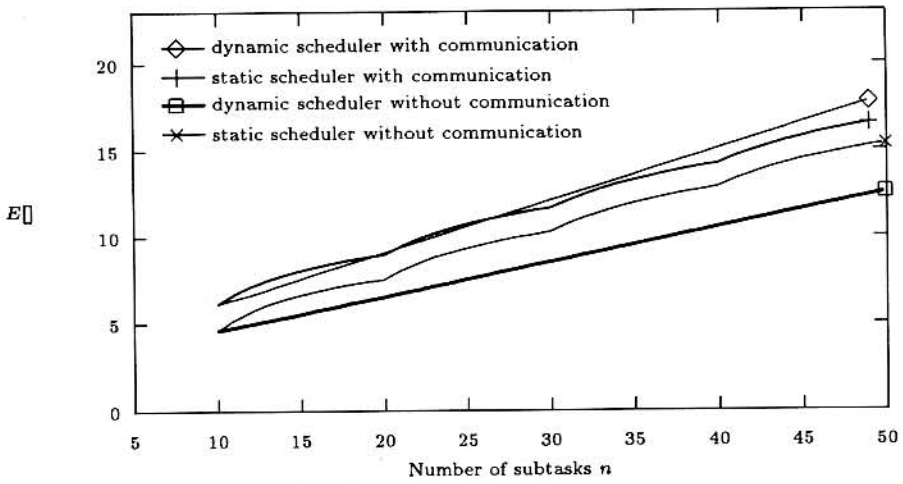


Fig. 12. Static versus dynamic scheduling with respect to communication.

shows that, for a growing number of subtasks, the static scheduler obtains shorter run-times compared to the dynamic scheduler.

5. Regular task structures

In this section we investigate iterative algorithms, which consist of multiple parallel-computation steps. The subtasks of one such step, or stage, can be performed independent of each other. The precedence relations between two successive steps have the same structure and the task graph models of this type of algorithms are regular in some sense.

The numerical solution of a system of linear equations using a relaxation

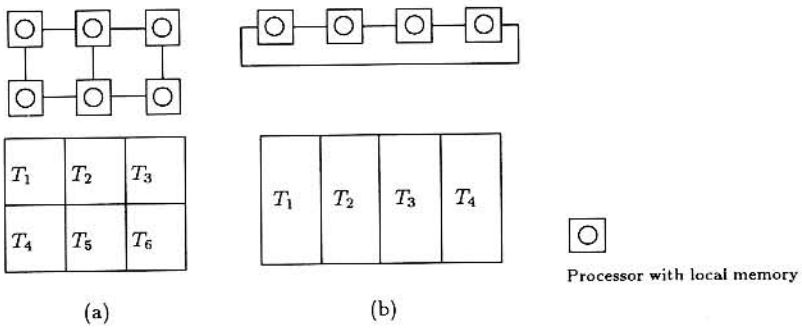
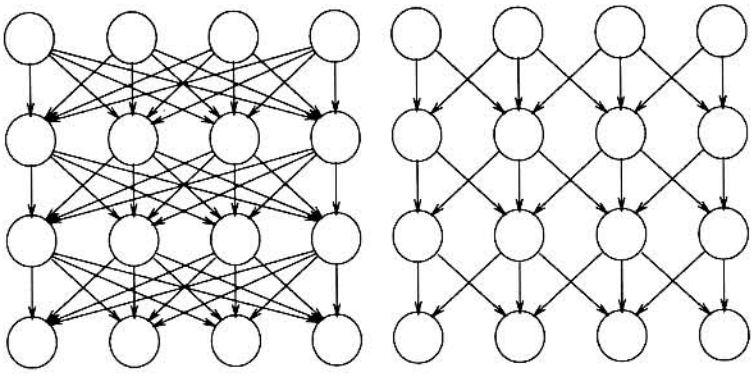
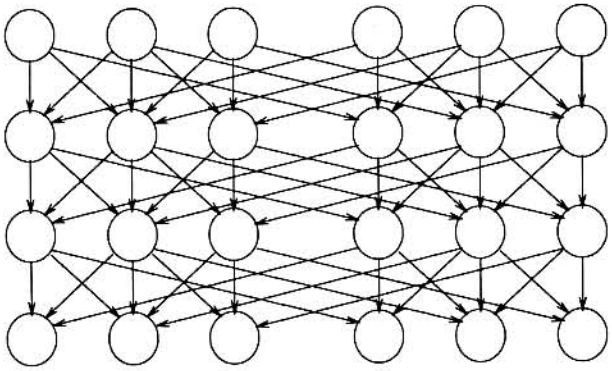


Fig. 13. Data partitioning for numerical solution of linear equations.



Ring with gobal synchronisation Ring with local synchronisation



Field with local synchronisation

Fig. 14. Task structures for numerical solution of linear equations.

method is typically representative of this class of applications. If we assume that the variables are arranged in a two-dimensional field there are several possibilities of placing the data on the available processors. In a ring architecture it seems to be adequate to use stripe partitioning as shown in Fig. 13 (b). In a mesh-connected architecture the use of the field partitioning shown in Fig. 13 (a) is reasonable.

We distinguish between *global* and *local* synchronization. Global synchronization means that all processors have to wait for the completion of *all* tasks on stage i before they can start execution on stage $i+1$. In contrast, the execution of a task on stage $i+1$ in *local* synchronization can be started when *just* the neighbour tasks and the task in the same column on stage i are finished. Fig. 14 shown the task graphs for global synchronization and for local synchronization on a ring architecture and a mesh connected processor field. All task graphs in Fig. 14 are regular in the sense of the definition given below:

A directed acyclic graph $G = (T, E)$ with a set of nodes $T = \{T_{ij} \mid i \in \{1, \ldots, n\},$

$j \in \{1, \ldots, m\}\}$ and a set of edges $E \subseteq T \times T$ is called *regular* if and only if there is a set W with:

$$W \subseteq \{1, \ldots, m\} \times (1, \ldots, m)$$

$$E = \left\{ \left(T_{ij}, T_{i+1,\hat{j}}\right) | \forall i \in \{1, \ldots, n-1\} : \left(j, \hat{j}\right) \in W \right\}$$

$$\forall T_{ij} \in T, i \neq 1 : pred\left(T_{ij}\right) \neq \emptyset$$

$$\forall T_{ij} \in T, i \neq n : succ\left(T_{ij}\right) \neq \emptyset$$

j is the number of the column and i is the index of the stage.

This regularity of task graphs causes regularity in the total execution time, a fact which is very important for the analysis of such regular structures. The following assumption could not be proven in general but numerous examples which were evaluated using simulation or Markov analysis showed this *regular* behavior. For the special case of the well-known N-Graph, which is the simplest non series-parallel graph, a proof is given in [14].

If we add a stage to a regular graph the mean value and the variance of the total execution time grow by a fixed amount $\Delta\mu$ and $\Delta\sigma^2$, respectively. We call this *stationary behavior*.

The above assumption facilitates the evaluation of large regular task graphs. Regular task graphs must only be evaluated up to a stage in which the stationary behavior is observed and, for bigger structures, the mean and the variance can be calculated by adding $\Delta\mu$ and $\Delta\sigma^2$. Stationary behavior can be assumed when the difference of the increases of the mean value and the variance becomes smaller than a limit ϵ.

6. Conclusion

Our paper has investigated performance analysis of parallel programs using precedence graphs. Starting from the practical experience that execution times of subtasks are typically characterized by small coefficients of variation, we introduced a generalization of exponential polynomials that allows the modelling of this type of distributions with a small number of stages k_i. We extended this class of distributions to truncated θ-exponential polynomials in order to take into account the fact that run-times of programs are limited. Using the limit distributions of θ-exponential polynomials and truncated θ-exponential polynomials we showed that the right-hand end point of distributions $x_0 = \sup\{y : F(y) < 1\}$ must be carefully chosen in modeling massively parallel structures.

We also investigated selected structures to derive approximation formulas which can be applied to large models. We presented approximate formulas for the parallel execution of independent subtasks in Section 4. In Section 4.1 formulas were derived for models in which the number of processors is sufficient to execute

all independent subtasks in parallel while in section 4.2 formulas for static and dynamic scheduling were given.

We also provided a method for the estimation of the total computation time of multi-stage structures in Section 5. This method is based on the observation that the regularity of the corresponding task-graphs causes a *stationary* behavior in the mean value of the total execution times. When a new stage is added to the structure – after a certain limit of stages has been reached – the mean value and the variance of the total execution time increases by a fixed value. This can be used to evaluate large structures by analyzing graphs which consist of a small number of stages and then extrapolating the results for large graphs.

References

[1] J.J. Martin, Distribution of the time through a directed acyclic network, *Operat. Res.*, 13 (1) (1965) 46–66.

[2] W. Kleinöder, Stochastische Bewertung von Aufgabenstrukturen für hierarchische Mehrrechnersysteme, Ph.D thesis, Universität Erlangen-Nürnberg, 1982.

[3] H.J. Fromm, Multiprozessor-Rechenanlagen: Programmstrukturen, Machinenstrukturen und Zuordnungsprobleme, 1982.

[4] R.A. Sahner and K.S. Trivedi, SPADE: A tool for performance and reliability evaluation, in: N. Abu El Ata, ed. *Modelling Techniques and Tools for Performance Analysis*, (1986) 147–163.

[5] G. Fleischmann and G. Werner, Modellierung und Bewertung paralleler Programme mit Hilfe von Markovketten, in: *Operations Research Proceedings* (Springer Verlag Berlin, 1989).

[6] G. Fayolle and M. Brun, The distribution of the transaction processing time in a simple fork-join system, in: *2nd Int. Workshop on Applied Mathematics and Performance / Reliability Modeld of Computer / Communication Systems*, Rome (1987) 203–212.

[7] A. Duda and T. Czachorski, Performance evaluation of fork and join synchronization primitives, *Acta Informat.* 24 (1987) 525–553.

[8] M.A. Marsan, G. Conte and G. Balbo, A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor Systems, *ACM Trans. Comput. Syst.* 2(2) (1989) 93–122.

[9] J. Mohan, Performance of parallel programs: Model and analyses, PhD thesis, Carnegie Mellon University, 1984.

[10] R. Sahner, A hybrid, combinatorial Markov method of solving performance and reliability model, PhD thesis, Dep. Comput. Sci., Duke Univ., Durham, 1986.

[11] R.A. Sahner and K.S. Trivedi, Performance and reliability analysis using directed acyclic graphs, *IEEE Trans. Software Eng.* SE-13(10) (Oct. 1987) 1105–1114.

[12] G. Bolch, G. Fleischmann, M. Gente, and B. Schiller, GIANT: A tool for performance evaluation of parallel programs, *Q-PASSPORT, Newsletter of the Dutch User Group Queueing / Performance AnalysiS Software (Q-PASS)* (1990).

[13] L. Schmickler, Approximation von empirischen Verteilungsfunktionen mit Erlangmischverteilungen und Coxverteilungen, in: U. Herzog and M. Paterok, ed., *Informatik Fachberichte 154: Messung, Modellierung und Bewertung von Rechensystemen, GI / NTG-Fachtagung* (Springer Verlag, Berlin, 1987) 118–133.

[14] G. Fleischmann, Leistungsbewertung paralleler Programme für MIMD-Architekturen: Modellbildung und mathematische Analyse, PhD thesis, Universität Erlangen-Nürnberg, 1990.

[15] U. Herzog, W. Hoffmann and W. Kleinöder, Performance modeling for hierarchically organized multiprocessor computer systems, in: *Proc. 1979 Int. Conf. on Parallel Processing* University of South Florida, 1979.

[16] S.I. Resnick, Extreme values, regular variation, and point processes, vol. 4 of *Applied Probability* (Springer Verlag, New York, 1987).

[17] A.M. Mood, F.A. Graybill and D.C. Boes, *Introduction to the Theory of Statistics* (McGraw-Hill, 1974).
[18] H.A. David, *Order Statistics* (John Wiley, New York, 1981).
[19] S. Madala and J.B. Sinclair, Performance of synchronous parallel algorithms with regular structures, *IEEE Trans. Parallel Distributed Syst.* 2(1) (1991) 105–116.

Parallel Computing

Publication information
PARALLEL COMPUTING (ISSN 0167-8191). For 1994 volume 20 is scheduled for publication.
Subscription prices are available upon request from the publisher. Subscriptions are accepted on a prepaid basis only and are entered on a calendar year basis. Issues are sent by surface mail except to the following countries where air delivery via SAL is ensured: Argentina, Australia, Brazil, Canada, Hong Kong, India, Israel, Japan, Malaysia, Mexico, New Zealand, Pakistan, PR China, Singapore, South Africa, South Korea, Taiwan, Thailand, USA. For all other countries airmail rates are available upon request.
Claims for missing issues must be made within six months of our publication (mailing) date.
Please address all your requests regarding orders and subscription queries to: Elsevier Science, Journal Department, P.O. Box 211, 1000 AE Amsterdam, The Netherlands. Tel.: 31-20-5803642, fax: 31-20-5803598.
US mailing notice – Parallel Computing (0167-8191) is published monthly by Elsevier Science (Molenwerf 1, Postbus 211, 1000 AE Amsterdam). Annual subscription price in the USA: US$769.00 (US$ price valid in North, Central and South America only), including air speed delivery. Second class postage paid at Jamaica, N.Y. 11431.
USA POSTMASTERS: Send address changes to Parallel Computing, Publication Expediting, Inc., 200 Meacham Avenue, Elmont, NY 11003. Air freight and mailing in the USA by Publication Expediting.

Instructions to authors
Contributions should be written in English and include a 50 to 100 word abstract. They will be published as full papers, preferably of not more than ten pages, or as short communications of not more than two pages, and should be sent in triplicate to the appropriate Regional Editor. The author's mailing address should appear on the manuscript. No page charge is made. Please make sure that the paper is submitted in its final form. Corrections in the proof stage, other than printer's errors, should be avoided; costs arising from such extra corrections may be charged to the authors. Upon acceptance of an article, the author(s) will be asked to transfer copyright of the article to the publisher. This transfer will ensure the widest possible dissemination of information. Manuscripts should be prepared for publication in accordance with instructions given in the "Guide for Authors" (available from the Publisher), details of which are condensed below:

1. The manuscript must be typed on one side of the paper in double spacing (including footnotes, references, and abstracts) with wide margins. A duplicate copy should be retained by the author.
2. All mathematical symbols which are not typewritten should be listed separately.
3. Footnotes should be kept to a minimum and as brief as possible, they must be numbered consecutively and typed on a separate sheet in the same format as the main text.
4. Special care should be given to the preparation of the drawings for the figures and diagrams. Except for a reduction in size, they will appear in the final printing in exactly the same form as submitted by the author; normally they will not be redrawn by the printer. In order to make a photographic reproduction possible, all drawings should be on separate sheets, with wide margins, drawn large size, in India ink, and carefully lettered. Exceptions are diagrams only containing formulae and a small number of straight lines (or arrows); these can be typeset.
5. References should be listed alphabetically, in the same way as the following examples:
   For a book:
   [1] S. Ginsburg, *Algebraic and Automata-Theoretic Properties of Formal Languages* (North-Holland, Amsterdam, 1975).
   For a paper in a journal:
   [2] J. Rhodes and B.R. Tilson, Lower-bounds for complexity of finite semigroups, *J. Pure Appl. Algebra* 1 (1971) 79–95.
   For a paper in a contributed volume:
   [3] A.K. Joshi, L.S. Levy and M. Takahashi, A tree generating system, in: M. Nivat, ed., *Automata, Languages and Programming* (North-Holland, Amsterdam, 1973) 454–455.
   For an unpublished paper:
   [4] J. Goldstine, Abstract families of languages generated by bounded languages, Ph.D. Thesis, Univ. California at Berkeley, 1970.

Authors: Please mention your email address and / or fax number on the first page of your manuscript

---

Authors' benefits
1. 50 reprints of each contribution free of charge.
2. 30% discount on all Elsevier Science books.
   (Order forms will be sent together with the proofs.)

Computer Benchmarks

edited by **J.J. Dongarra** *and* **W. Gentzsch**

Advances in Parallel Computing Volume 8

The performance of a computer is a complicated issue and a function of many interrelated quantities. These quantities include: the application, the language, the implementation, the compiler, the architecture, and the hardware characteristics. The usual method to evaluate the performance is to compose a benchmark of programs. This book presents a useful overview on benchmarking. Over twenty experts contributed papers on five important topics concerning benchmarking advanced scientific computer systems: taxonomy and performance metrics, well-known standard and application benchmarks, compiler benchmarks for parallel computers, and benchmarks for database systems.

Contents: Preface to the series. Preface. Introduction. **I. Performance Prediction.** Toward a taxonomy of performance metrics (J. Worlton). Toward a better parallel performance metric (X.-H. Sun, J.L. Gustafson). Performance parameters and benchmarking of supercomputers (R.W. Hockney). A framework for benchmark performance analysis.(R.W. Hockney). Performance estimates for supercomputers. The responsibilities of the manufacturer and of the user (W. Schönauer, H. Häfner). A philosophical perspective on performance measurement (D.F. Snelling). **II. Performance Measurement.** A detailed look at some popular benchmarks (R.P. Weicker). Scientific benchmark characterizations (M. Berry, G. Cybenko, J. Larson). The SPEC benchmarks (K.M. Dixit). The benchmark of the EuroBen group (A.J. van der Steen). Performance of various computers using standard sparse linear equation solving techniques (J.J. Dongarra, H.A. van der Vorst). **III. Compiler Benchmarks.** A comparative study of automatic vectorizing compilers (D. Levine, D. Callahan, J. Dongarra). Parallel loops - A test suite for parallizing compilers: Description and example results (J. Dongarra *et al.*). **IV. Benchmarks for Parallel Computers.** NAS parallel benchmark results (D.H. Bailey *et al.*). Parallel performance of applications on supercomputers (C.M. Grassl). The Genesis distributed memory benchmarks (C.A. Addison *et al.*). Performance of the Intel iPSC/860 and Ncube 6400 hypercubes (T.H. Dunigan). CFD applications on transputer systems (M. Faden *et al.*). Benchmarking parallel programs in a multiprogramming environment: the PAR-Bench system (W.E. Nagel, M.A. Linn). **V. Benchmarks for Database Systems.** A hybrid benchmarking model for database machine performance studies (J.A. McCann, D.A. Bell). Database system benchmarking and performance testing (R. Longbottom).

1993 364 pages
Dfl. 220.00 (US 125.75)
ISBN 0-444-81518-X

Elsevier Science Publishers
P.O. Box 103
1000 AC Amsterdam
The Netherlands

P.O. Box 945
Madison Square Station
New York, NY 10160-0757

The Dutch Guilder (Dfl.) price is definitive. US $ prices are subject to exchange rate fluctuations. Customers in the European Community should add the appropriate VAT rate applicable in their country to the price.