In: *Proc. 8th Int'l Conf. on Model. Techn. and Tools for Comput. Perform. Evaluation*, Heidelberg, Sept. 20-22, 1995.

Compile-time Performance Prediction of Parallel Systems

Arjan J.C. van Gemund

Dept. of Electrical Engineering
Delft University of Technology
P.O.Box 5031, NL-2600 GA Delft, The Netherlands
e-mail: a.vgemund@et.tudelft.nl

Abstract. A compile-time technique is outlined that yields low-cost, analytic performance models, intended for crude scalability analysis and first-order system design. The approach extends current static techniques by accounting for any type of resource contention that may occur. In this paper we report on the accuracy of the prediction method in terms of theory, simulation experiments, as well as measurements on a distributed-memory machine. It is shown that for series-parallel computations with random resource access patterns, the average prediction error is limited well within 50 % regardless the system parameters, where traditional compile-time methods yield errors up to orders of magnitude.

1 Introduction

In the performance prediction of parallel systems many approaches exist that represent a specific trade-off between accuracy and cost. Although compile-time techniques entail a sacrifice in accuracy when compared to, e.g., simulation, this loss may be acceptable during the first phases of system design in view of the low cost and high level of parameterization that can be achieved. Where the cost issue plays a prominent role in the development towards automatic system optimization, the symbolic nature of the models enables an efficient parameter (e.g., scalability) study based on a one-only developed model.

The quality of a performance modeling approach for parallel systems is highly determined by the way in which task synchronization (condition synchronization) as well as resource contention (mutual exclusion) are accounted for. Whereas static techniques focus on task synchronization they do not account for resource contention except for *ad hoc* approaches. Queuing for resources alone, however, can already degrade performance by orders of magnitude. Clearly, a static approach that *sustains* a minimum accuracy across a large parameter (search) space would be of value.

Aimed to overcome the fundamental lack of prediction robustness of static methods, an extension has been proposed, that, at the same low cost, approximately accounts for resource contention. The analysis method is defined in terms of a simulation formalism called PAMELA (PerformAnce ModEling LAnguage [8]). Both (parameterized) program and machine are modeled in terms of

separate PAMELA submodels, which, when combined by substitution, results in a model L of the complete system. Rather than simulating (executing) L to obtain the execution time estimate T, L is compiled into a parameterized performance model [9] that computes an alternative, lower bound T^l at much lower cost.

Thus, the emphasis in our approach is the derivation of symbolic, low cost performance models while limiting the sacrifice in accuracy by integrating contention analysis within the static scheme. In this paper we study the accuracy of T^l for series-parallel (SP) graphs both through simulation experiments as well as experiments involving a distributed-memory machine. It is shown that for a large class of task systems the average error of T^l relative to T is limited within a factor of 2, regardless program or machine parameter settings.

The rest of the paper is organized as follows. For the sake of completeness, in Section 2 we briefly present the approach as well as a rationale in terms of related work. In Section 3 we study the nature of the average prediction error based on 1000+ simulation experiments. In Section 4 we report on the case study involving the distributed-memory machine. The paper is concluded in Section 5.

2 PAMELA

2.1 Formalism

For the sake of completeness we briefly describe the subset of the formalism that applies to the analysis of the SP models presented in the paper. A more elaborate presentation appears in e.g., [10].

Basically, PAMELA is an imperative formalism extended with a number of constructs to express concurrency and (virtual) time. Apart from the conditional control flow operators **if** and **while**, PAMELA includes binary (infix) operators to describe sequentialism (i.e., '**;**') and fork/join parallelism (i.e., '$\|$'). The parallel operator implicitly enforces barrier synchronization that allows for the construction of SP models. Work is described by the **use** construct, like in $\mathbf{use}(s, \tau)$, in which the invoking process (task) exclusively acquires service from server s for τ units time (excluding possible queuing delay). In the sequel we will often refer to servers as (active) *resources*. A resource s has a multiplicity, denoted $|s|$ that may be larger than 1. The service time τ may be deterministic or stochastic. Although stochastic (simulation) models can be specified, in the compile-time calculus described in Section 2.2 only deterministic (or mean) values will be considered. Like in queuing networks, it is convenient to define an infinite-server ρ such that $|\rho| = \infty$. Instead of $\mathbf{use}(\rho, \tau)$ we will simply write $\mathbf{delay}(\tau)$. Replication is described by the reductions **seq** and **par**, defined by

$$\mathbf{seq}\ (i = a, b)\ L_i = L_a\ ;\ \ldots\ ;\ L_b\ \ ,\quad \mathbf{par}\ (i = a, b)\ L_i = L_a \parallel \ldots \parallel L_b$$

Corresponding to the formal approach towards model construction and analysis, we write a PAMELA model according to the usual equation syntax, which implies a simple substitution semantics. Consider the following PAMELA model L, i.e., $L = \mathbf{delay}(1)\ ;\ x = 2\ ;\ \mathbf{delay}(x)$. The '**;**' operators in the (process-algebraic)

expression L specify a sequence of three operations, two of which directly affect virtual time. The resulting execution time is given by $T = 1 + 2 = 3$. The example illustrates that PAMELA models may include data operations which indirectly affect timing behavior. Although performance simulation models generally exclude the original data operations, a part must sometimes be preserved in order to account for data-dependent control flow.

Example 1 Consider a machine repair model (MRM) in which P clients either spend a mean time τ_l on local processing, or request service from a server s ($|s| = 1$), with service time τ_s (both according to some distribution), with a total cycle count of N iterations (unlike steady-state analysis, in our approach we require models to terminate). The PAMELA model is specified by

$$L = \mathbf{par}\ (p = 1, P)\ \mathbf{seq}\ (i = 1, N)\ \{\mathbf{delay}(\tau_l);\ \mathbf{use}(s, \tau_s)\}$$

in which the exclusive service is expressed by the **use** operation applied to resource s that represents the server. The example illustrates the *material-oriented* modeling approach [15] in which the server is modeled by a passive construct. In the machine-oriented approach, the server would be modeled by a separate *process* that synchronizes through message-passing. Despite the advantages of message-passing in model *construction*, our approach permits a straightforward model *analysis*. □

As the emphasis in this paper is on the *analysis*, the use of PAMELA in *modeling* shared and distributed-memory programs and (vector) machines is discussed elsewhere [8, 10]. Note, that PAMELA's operators essentially enable the same modeling accuracy when compared to (hybrid) task graph and/or queuing approaches, while hardware and software are modeled in terms of one formalism. Through the natural expression of data-dependent control flow even a larger degree of modeling flexibility is possible. Being an imperative formalism, a PAMELA model L can be directly executed (simulated). However, we will use this evaluation mode only in order to validate our analytic technique.

2.2 Analysis

Our analytic approach is based on a lower bound approximation of contention integrated within a critical path analysis of the task graph. In the following we briefly summarize the analysis for SP graphs. More details can be found in [9, 10].

Although, typical for compile-time approaches, in many cases for each input data set conditional control flow will be accounted for in terms of e.g., branch probabilities or statement frequencies, the transformation $\mathbf{if}\ (c)\ \mathbf{use}(r, \tau) \rightarrow \mathbf{use}(r, [c]\tau)$ shows in principle how conditional control flow is formally handled in the symbolic analysis. The $[\ldots]$ construct denotes Iverson's operator defined by $[false] = 0$ and $[true] = 1$. Since a PAMELA model is block-structured the above transformation can be applied recursively, eventually yielding a model without conditionals. Although this scheme enables preserving those parameter

dependencies that are of interest, usually the [...] terms are eventually reduced based on the auxiliary information mentioned above [10].

Task synchronization, as in conventional static approaches, is accounted for by critical path analysis in which we ignore the effects of contention (i.e., each **use** statement is interpreted as if it were a **delay** statement). Let $\varphi(L)$ denote the execution time given by critical path analysis. Where for general task graphs the computation graph of $\varphi(L)$ is isomorphic to L (i.e., a set of symbolic expressions), for SP graphs $\varphi(L)$ forms one expression that is amenable to SP reduction. In terms of the PAMELA operators ';' and '||', the following recursion holds

$$\varphi(L) = \begin{cases} \varphi(L_1) + \ldots + \varphi(L_N), & L = L_1 \ ; \ldots ; \ L_N ; \\ \varphi(L_1) \max \ \ldots \ \max \ \varphi(L_N), & L = L_1 \parallel \ldots \parallel L_N ; \\ \tau, & L = \mathbf{delay}(\tau) \text{ or } L = \mathbf{use}(r, \tau). \end{cases} \quad (1)$$

Resource contention is approximated by a simple *lower bound* analysis based on a computation of the total service demand as shown in the following. Let $\underline{\delta}(L) = (\delta_1, \ldots, \delta_M)$ denote the total service demand vector of L where M is the total number of resources involved and δ_m denotes the service demand on resource r_m. We will write $\delta_m(L)$ to denote the m-th element of $\underline{\delta}(L)$. Clearly,

$$\underline{\delta}(L) = \begin{cases} \underline{\delta}(L_1) + \ldots + \underline{\delta}(L_N), & L = L_1 \ ; \ldots ; \ L_N \text{ or } L = L_1 \parallel \ldots \parallel L_N ; \\ \tau \underline{e}^m, & L = \mathbf{use}(r_m, \tau). \end{cases} \quad (2)$$

where $\underline{e}^m = (0, \ldots, 0, 1, 0, \ldots, 0)$ is the M-dimensional unit vector in the m direction, and addition and multiplication are defined element-wise. Let ω denote the lower bound on the execution time of L due to the fact that each access to a resource is at least serialized. Then

$$\omega(L) = \max_{m=1 \ldots M} \frac{\delta_m(L)}{|r_m|} \quad (3)$$

Combining the lower bound due to contention (ω) with the result of critical path analysis (φ) it follows that the lower bound on T is predicted by

$$T^l(L) = \max(\varphi(L), \omega(L)) \quad (4)$$

Where Eq. 4 applies to basic parallel sections, for general (possibly non-SP) models the following (recursive) generalization provides a much sharper bound as will be illustrated in Example 3.

$$T^l(L) = \begin{cases} T^l(L_1) + \ldots + T^l(L_N), & L = L_1 \ ; \ldots ; \ L_N ; \\ T^l(L_1) \max \ \ldots \ \max \ T^l(L_N) \max \ \omega(L), & L = L_1 \parallel \ldots \parallel L_N ; \\ \max(\varphi(L), \omega(L)), & \text{otherwise.} \end{cases} \quad (5)$$

Note, that conventional compile-time analysis disregards ω while queuing analysis (partially) disregards φ. Due to the fact that, in addition to the critical path, we account for the serialization due to mutual exclusion, we have coined this lower bound approach serialization analysis. Like conventional analysis, for SP models serialization analysis has a linear solution complexity.

Example 2 Recall the MRM in Example 1. By Eq. 1 and Eq. 3 it follows

$$\varphi = \max_{p=1...P} \sum_{i=1}^{N} (\tau_l + \tau_s) = N(\tau_l + \tau_s) \ , \quad \omega = \sum_{p=1}^{P} \sum_{i=1}^{N} \tau_s = PN\tau_s$$

Hence, by Eq. 4 (or Eq. 5) it follows $T^l = N \max(P\tau_s, \tau_l + \tau_s)$. Unlike conventional compile-time analysis T^l accounts for the additional queuing delay when s is saturated. The above analysis yields the same result as asymptotic bound analysis in queuing theory. Let R denote the response time and let $Z = \tau_l$ denote the think time. Then the mean cycle time $R + Z$ equals φ/N for $P \ll P^*$ and ω/N for $P \gg P^*$, where the saturation point $P^* = (\tau_s + \tau_l)/\tau_s$ denotes the crossover between the asymptotes. □

Example 3 In order to demonstrate the vital importance of Eq. 5, consider the following model, i.e., $L =$ **seq** $(i = 1, N)$ **par** $(p = 1, P)$ **use**(r_i, τ), in which resource usage is non-uniformly distributed over the length of the entire computation. Where Eq. 4 yields $T^l = \max(P\tau, N\tau)$, Eq. 5 yields $T^l = \sum_{i=1}^{N} \max(P\tau, \tau) = NP\tau$. Thus applying applying Eq. 4 to *each* parallel section instead of only once improves the bound by as much as a factor N. □

2.3 Related Work

In order to provide a rationale for our approach, in this section we review some of the many interesting approaches to the performance modeling of parallel systems. For a more elaborate survey the reader is referred to [10].

As mentioned earlier, modeling accuracy is highly determined by the way in which task synchronization and resource contention are accounted for. With its well-established theory, (timed) Petri nets are frequently used either as an explicit modeling formalism [2] or as an intermediate representation [23]. Although inherently capable to accurately model both types of synchronization the exponential complexity of the associated state space analysis prohibits an approach where low cost is of key interest. This also applies to stochastic process algebras, despite their attractive language properties [11].

Many approaches focus on the analysis of task synchronization, using a task graph representation with stochastic task durations to account for the nondeterminism of conditional control flow and contention. Aimed to circumvent the exponential analysis complexity due to the use of stochastic parameters, many approaches focus on SP reduction [7], sometimes in combination with a restriction to exponential-type distributions [20]. Other techniques either approximate the graph structure in terms of an SP version [12] or approximate the task distribution by a combination of deterministic and exponential terms [22]. Despite the use of stochastic variables, the inherent inability of a task graph to model resource contention prohibits the use of task graphs for performance prediction of systems where (machine) resource parameter variations are of interest.

Alternative approaches are described that combine a stochastic graph with a queuing network which accounts for machine level contention. In order to

circumvent the exponential analysis complexity a number of approaches either focus on SP reduction [17] or apply path analysis based on the fact that the actual variance in task times is usually very limited [1, 14]. Although the path analysis approach is very efficient, due to the underlying queuing network the solution complexity is polynomial at best.

By tradition, compile-time approaches are based on the analysis of deterministic graphs in which case no state space analysis is required. As a result of the predominant data parallel structure of parallel programs, a simple, scalar SP reduction is applied which implies a solution cost that is only linear in the size of the program source [5, 6, 21]. As SP reduction is applied to the program rather than the associated graph, a symbolic analysis scheme is possible where program parameters (e.g., loop bounds) are retained within the resulting performance model [4, 19, 24]. Despite the fact that some of the techniques feature a (usually machine-specific) analysis of (processor, memory, and/or network) contention (e.g., [4, 24, 18]), the approach suffers from the same inability to naturally account for contention as mentioned above for stochastic graphs.

In our approach, contention is naturally included by using a concurrent language as representation formalism. Unlike most simulation languages, however, we exploit the concept of compile-time analysis yielding highly parametric performance models, rather than just compiling simulators. Extending the deterministic graph analysis mentioned earlier we do account for any form of contention in order to provide the robustness needed in view of the large parameter range typically covered by a largely symbolic performance model. The basic premise of our approach is the following. Due to the fact that task variance is relatively small [1], a deterministic approach (critical path analysis) will not entail large errors with respect to task synchronization effects. Assuming that control flow for a representative data set can be accounted for in terms of mean task durations, it follows that a parametric analysis method is possible if contention can be analytically approximated. In this paper we show that even for our simple analytical approach to contention analysis the average error is quite limited.

3 Average Accuracy

3.1 Introduction

In this section, we study the deviation of T^l relative to the mean value of T based on simulation experiments involving random task graphs. As illustrated by Example 2, for models in which the resource demand is reasonably uniform during the entire computation (i.e., in contrast to models such as in Example 3), T^l approaches the mean value of T either when $\varphi \gg \omega$ (critical path dominates) or when $\varphi \ll \omega$ (queuing dominates). Thus the average error is often quite acceptable (as experiments will show later on). As mentioned in Example 2 the choice of the lower bound as a practical estimate is also inspired by similarities between the execution of L and interactive queuing systems. Although, formally, the resemblance is extremely remote it is interesting to relate the lower bound

approach to the asymptotic bound analysis of an (operationally) comparable interactive queuing system[1]. If we define Z as the think time, D as the total service demand and D_{max} as the service demand at the bottleneck device, we can interpret φ as the horizontal cycle time asymptote $D + Z$ (Z accounts for task synchronization delay), while ω corresponds to the ND_{max} asymptote. The largest deviation occurs at the saturation point, where $D + Z = ND_{max}$.

Consequently, in order to present our experimental results we use an operational metric called "serialization index" that characterizes the degree of contention within a system. The metric is defined by

$$\theta = \log\left(\frac{\omega}{\varphi}\right) \tag{6}$$

As will be shown, θ characterizes a model as to the the likelyhood of T^l being an accurate prediction. For models with large $|\theta|$ the average accuracy of T^l is expected to be better than for models where $|\theta| \approx 0$.

3.2 Experiments

Experiments have been performed involving 1000+ random SP graphs in which the predictions T^l are compared to the simulation results T. The models are generated such that the θ values lie around the (worst case) region of interest ($|\theta| \approx 0$). Apart from the fact, that many computations of interest are SP structured[2], the choice for SP models is also motivated by the fact that it enables an evaluation of the improvement of Eq. 5 on the accuracy compared to Eq. 4. Each model comprises $N = 100$ tasks while the number of resources involved varies from $M = 2 \ldots 150$. The graphs are generated by a simple algorithm that iteratively adds a new task t_i to a random selected task t_j within the graphs generated up to then (j is determined in each iteration). The probability that t_i is placed in series or parallel with t_j is determined by an input parameter, denoted s. Each task t_i is characterized by a unique service demand vector $\underline{\delta}_i = (\delta_{i,1}, \ldots, \delta_{i,M})$ in which each element is i.i.d. uniformly over $[0, 1]$. Thus, balanced systems are generated (on average). Experiments have verified that this choice indeed provides the worst case with respect to the accuracy of T^l. Each resource m is accessed multiple times based on the existence of some deterministic service time τ. Thus each task executes $\delta_{i,m}/\tau$ accesses to resource m. The order in which the resources are visited is random. In order to minimize simulation time (many models are simulated), τ is chosen such that the mean of T does not deviate significantly from results for $\tau \to \infty$ (values in the order

[1] Note, that the comparison is purely intuitive as we disregard many details, e.g., the fact that each task should map to a unique job class; possible transient phases like startup and shutdown are ignored; the task graph should be cyclic in order to have steady state execution, etc.

[2] Note, that the application range of PAMELA SP models is essentially greater than just SP task graphs. For instance, pipelining can be expressed in terms of a parallel section of contending tasks [8].

of 1 % of the largest service demand $\delta_{i,m}$ ($\approx$ 100 visits) have been found to suffice). As N is fixed ($N = 100$), the parameters M and s determine the (mean) θ value of the generated models. As φ is proportional to M, large values of M will generate models with a negative θ. For low s, however, many parallel tasks are created on average which has a positive influence on θ.

Figure 1 shows the ratio T^l/T based on 1200 random models exhibiting θ values ranging from $-2 < \theta < 2$. Both the prediction ratios based on Eq. 4 (α) and Eq. 5 are shown (β). Each data point of both series of 120 points represents an average value based on 10 random draws in order to reduce noise. The results

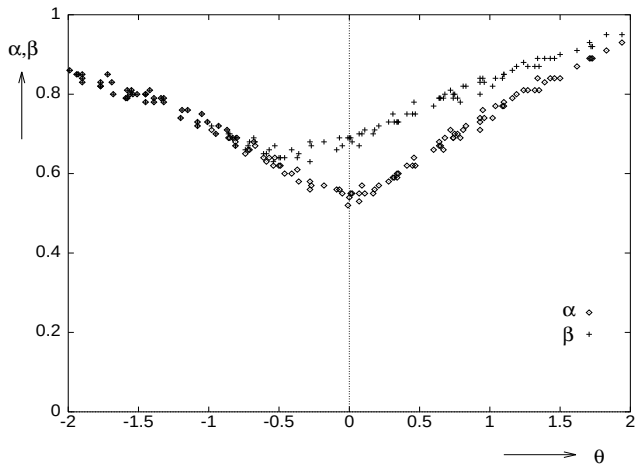

Fig. 1. T^l accuracy (α, β) based on 1200 random SP models ($N = 100$)

clearly show a high correlation between (α, β) and θ in which the deviation from unity is indeed maximal for models that exhibit $\theta = 0$. Thus, for random graphs the diagnostic value of the operational parameter θ appears to be quite significant, especially when considering the fact that two graphs with comparable θ values usually have quite a different structure. While the essential necessity of Eq. 5 has already been demonstrated (cf. Example 3), even for the models with uniform resource demand (in time) as produced by the random generator, its application (β) still yields an improvement for models with highly parallel subsections (e.g., $\theta > 0$). In the following we will only consider α. In the above experiments the models are generated for $s = 0.1$ with M varying from $M = 2$ ($\theta \approx 2$) to $M = 150$ ($\theta \approx -2$). Models with $\theta \approx 0$ are generated for $M = 20$. For each value of M the variance of θ is approximately 0.05 which accounts for the reasonably continuous plot. Although no extensive experiments have yet been performed for different values of N, measurements indicate that the α curves tend to be more 'v'-shaped for small N, corresponding to the fact that the scale of θ

is still somewhat dependent on the problem size. However, $N = 100$ also appears to be quite representative for larger models[3] as well. Additional measurements (described in [10]) indicate that the *minimum* value of α at $\theta = 0$, i.e., α^*, highly correlates with M. For instance, each of the following set of parameter tuples, i.e., $(N, M, s) \in \{(30, 8, 0.1), (100, 8, 0.3), (300, 8, 0.5)\}$ generates models with $|\theta| \approx 0$ that yield $\alpha^* \approx .6$ on average. The results show the existence of an asymptote for large M given by $\alpha^* \approx 0.5$. Again, it is tempting to compare this upper bound on the (mean) deviation with the result from asymptotic bounding analysis of interactive queuing systems. For instance, consider the MVA recursion for an M server balanced system [25] with total service demand D, i.e.,

$$R(N) = D + \frac{D}{M} \frac{R(N-1)}{R(N-1)+Z} (N-1) \tag{7}$$

where $R(N)$ denotes the response time as a function of the number of jobs in the system (N). Let $C(N) = R(N) + Z$ denote the mean cycle time (comparable to T). From Eq. 7 it follows that for small N the slope of $R(N)$ is (still) less[4] than D/M. Consequently, at the saturation point $N = N^*$, for which the deviation between $C(N)$ and its lower bound $C^l = D + Z$ is the largest, it holds $C(N^*) < C^l + (D/M)N^*$. With $N^* = (D + Z)/(D/M)$ it follows $C(N^*) < 2C^l$, which corresponds to the lower bound on α^*. Generating models with large M also implies a large value for D. In terms of the analogy this implies a relatively decreasing Z. Indeed, from Eq. 7 it is easily seen that $\lim_{Z \to 0} C(N^*) = 2C^l$. Thus the above theory intuitively supports the observations that the worst case average deviation of T^l relative to T is limited to a factor 2.

A first glance, a worst case 50 % underestimation may seem unacceptable regardless how much worse conventional static techniques may be. Note, however, that in most cases the relative error is much less than 50 %. More importantly, our approach enables a *fast*, first-order analysis and ranking of different design choices (easily entailing larger performance differences than a factor 2) with a *sustained* minimum accuracy *regardless* the system's parameter values.

4 Case Study

4.1 Introduction

In this section we present a case study in which the measured execution times of 14 random programs on a 4×4 mesh partition of a Parsytec GCel T800 transputer system[5] are compared with our predictions based on both simulation as well as our analytic technique. Each program involves the execution of a generic task executive utility that reads a random SP task graph description file

[3] For $N = 10$ the range of interest is $|\theta| < 0.5$. For $N \geq 1000$ the range still is $|\theta| < 2$.

[4] An accurate analysis of the balanced upper bound is given in [25]. However, for our purpose the above analysis suffices.

[5] Kindly made available by the Interdisciplinary Center for Computer-based Complex systems research Amsterdam (IC^{3}A).

(that specifies the user computation) and executes it in a data flow-style. The task graphs are generated by the same random generator as used in the previous section. Again, $N = 100$. Each task t_i executes a simple loop kernel with a random loop count given by w_i. A task t_i is randomly mapped onto a processor p_i according to a uniform distribution between 1 and $P = 16$. Thus, on average, 100/16 multiple tasks are mapped onto the same processor. Each task is executed by a separate (lightweight) thread scheduled dynamically by a node's run-time kernel. In order to enable true data flow execution, after each task has executed, the (same) produced data set (l_i bytes) is asynchronously sent to each successor task (thread) except when a successor resides locally. A typical example of this type of application is described in [16] where the task graphs represent sparse finite element computations in structural analysis. A more detailed description of the architecture of the task executive appears in [10].

Due to the dynamic approach towards task computation and communication, the case study (intentionally) provides an excellent example of the added value of serialization analysis compared to conventional static prediction techniques. Where static analysis inherently ignores the additional delay incurred by tasks sharing a processor, our approach naturally accounts for this delay by modeling task execution in terms of "processor contention". Apart from this, the use of non-blocking communication introduces the possibility of link contention as multiple task communications may share the communication link(s) simultaneously. Again, conventional static analysis makes no provision to account for the additional queuing delay, that may easily dominate performance (as will be shown). In our aim just to demonstrate the impact contention analysis may have, we simply consider coarse grain task execution where each task entails a large amount of computation ($O(10^6)$ floating point operations) as well as communication ($O(10^6)$ byte transfers). Hence, without loss of generality we can simply concentrate on computational and communication *bandwidths* rather than startup times (and other sources of overhead), which simplifies the discussion. However, the method applies to small communication volumes as well.

4.2 Computation Model

Let G denote the task graph to be executed, consisting of tasks $t_i, i = 1, \ldots, N$. Let L denote the PAMELA model of the executive, instantiated by data set G. Then L is given by starting with a graph topologically similar to G where each task i specifies a computation model $comp(i)$. In addition, every arc between a task i and j is expanded by a communication model $xfer(i, j, l)$ that accounts for the communication (l is message length in bytes) induced by that arc (discussed at length elsewhere). Since G is an SP graph the resulting program model L is also an SP graph which implies that L can be expressed in terms of one single (possibly complex) PAMELA expression. As a result, L can be directly compiled into a single expression T^l based on the application of Eq. 5. For instance, consider the following 3-tasks graph $G = t_1;\ (t_2 \parallel t_3)$. The PAMELA model of the (instantiated) executive is given by the expression $L = comp(1);\ ((xfer(1, 2, l_1);\ comp(2)) \parallel (xfer(1, 3, l_1);\ comp(3)))$.

The *comp* model represents the actual task computation. In our aim just to evaluate the analysis technique, we refrain from modeling the local loop kernel in detail and simply measure it as a whole. For the amount of work we consider ($w_i = 10^4 \ldots 10^6$ loops) the execution time increases linearly with w_i according to 6.1 μs per iteration (startup time negligible). Thus the execution time is expressed by the following (contention) model

$$comp(i) = \mathbf{seq}\ (k = 1, 6.1w_i/\tau_c)\ \mathbf{use}(p_i, \tau_c)$$

expressed in μs where τ_c denotes the basic CPU time slice. The (small) effect of multithreading overhead is automatically accounted for in the coefficient since during the calibration the kernel is run as a thread.

4.3 Communication Model

In conventional static techniques a linear **delay** model is often used to predict the transfer delay (e.g., [3, 13]) in terms of startup, hop count and bandwidth. Although precise for *isolated* transfers, these models do not account for additional queuing delay induced by concurrent traffic contending for the intermediate (sending, forwarding, receiving) link and node services. This is especially true in applications that exhibit a high level of parallel slackness (e.g., multiple threads/communications with latency hiding).

In the following we develop a simple transfer contention model that provides a first-order bandwidth approximation for *simultaneous* communications. In contrast to the task indices used above, in the following we will consider $xfer(s, r, l)$ where s and r denote sender and receiver processor, respectively ($i = p_s, j = p_r$). The communication service of the transputer system is based on a multiplexing scheme in which each 120 bytes of the message is packetized. Each packet is statically routed through the mesh in a pipelined fashion based on software forwarding pending a T9000 upgrade. With each physical link between neighboring transputers we will associate a *service complex* comprising a subsystem of physical (e.g., DMAs at both link ends) and/or semi-logical (software servers at both ends) resources. Without any loss of generality, we project the service complex at the receiving node of each link, as shown in Fig. 2. In the following we consider the communication system at the packet level which is the smallest level of granularity with respect to resource sharing. Although the service complex at each link comprises several software/hardware components, it can be modeled as to provide two services at the packet level that are subsequently denoted e and f (see Fig. 2).

The first service e represents the reception service at the packet destination involving the exclusive transfer of one packet across the link (duration $\tau_x = 108$ μs [10]), including the software overhead at both ends (e.g., moving, handshaking). The second service f represents the forwarding service (including intermediate byte storage and protocol overhead) required for a packet destined for a different node. Consequently, compared to e, f includes additional routing/forwarding work load (duration $\tau_y = 73$ μs). In general, a packet transfer

from node $s = n_1$ to $r = n_K$ will require forwarding at $n_2 \ldots n_{K-1}$ and one reception service at n_K. Both services are based on an underlying service, represented by the resource x that represents the basic link service that has to be shared. Consequently, e and f are *logical* resources (kernel servers) sharing the underlying link service. Typical for the PAMELA methodology, we use a material-

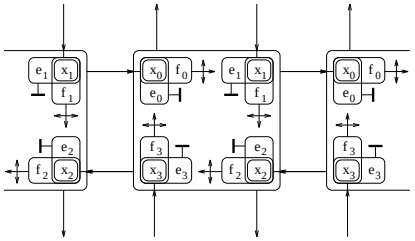

Fig. 2. Message-passing service model of the T800 transputer mesh

$\underline{s}$	$\underline{r}$	T^m	T	T^l	T^t
(0)	(1)	0.9	0.9	0.9	0.9
(0)	(2)	1.5	1.5	1.5	1.5
(0,0)	(1,1)	1.8	1.8	1.8	0.9
(0,0)	(1,2)	1.8	1.8	1.8	1.5
(0,0)	(2,2)	3.0	3.0	3.0	1.5
(0,0,0)	(1,2,2)	3.3	3.3	3.0	1.5
(0,0,0,0,0,0)	(1,1,1,2,2,2)	6.0	5.4	5.4	1.5

Fig. 3. Results for 10^6 byte concurrent communications (s)

oriented approach to model packet propagation, in which we model the entire transfer as a parallel section of contending tasks [10]. Let $n_k = s \ldots r$ denote the index of the K nodes involved in the pipeline route. Then the PAMELA model[6] is given by the simple expression

$$
\begin{aligned}
xfer(s,r,l) = \textbf{par}\ (i = 1, l/120)\ \{ & \\
& \textbf{seq}\ (k = 2, K-1)\ \{\ \textbf{use}(f_{n_k}, \tau_x + \tau_y)\ \|\ \textbf{use}(x_{n_k}, \tau_x)\ \} \\
& \textbf{use}(x_r, \tau_x) \\
\}\ &
\end{aligned}
$$

Note, that this model ignores startup delay and approximates the work load in terms of an integer number of packets. However, for large data communications this model suffices to accurately capture the effective bandwidth degradation when many virtual links are simultaneously active. The above model has been validated for many types of concurrent communications (equal message lengths) as well as random patterns (as discussed later on). The table in Fig. 3 shows a few typical results for (10^6 byte) data transfers involving only the first three nodes of the first mesh row (nodes 0, 1, and 2). The nodes that are simultaneously sending are expressed by the $\underline{s}$ vector, while the receivers are expressed by the $\underline{r}$ vector. Each pair (s_n, r_n) corresponds to one communication. Apart from the measured value T^m and the simulation result T the lower bound prediction T^l as well as the traditional static prediction T^t are listed. The simulation results

[6] The actual model is somewhat different in that it features a nested resource usage. In order to enable the application of our compile-time technique, a slight approximation has been applied that degrades accuracy by only a few percent. See [10] for details.

show that the *xfer* model is reasonably accurate. Only in a very few situations a limited deviation is measured (cf. last row). This optimistic prediction is due to the precise packet scheduling which is left undetermined in the PAMELA model. Nevertheless, the prediction accuracy of T^l is within 10 % whereas the traditional model yields errors up to hundreds of % [10].

4.4 Results

In this section we present the measurement results for the execution of the 14 random SP graphs $G_1 \ldots G_{14}$ on the transputer mesh. The computational work load w_i is i.i.d. uniformly over $[10^4, 10^6]$ (loops) that corresponds to an average total problem size of 305 s. In order for the communication to have a significant impact, the data size sent by each task to its successors is also i.i.d. uniformly over $[10^4, 10^6]$ (bytes) which corresponds to an average communication delay between 0.9 s and 1.5 s (per *isolated* transfer). As in the simulation experiment, the graphs are generated to cover the θ region of interest. Table 1 summarizes

G	T^m	T	T^t	θ	α	β
G_1	118.7	114.7	25.9	1.08	0.66	0.73
G_2	93.9	92.5	21.2	0.85	0.53	0.62
G_3	95.6	92.8	25.8	0.76	0.60	0.68
G_4	94.1	87.4	31.5	0.37	0.53	0.70
G_5	73.4	70.9	30.3	0.27	0.56	0.66
G_6	105.8	103.9	58.3	-0.19	0.55	0.55
G_7	98.4	87.0	47.4	-0.28	0.54	0.60

G	T^m	T	T^t	θ	α	β
G_8	89.2	87.1	52.5	-0.35	0.61	0.68
G_9	87.6	84.4	65.7	-0.73	0.78	0.78
G_{10}	109.5	106.4	79.5	-0.91	0.75	0.75
G_{11}	141.2	138.4	107.6	-1.23	0.78	0.80
G_{12}	149.5	144.8	125.0	-1.51	0.87	0.87
G_{13}	165.9	163.2	140.2	-1.62	0.86	0.86
G_{14}	172.3	171.0	165.4	-1.70	0.96	0.96

Table 1. Measurements vs. predictions (s)

the main results for each of the 14 programs. T^m denotes the measured execution time (s). T denotes the simulation result (s) of the corresponding PAMELA model L (variance is negligible). The result T^l of serialization analysis is represented in terms of θ, α, and β that are defined as before. The total number of resources involved in the simulation and analysis is $M = 144$ (P processors, $4P$ link, and $4P$ forwarding services). The T^t value (i.e., φ) has been included to demonstrate the (severe) prediction error of traditional static analysis.

The results show that the performance of the executive is indeed captured by the PAMELA model with reasonable accuracy. On average, T under-estimates T^m with about 4 % which is entirely due to the fact that the above communication model does not account for the effects of acknowledgement traffic and the minor decrease in computational bandwidth (explained in [10]). The results for α and β show that the average prediction error is well within 50 %. As expected, for relatively parallel graphs, the β values tend to be somewhat better

than the α values. The increase in accuracy for positive θ is somewhat less when compared to Fig. 1. This phenomenon is caused by the fact that for some task mappings parallel communications involving the same resource may occur at relatively concentrated points in time. Thus, the (link) resource usage is not always uniformly (randomly) distributed over the entire length of the computation (cf. Example 3). An elaborate explanation appears in [10]. Finally, note that for high contention levels the error in T^t is quite considerable.

5 Conclusion

We have outlined a simple compile-time technique that yields an analytic performance model of a parallel system. The work is inspired by the need for low cost, highly parametric models during the initial optimization loops in high-performance application design. Aimed to sustain an acceptable accuracy for *any* choice of system parameters, the method integrates critical path analysis typical for compile-time methods, with asymptotic bounding analysis from queuing theory. Whereas a single simulation run (T) requires hundreds of seconds on a typical workstation, the symbolic model T^l evaluates in a split second. Our results show that the worst-case penalty for large random task systems is a mere 50 % under-estimation on average.

To the best of our knowledge the study concerning the merit of fully integrating contention analysis within a static performance modeling technique for parallel system design has not yet been described. Although the accuracy of the method is inherently limited, the technique enables a fast, first-order analysis and ranking of different design choices with a sustained minimum accuracy regardless any system parameter value. Further validation involving real applications as well as (stochastic) extensions of the technique are under way.

Acknowledgements

It is a pleasure to express my indebtness to Prof. G.L. Reijns for supporting this research. The valuable comments of the referees are also gratefully acknowledged.

References

1. V.S. Adve, *Analyzing the Behavior and Performance of Parallel Programs.* PhD thesis, University of Wisconsin, Madison, Dec. 1993. Tech. Rep. #1201.
2. M. Ajmone Marsan, G. Balbo and G. Conte, "A class of Generalized Stochastic Petri Nets for the performance analysis of multiprocessor systems," *ACM Tr. on Comp. Syst.*, 2, May 1984, pp. 93–122.
3. M. Annaratone, C. Pommerell and R. Rühl, "Interprocessor communication and performance in distributed-memory parallel processors," in *Proc. 16th Symp. on Comp. Archit.*, May 1989, pp. 315–324.
4. D. Atapattu and D. Gannon, "Building analytical models into an interactive prediction tool," in *Proc. Supercomputing '89*, 1989, pp. 521–530.

5. V. Balasundaram, G. Fox, K. Kennedy and U. Kremer, "A static performance estimator to guide data partioning decisions," in *Proc. 3rd ACM SIGPLAN Symp. on PPoPP*, Apr. 1991.
6. T. Fahringer and H.P. Zima, "A static parameter-based performance prediction tool for parallel programs," in *Proc. 7th ACM ICS*, Tokyo, July 1993, pp. 207–219.
7. E. Gelenbe, E. Montagne, R. Suros and C.M. Woodside, "Performance of block-structured parallel programs," in *Parallel Algorithms and Architectures*, North-Holland, 1986, pp. 127–138.
8. A.J.C. van Gemund, "Performance prediction of parallel processing systems: The PAMELA methodology," in *Proc. 7th ACM ICS*, Tokyo, July 1993, pp. 318–327.
9. A.J.C. van Gemund, "Compiling performance models from parallel programs," in *Proc. 8th ACM ICS*, Manchester, July 1994, pp. 303–312.
10. A.J.C. van Gemund, "On the accuracy of compile-time performance prediction," Tech. Rep. 1-68340-44(1994)02, Delft University of Technology, Sept. 1994.
11. N. Götz, U. Herzog and M. Rettelbach, "Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras," *LNCS*, 729, Springer, 1993.
12. F. Hartleb and V. Mertsiotakis, "Bounds for the mean runtime of parallel programs," in *Proc. 6th Int. Conf. Modelling Techniques and Tools for Comp. Perf. Eval.*, Edinburgh, Sept. 1992, pp. 197–210.
13. R.W. Hockney, "Performance parameters and benchmarking of supercomputers," *Parallel Computing*, 17, 1991, pp. 1111–1130.
14. H. Jonkers, A.J.C. van Gemund and G.L. Reijns, "A probabilistic approach to parallel system performance modelling," in *Proc. 28th HICSS*, 1995, pp. 412–421.
15. W. Kreutzer, *System simulation, programming styles and languages.* Addison-Wesley, 1986.
16. H.X. Lin and H.J. Sips, "Parallel direct solution of large sparse systems in finite element computations," in *Proc. 7th ACM ICS*, Tokyo, July 1993, pp. 261–270.
17. V.W. Mak and S.F. Lundstrom, "Predicting performance of parallel computations," *IEEE Trans. on PDS*, 1, July 1990, pp. 257–270.
18. A.D. Maloney, V. Mertsiotakis and A. Quick, "Automatic scalability analysis of parallel programs based on modeling techniques," *LNCS*, 794, Springer, 1994, pp. 139–158.
19. C.L. Mendes, J-C. Wang and D.A. Reed, "Automatic performance prediction and scalability analysis for data parallel programs," in *Proc. 2nd. Workshop on Autom. Data Layout and Perf. Pred.*, Houston, Apr. 1995.
20. R.A. Sahner and K.S. Trivedi, "SPADE: A tool for performance and reliability evaluation," in *Modelling Techn. and Tools for Perf. Anal.* 1986, pp. 147–163.
21. V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors.* Pitman, 1989.
22. F. Sötz, "A method for performance prediction of parallel programs," *LNCS*, 457, Springer, 1990, pp. 98–107.
23. H. Wabnig and G. Haring, "Petri net performance models of parallel systems - methodology and case study," *LNCS*, 817, Springer, 1994, pp. 301–312.
24. K-Y. Wang, "A framework for static, precise performance prediction for superscalar-based parallel computers," in *Proc. 4th Int. Workshop on Compilers for Par. Comput.*, Delft, Dec. 1993, pp. 413–427.
25. J. Zahorjan *et al.*, "Balanced job bound analysis of queueing networks," *CACM*, 25, Feb. 1982, pp. 134–141.