PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 35, NO. 3, PP. 193-219 (1991)

PETRI NET BASED MODELING OF PARALLEL PROGRAMS EXECUTING ON DISTRIBUTED MEMORY MULTIPROCESSOR SYSTEMS

A. FERSCHA, G. HARING

Institut für Statistik und Informatik, Universität Wien

Received

Abstract

The development of parallel programs following the paradigm of communicating sequential processes to be executed on distributed memory multiprocessor systems is addressed. The key issue in programming parallel machines today is to provide computerized tools supporting the development of *efficient* parallel software, i.e. software effectively harnessing the power of parallel processing systems. The critical situations where a parallel programmer needs help is in expressing a parallel algorithm in a programming language, in getting a parallel program to work and in tuning it to get optimum performance (for example *speedup*).

We show that the Petri net formalism is higly suitable as a performance modeling technique for asynchronous parallel systems, by introducing a model taking care of the parallel program, parallel architecture and mapping influences on overall system performance. PRM-net (*P*rogram-*R*esource-*M*apping) models comprise a Petri net model of the multiple flows of control in a parallel program, a Petri net model of the parallel hardware and the process-to-processor mapping information into a single *integrated* performance model. Automated analysis of PRM-net models addresses correctness and performance of parallel programs mapped to parallel hardware. Questions upon the correctness of parallel programs can be answered by investigating behavioural properties of Petri net programs like liveness, reachability, boundedness, mutualy exclusiveness etc. Peformance of parallel programs is usefully considered only in concern with a dedicated target hardware. For this reason it is essential to integrate multiprocessor hardware characteristics into the specification of a parallel program. The integration is done by assigning the concurrent processes to physical processing devices and communication patterns among parallel processes to communication media connecting processing elements yielding an integrated, Petri net based performance model. Evaluation of the integrated model applies simulation and markovian analysis to derive expressions characterising the peformance of the program being developed.

Synthesis and decomposition rules for hierarchical models naturally give raise to use PRM-net models for *graphical, performance oriented* parallel programming, supporting top-down (*stepwise refinement*) as well as bottom-up development approaches. The graphical representation of Petri net programs visualizes phenomena like parallelism, synchronisation, communication, sequential and alternative execution. Modularity of program blocks aids reusability, prototyping is promoted by automated code generation on the basis of high level program specifications.

Keywords: performance evaluation, petri nets, concurrent programming, distributed memory multiprocessor systems, mapping.

1. Introduction

The performance of parallel systems (a parallel program executing on parallel hardware) is not only determined by the performance of the hardware itself (e.g. processor-, bus- or link-, memory access-speed, etc.), but also by the structure of the parallel program (the underlying algorithm, the communication pattern, synchronisation of tasks etc.) and the assignment of program parts (tasks that execute concurrently and cooperatively) to resources. Neither the approach of *resource oriented* performance evaluation of parallel processing systems, where only the system resources are modeled to some extent of detail (LAZOWSKA et al., 1984), nor the *program* or *process oriented* approach, where exclusively software aspects are subject to performance modeling and evaluation (VERNON, 1985.) (VERNON, 1987.), (GELENBE et al., 1986); (CHIMENTO and TRIVEDI, 1988) seem adequate to characterize the performance of parallel systems. The actual performance of such systems is always determined by the interdependencies between hardware performance and the requirements of parallel programs, i.e. the proper utilization of hardware performance by the program.

2. PRM-nets: An Integrated Performance Model

The main objective of the PRM-net (FERSCHA, 1990) approach is to give a modeling technique considering hardware, software and mapping as the performance influencing factors along with a computationally efficient and accurate method for the prediction of performance of parallel computations running on parallel hardware. The performance measures of interest are the (expected) *execution time* of the program and the degree of *resource utilization* at the hardware level. The problem in determining accurate figures of the execution time as well as resource utilization is the presence of two types of delay typically arising in parallel systems: *synchronisation delay* and *contention delay.* The first type of delay is due to the fact that tasks have different residence times and have to be synchronized (for example for communication), i.e. tasks finished earlier are made waiting for the completion of others. When several tasks request service from a single physical resource, only one of them can be granted service while the others have to wait. We call the delay arising because of resource contention *contention delay.* The PRM-net model explicitly accounts for these two critical determinants of execution time.

We restrict our considerations to distributed memory multiprocessor systems combining a variable amount of *processing elements* (PE's) connected to each other by point-to-point message links and working asynchronously in parallel when executing a global parallel program. The computational model of parallel programs is assumed to be sequential processes that communicate and synchronize with each other by (synchronous) message passing (like CSP (HOARE: 1978.)).

2.1 P-nets

A Petri net (REISIG, 1985); (MURATA, 1989) oriented process model is used to describe the structure and resource requirements of parallel programs. (Let further a Petri net $PN = (P, T, R, M_{start})$ be characterized as a set of *places* $P = \{p_1, p_2, \ldots, p_{n_P}\}$, a set of *transitions* $T = \{t_1, t_2, \ldots, t_{n_T}\}$, a *flow relation* $R \subseteq (P \times T) \cup (T \times P)$ and an *initial marking* $M_{Start} = (m_1, m_2, \ldots, m_{n_P})$, $m_i \in I\!N^+ \cup 0$. Let t_i^I (t_i^O) denote the set of *inputplaces* (*outputplaces*) of t_i.) A *process* is graphically represented by a transition, where inputplaces and outputplaces to the transition are used to model the current state of the process. A process t is *ready to get active*, if its corresponding process transition is *enabled*; the process gets *active* as the corresponding transition starts firing, and remains active for the firing duration. The process *terminates* by releasing tokens to outputplaces, therewith making subsequent processes (transitions) ready to get active (enabled). The Petri net specification of processes (components of parallel programs) is called a P-net. Processes can be arranged to get executed in sequence, in parallel, alternatively or iteratively. Concurrent processes are allowed to communicate on a synchronous message passing basis. In Fig. 1 (*a*) the P-net of a simple program constituted by two cyclic processes working in parallel and communicating with each other is given. It is built by a set of process transitions in a proper arrangement determining the dynamic behaviour of the program. The P-net is parametrized by associating *resource requirements* in terms of multisets of the services offered by physical resources to process transitions. To support hierarchical modeling, process compositions can be folded to form a single, *compound* process, graphically represented by a single transition (box), by aggregation of the resource requirements of all the constituting processes. The opposite is also possible: a single process can be refined by specifying its internal structure in terms of complex process compositions. Fig. 1 (*b*) shows that process comp 1 is constituted by three subprocesses sub 1, sub 2 and sub 3, each of them requiring a certain amount of the physical resource services π_1, π_2 and π_3. The type of resource p (processor) is also specified. When aggregating

sub 1, sub 2 and sub 3 to comp 1, the resource requirements are cumulated; decomposition of comp 1 on the other hand can help to get a more precise figure of its resource consumption by investigating its internals.

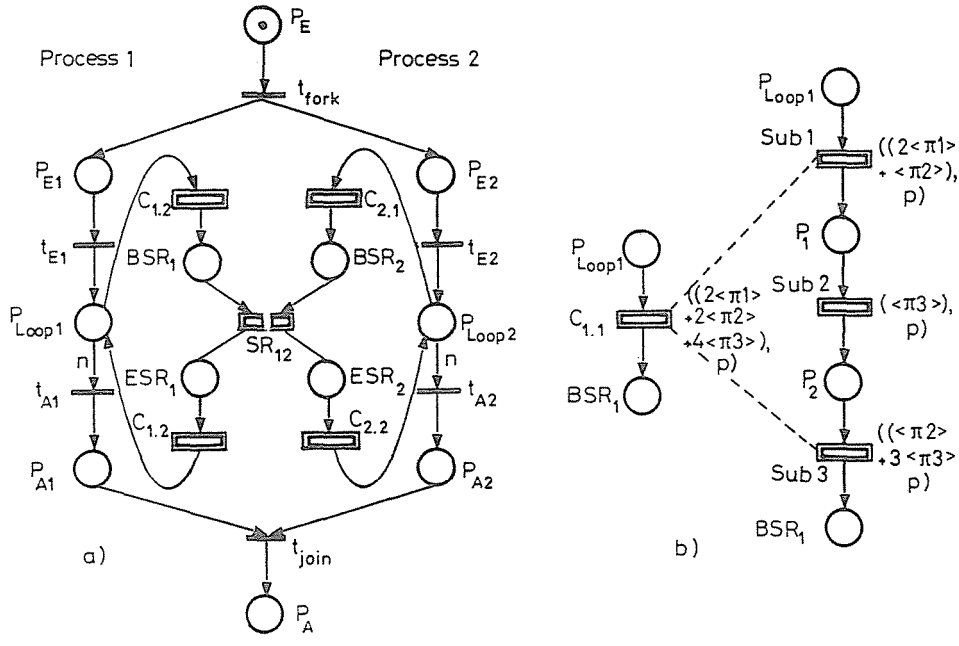


Fig. 1. P-net model of a parallel program

Further we give a formal presentation of P-nets and valid process compositions.

Definition 2.1 (P-net) *A P-Net is a process graph* $\mathbf{P} = (P^{\mathbf{P}}, T^{\mathbf{P}}, R^{\mathbf{P}}, M_{start}, \mathcal{R})$ *where:*

(*i*) $(P^{\mathbf{P}}, T^{\mathbf{P}}, R^{\mathbf{P}})$ *is the underlying net with* $P^{\mathbf{P}} = \{p_1, p_2, \ldots, p_{n_P}\}$ *and* $T^{\mathbf{P}} = \{t_1, t_2, \ldots, t_{n_T}\}$. *The elements* $t_i \in T^{\mathbf{P}}$ *are called processes.*

(*ii*) $\exists p_E \in P^{\mathbf{P}}$ *with* $p_E \notin t^O \forall t \in T$. p_E *is the entry place of* $\mathbf{P}$

(*iii*) $\exists p_A \in P^{\mathbf{P}}$ *with* $p_A \notin t^I \forall t \in T$. p_A *is the termination place of* $\mathbf{P}$

(*iv*) $\forall t \in T^{\mathbf{P}}$: t *is either a primitive or a compound process*

(*v*) *The direction of each* $r \in R^{\mathbf{P}}$ *defines the direction of the flow of control.*

(vi) $M_{start} = \{m_1, m_2, \ldots, m_{n_P}\}$ *is the initial marking with* $m_E = 1$ *and* $m_i = 0 \quad \forall p_i \in P \setminus p_E$.

(vii) $\mathcal{R} = \{\varrho_1, \varrho_2, \ldots, \varrho_{n_T}\}$ *is the set of resource requirements of* $\{t_1, t_2, \ldots, t_{n_T}\}$ *where* ϱ_i, *the requirement of process* t_i, *is a set of tuples* (σ, ω) *and* σ *is a multiset of primitive processes requiring a resource of type* ω.

Primitive processes (drawn by bar transitions) are deterministic in behaviour, i.e. they have deterministic resource requirements in that they always require the same amount of services from the physical resources. They are no further divisible and hence represent the building blocks of a parallel program. We denote the set of all *possible* primitive processes within a parallel program by $\Pi = \{\pi_1, \pi_2, \ldots, \pi_{n_\Pi}\}$. The *graph* of a primitive process is a single transition (represented by a bar) with an *entry place* and one *termination place*. Complex structures of parallel programs are represented by *valid* compositions of primitive processes or in turn of process compositions. A sufficient set (to model any kind of block structured parallel program) of *valid* process compositions is given in terms of *process graph compositions* (see Fig. 2).

Definition 2.2 (Sequential) *A sequential process composition is a process graph* $\mathbb{P}_{seq} = (P^{\mathbb{P}}, T^{\mathbb{P}}, R^{\mathbb{P}}, M_{start}, \mathcal{R})$ *with* $P^{\mathbb{P}} = \{p_1, p_2, \ldots p_{n_T+1}\}$, $T^{\mathbb{P}} = \{t_1, t_2, \ldots t_{n_T}\}$, $M_{start} = \{1, 0, \ldots, 0\}$ *and* $\mathcal{R} = \{\varrho_1, \varrho_2, \ldots \varrho_{n_T}\}$ *where*

$$t_i^I = \begin{cases} p_E = p_1 & i = 1 \\ t_{i-1}^O = p_i & 2 \leq i \leq n_T \end{cases}, \qquad t_i^O = \begin{cases} p_A = p_{n_T+1} & i = n_T \\ t_{i+1}^I = p_{i+1} & 1 \leq i \leq n_T - 1 \end{cases}.$$

$\mathbb{P}_{seq}$ *is aggregated to a compound process graph* $\mathbb{P} = (P^{\mathbb{P}} = \{p_E, p_A\}, T^{\mathbb{P}} = \{t\}, R^{\mathbb{P}} = \{(p_E, t), (t, p_A)\}, M_{start} = \{1, 0\}, \mathcal{R} = \{\varrho\})$, *where* $\varrho = \bigcup_{i=1}^{n_\omega}(\sigma_i, \omega_i)$ *denotes the set of all tuples of multisets of primitive processes and resource types respectively if* n_ω *different types of resources are required by the processes* $t_1, t_2, \ldots t_{n_T}$.

Let $\varrho_i = \bigcup_{k=1}^{n_{\omega_i}}(\sigma_k, \omega_k)$ *be the resource requirement of the process* $t_i \in T^{\mathbb{P}}$, *assuming that* n_{ω_i} *is the number of different types of resources required by* t_i. $\Omega_i = \{\omega_1, \omega_2, \ldots, \omega_{n_{\omega_i}}\}$ *is the set of all types of resources required by* t_i, and $\Omega = \bigcup_{i=1}^{n_T} \Omega_i$ *the set of all types of resources wanted by the whole composition. The compound resource requirement (of the whole composition) is*

$$\varrho = \bigcup_{j|\omega_j \in \Omega} \Big(\sum_{i|(\sigma_k, \omega_j) \in \varrho_i} \sigma_k, \omega_j \Big),$$

where $\sum$ is a symbol for the sum of multisets[1]

By definition 2.2 consecutive (sub-)processes t_i, t_{i+1} are forced to be executed *sequentially*, i.e. t_{i+1} becomes *ready to get active* as soon as t_i has *terminated* (see Fig. 2(*b*)). When aggregating a set of sequential processes to a single compound process as a matter of *abstraction*, the resource requirements of the constituent processes $\{t_1, t_2, \ldots, t_{n_T}\}$ have to be cumulated with respect to different types of physical resources: All the multisets of primitive processes requiring the same type of resource are cumulated and associated to that resource type to form a tuple. Hence ϱ of the resulting compound process comprises the union of tuples for all different types of resources. Definition 2.2 on the other hand implicitely defines decomposability of process transitions as a matter of *refinement*.

Definition 2.3 (Parallel) *The process graph*

$$\mathbf{P}_{par} = (P^{\mathbf{P}}, T^{\mathbf{P}}, R^{\mathbf{P}}, M_{start}, \mathcal{R})$$

of a parallel process composition with $P^{\mathbf{P}} = \{p_0, p_1, \ldots p_{2n_T+1}\}$ *and* $T^{\mathbf{P}} = \{t_f, t_1, t_2, \ldots, t_{n_T}, t_j\}$ *comprises two additional processes: a fork process* t_f *and a join process* t_j *such that*

$$t_f^I = p_E = p_0, \quad t_f^O = \bigcup_{i=1}^{n_T} t_i^I = \bigcup_{i=1}^{n_T} p_i, \tag{i}$$

$$t_j^I = \bigcup_{i=1}^{n_T} t_i^O = \bigcup_{i=1}^{n_T} p_{2i}, \quad t_j^O = p_A = p_{2n_T+1}, \tag{ii}$$

$$t_i^I = p_i, \quad t_i^O = p_{2i} \quad \forall \ t_i, \ 1 \le i \le n_T. \tag{iii}$$

$\mathbf{P}_{par}$ *is aggregated to a compound process graph* $\mathbf{P} = (P^{\mathbf{P}} = \{p_E, p_A\}, T^{\mathbf{P}} = \{t\}, R^{\mathbf{P}} = \{(p_E, t), (t, p_A)\}, M_{start} = \{1, 0\}, \mathcal{R} = \{\varrho\})$ *in the same way as* $\mathbf{P}_{seq}$, *where the compound resource requirement calculates as*

$$\varrho = \bigcup_{j|\omega_j \in \Omega} \Big(\sum_{i|(\sigma_k, \omega_j) \in \varrho_i} \cdot \ \sigma_k, \omega_j \Big).$$

[1] A *multiset* of Π is a linear combination of elements in Π with integer coefficients denoting the multiplicity of elements. The set $\mathcal{B}(\Pi)$ is the set of all *multisets* over Π. Addition of two *multisets* $a, b \in \mathcal{B}(\Pi)$ is defined by $(a + b) : x \mapsto a(x) + b(x), \quad x \in \Pi$, multiplication by a scalar z is defined by $(za) : x \mapsto za(x), \quad x \in \Pi$) .

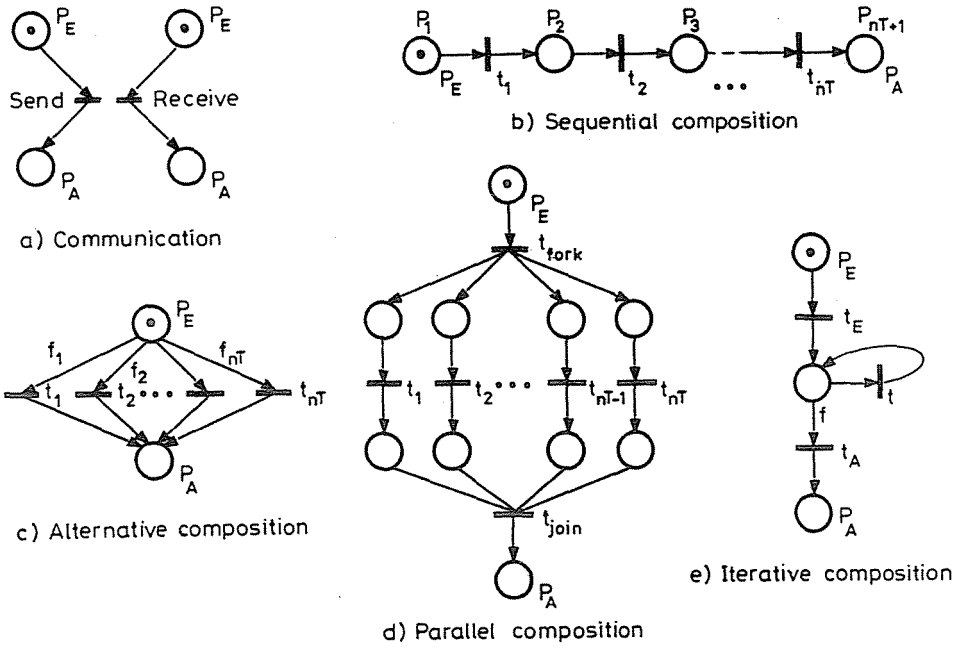


Fig. 2. Graphs of process compositions

In the graph of a *parallel* process composition $\mathbb{P}_{par}$ the only input place to t_f is p_E and the only output place of t_j is p_A. The outputplaces of t_f are $t_f^O = \bigcup_{i=1}^{n_T} t_i^I$, the inputplaces of t_j are $t_j^I = \bigcup_{i=1}^{n_T} t_i^O$. Fig. 2($d$) shows that the parallel processes $(t_1, t_2, \ldots, t_{n_T})$ are allowed to get active concurrently if the fork process has terminated. The join process t_j gets active as soon as the last $t \in \{t_1, t_2, \ldots, t_{n_T}\}$ has terminated. We assume that neither the fork-, nor the join-process require resources for their execution.

Communication Concurrent processes, i.e. processes having a fork-transition in common are allowed to communicate with each other. Interprocess communication is expressed by matching send (!) and receive (?) primitives and takes place if both processes issuing these commands are ready for the message exchange at the same time ('rendez-vous' synchronisation). We recognize the operations ! and ? to be primitive processes and model the communication of processes by a *synchronisation transition* (see Fig. 2(a)).

Definition 2.4 (Alternative) *The graph of an alternative process composition is defined by* $\mathbf{P}_{alt} = (P^{\mathbf{P}}, T^{\mathbf{P}}, R^{\mathbf{P}}, \mathcal{B}_{P^{\mathbf{P}} \times T^{\mathbf{P}}}, M_{start}, \mathcal{R})$ *where:*

(i) $\mathcal{B}_{P^{\mathbf{P}} \times T^{\mathbf{P}}} : P^{\mathbf{P}} \times T^{\mathbf{P}} \mapsto \mathbf{F}_s$ *assigns guards* $f \in \mathbf{F}_s$ *to elements* $(p,t) \in P^{\mathbf{P}} \times T^{\mathbf{P}}$.

(ii) $T^{\mathbf{P}} = \{t_1, t_2, \ldots t_{n_T}\}$ *is the set of alternative processes.*

(iii) $P^{\mathbf{P}} = \{p_E, p_A\}$ *with* $t_i^I = p_E \quad \forall t_i \in T^{\mathbf{P}}$ *and* $t_i^O = p_A \quad \forall t_i \in T^{\mathbf{P}}$

Let $q_i = \mathcal{P}\{f_{(p_E,t_i)} = \mathbf{t}\}$ *be the probability that guard* f *assigned to the arc* (p_E, t_i) *is* **true** $(\sum_{i=1}^{n_T} q_i = 1)$, *then* $\mathbf{P}_{alt}$ *is aggregated to a compound process with resource requirement:*

$$\varrho = \bigcup_{j|\omega_j \in \Omega} \Big(\sum_{i|(\sigma_k,\omega_j)\in\varrho_i} q_i \sigma_k, \omega_j \Big).$$

The graph of an *alternative* process composition (see Fig. 2 (*c*)) $\mathbf{P}_{alt}$ has only two places $P^{\mathbf{P}} = \{p_E, p_A\}$, i.e. an entry-place and an exit place, allowing at most one of the alternative processes $(t_1, t_2, \ldots, t_{n_T})$ to get active at the same time. If none of the guards $f \in \{f_1, f_2, \ldots, f_{n_T}\}$ is **true**, none of the corresponding processes can get active. If more than one of them are **true** simultaneously, one of the corresponding processes is selected at random to get active (nondeterministic choice). The aggregation of $\mathbf{P}_{alt}$ hence is probabilistic.

Definition 2.5 (Iterative) $P_{iter} = (P^{\mathbf{P}}, T^{\mathbf{P}}, R^{\mathbf{P}}, M_{start}, \mathcal{R})$ *is the graph of an iterative process composition if:*

(i) $\exists p_L \in P^{\mathbf{P}}$ *and* $\exists t_E, t_A \in T^{\mathbf{P}}$ *with:*

(1) $t_E^I = p_E$, $t_E^O = p_L$

(2) $t_A^I = p_L$, $t_A^O = p_A$

(ii) $\exists t \in T^{\mathbf{P}}$ *with* $t^I = t^O = p_L$

(iii) $f \in \mathbf{F}_s$ *is a guard assigned to the arc* (p_L, t_A).

Let q *be an estimate of the number of loop iterations.* $\mathbf{P}_{iter}$ *aggregates to a single, compound process with*

$$\varrho = \bigcup_{k=1}^{n_\omega} (q\sigma_k, \omega_k).$$

The graph of an iterative process composition $\mathbf{P}_{iter}$ (see Fig. 2 (*e*)) contains a *'loop place'* p_L with entry- and exit-processes t_E and t_A, which are (for simplicity) assumed to require no resource services. The 'loop body' t is activated until the termination condition f holds.

2.1.1 Stochastic Aggregation in P-Nets

The modular conception of process compositions directly supports stochastic analysis on the basis of individual probability distributions of *stochastic resource requirements.* In cases where the amount of resources of a given type cannot be given deterministically but only in terms of a Coxian phase-type probability distribution (exponential, hyperexponential, k-stage Erlang etc.) (TRIVEDI, 1984), the aggregation to requirements of a compound process can be done by folding probability distributions.

Let $T^{\mathrm{P}} = \{t_1, t_2, \ldots t_{n_T}\}$ be a *sequential process composition* $\mathbf{P}_{seq}$, with resource requirements $\mathcal{R} = \{\varrho_1, \varrho_2, \ldots \varrho_{n_T}\}$ and $\varrho_i = \bigcup_{k=1}^{n_{\omega_i}} (\sigma_k, \omega_k)$ as above. Let $\{X_1^\pi, X_2^\pi, \ldots X_{n_T}^\pi\}$ be random variables denoting the sum of multiplicities of the primitive process π in all the σ_k of ϱ_i, and let them be distributed according to $\{F_1^\pi(t), F_2^\pi(t), \ldots F_{n_T}^\pi(t)\}$, $F_i^\pi(t)$ being of the Coxian phase-type, then the multiplicity X_{seq}^π for (every) π in the process composition follows

$$X_{seq}^\pi \sim F_{seq}^\pi(t) = \bigotimes_{i=1}^{n_T} F_i^\pi(t),$$

($\otimes$ stands for the *convolution*: $F_i^\pi(t) \otimes F_{i+1}^\pi(t) = \int_0^t F_{i+1}^\pi(t-x)\ dF_i^\pi(x)$.) The same folding would apply to a parallel process composition being aggregated to a compound process ($X_{par}^\pi \sim F_{par}^\pi(t) = \bigotimes_{i=1}^{n_T} F_i^\pi(t)$). If the selection probabilities $p_i = P[f_{(p_E,t_i)} = \mathrm{t}]$ of the alternatives $T^{\mathrm{P}} = \{t_1, t_2, \ldots t_{n_T}\}$ of an *alternative process composition* are known, then the multiplicity X_{alt}^π for π under the assumption of definitive choice (see below) would follow

$$X_{alt}^\pi \sim F_{alt}^\pi(t) = \sum_{i=1}^{n_T} p_i \cdot F_i^\pi(t).$$

X_{iter}^π for π in an *iterative process composition* would be

$$X_{iter}^\pi \sim F_{iter}(t) = \bigotimes_{i=1}^{q} F_1^\pi(t),$$

if q iterations over process t_1 ($X_1^\pi \sim F_1^\pi(t)$) are to be expected (proper termination).

2.1.2 Pr/T-Net Specification of P-Nets

For process compositions with replicated structures P-nets can be specified in terms of Pr/T-nets (GENRICH, 1987) (GENRICH, 1988), to get a more

concise and easier to understand specification of compositions of huge sets of *identical* processes. This is of practical importance especially for sequential and parallel processes compositions.

A $\mathbf{P}_{seq}$ with $T^{\mathbf{P}} = \{t_1, t_2, \ldots t_{n_T}\}$ as in Definition 2.2 is called a *replicated sequential process composition* if all $t_i \in T^{\mathbf{P}}$ are instances of one and the same process t (see Fig. 3(a)). In this case $\mathbf{P}_{seq}$ is denoted by a Pr/T-net as in Fig. 3(b). The variable predicate $P_E x$ initially holds for all individual processes $\sum_{1 \le i \le n_T} < i >$. The transition selector $x = \min_i < i >$ of t selects processes by increasing identifiers to get executed (fired) by t (removed from $P_E x$ and deposited into $P_A x$). (In general: an individual $< i >$ is allowed to leave $P_E x$ only if $< j >$, $j = i - 1$ has arrived in $P_A x$). For brevity we denote the Pr/T-net in Fig. 3(b) as in Fig. 3(c), inscribing the 'range' of replication in parentheses. Multiple replications are denoted as in Fig. 3(d), expressing that a replicated sequential process composition is in turn replicated.

The same is possible for replicated parallel process compositions with notations as in Fig. 4. Note that t has no transition selector, i.e. t can fire concurrently in all modes of x. The range is enclosed in brackets (Fig. 4(c)).

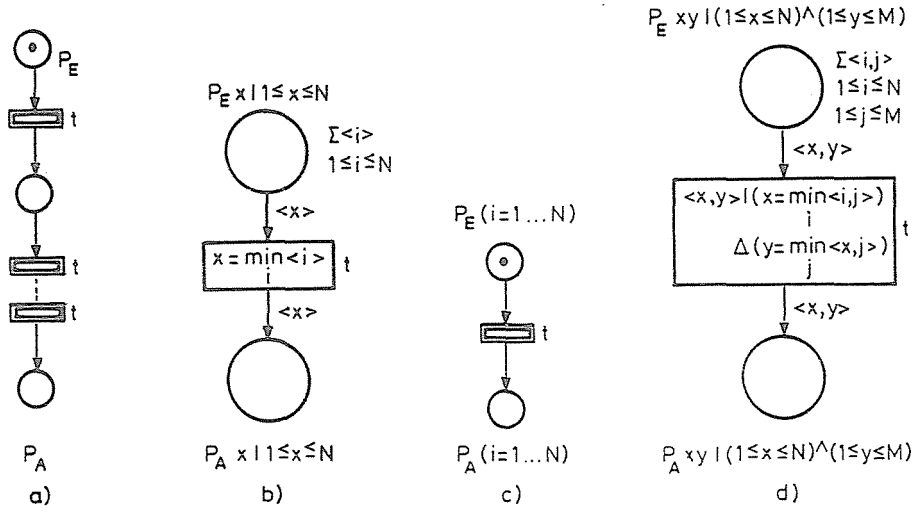


Fig. 3. Replicated sequential process compositions

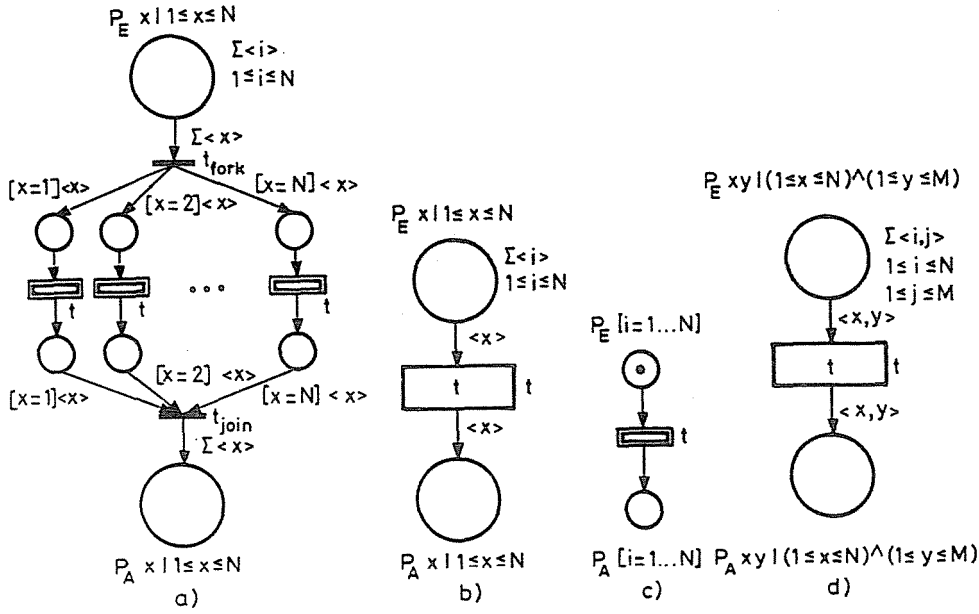


Fig. 4. Replicated parallel process compositions

The following relation among individual tokens in multiple (hierarchical) replications and plain tokens guarantees consistency with plain P-nets: Let $u_k \leq i_k \leq o_k$ be the range of individuals in the k-th replication then

$$\sum_{(u_k \leq i_k \leq o_k)} < i_1, i_2, \ldots, i_k > = < i_1, i_2, \ldots, i_{k-1} >$$

holds with $\sum_{(u_1 \leq i_1 \leq o_1)} < i_1 > = \bullet$, i.e. availability of all individuals of single replication is equivalent to the presence of the plain token.

Firing Rules P-nets have the same firing behaviour as Place/Transition nets (REISIG, 1985); (REISIG, 1987) with the exception of the guarded transition firing in alternative or iterative process compositions. In both cases the conflict among enabled transitions is solved depending on data in the parallel program - usually performed (interactively) by a user interested

in the behaviour of the program when simulating the P-net. For the purpose of investigating structural or performance properties of the P-net it is no longer necessary to consider guarded processes, one is rather interested in what the 'real alternatives' and the 'real number of loop iterations' are (in contrast to the situation where P-nets are acting as high level specifications of parallel programs). Hence we restrict our further considerations to P-nets meeting the following assumptions:

Definitive Choice Whenever the entry—place of an alternative process composition is reached, at least one guard is **true**.

Definitive Termination The termination condition of an iterative process composition will definitely hold after a finite number of iterations.

According to these assumptions we can rewrite a general P-net to one with unguarded alternative processes and the arc (p_L, t_A) in an iterative process composition being a *counter arc* (DUGAN et al, 1984) denoting a specific number of loop iterations. (In the example P-net in Fig. (*a*) (P_{Loop1}, t_{A1}) is a *counter arc* labeled by n, denoting that t_{A1} is enabled if there are $n+1$ tokens in P_{Loop1}. $(P_{Loop1}, C_{1.1})$ is the corresponding *counter-alternate arc* enabling $C_{1.1}$ when the count in P_{Loop1} is between 1 and n inclusively. Firing of $C_{1.1}$ is allowed whenever a new token enters P_{Loop1}, does not remove tokens from there, but places tokens to subsequent outputplaces. An iterative process composition can be enrolled to a sequential process composition constituted by n replicates of the 'loop body'.) P-nets rewritten in this way have exactly the same firing rules like Place/Transition nets and will be subject of all further investigations.

2.1.3 Properties of P-Nets

Every valid P-net is *safe*, i.e. $m_i \leq 1 \forall p_i \in P^{\mathbf{P}}$ in every marking $M_{start}M_i$ reachable from the initial marking

$$M_{start} = \{m_E, m_2, \ldots, m_A\} = \{1, 0, \ldots, 0\},$$

and satisfies the *free choice condition*

$$\forall \ (p_i, t_j) \in (P^{\mathbf{P}} \times T^{\mathbf{P}}), \quad p_i^O = \{t_j\} \vee t_j^I = \{p_i\}.$$

by definition. For the investigation of behavioural properties of P-nets we first define the following extension to P-nets:

Definition 2.6 (Extended P-net) *Let* $\mathbf{P} = (P^{\mathbf{P}}, T^{\mathbf{P}}, R^{\mathbf{P}}, M_{start}, \mathcal{R})$ *be a P-net. Its extension to* $\overline{\mathbf{P}} = (\overline{P}^{\mathbf{P}}, \overline{T}^{\mathbf{P}}, \overline{R}^{\mathbf{P}}, M_{start})$, *with:*

$$\overline{P}^{\mathbf{P}} = P^{\mathbf{P}}, \quad \overline{T}^{\mathbf{P}} = T^{\mathbf{P}} \cup t_e, \quad \overline{R}^{\mathbf{P}} = R^{\mathbf{P}} \cup (p_A, t_e) \cup (t_e, p_E)$$

is called the extension of $\mathbf{P}$.

The extension of the P-net in Fig. 1 is constructed by adding a transition t_e and arcs from P_A and to P_E thus 'looping back' the token flow from P_A to P_E.

Corollary 2.1 *The extension* $\overline{\mathbf{P}}$ *of a valid P-net* $\mathbf{P}$ *is a strongly connected Free Choice Net (FC-P-net) (*MURATA, 1989*) (It meets (1) and* $\mid p^I \mid \geq 1$, $\mid p^O \mid \geq 1, \forall\ p \in P$*). If there are no alternative or iterative process compositions in* $\mathbf{P}$, *then* $\overline{\mathbf{P}}$ *is a strongly connected Marked Graph (MG-P-net) (*MURATA, 1989.*)* ($\forall\ p \in P$, $\mid p^I \mid\ = \mid p^O \mid\ = 1$)

To determine whether a parallel program is free of static deadlocks (*termination*) we define:

Definition 2.7 (Termination) *A P-net* $\mathbf{P}$ *with*

$$M_{start} = \{m_E, m_2, \ldots, m_A\} = \{1, 0, \ldots, 0\}$$

is said to terminate if

$$M_{stop} = \{m_E, m_2, \ldots, m_A\} = \{0, 0, \ldots, 1\}$$

is reachable from every marking $M_{start} M_i$.

Theorem 2.1 *Let* $\overline{\mathbf{P}}$ *be the extension of* $\mathbf{P}$. $\mathbf{P}$ *terminates if* p_E *as well as* p_A *are covered by all (minimal) place invariants of* $\overline{\mathbf{P}}$[2] *(Proof omitted).*

The theorem says that if an invariant does not cover p_E as well as p_A then two undesirable situations appear depending on whether the invariant is initially marked or not: if it is marked, then there is a cycle in the process causing livelock, if not, then (some) subprocesses can never get active (dead transitions). *Fig. 5* shows the incidence matrix representation $I^I_{\overline{\mathbf{P}}}$ of the P-net in *Fig. 5(a)* along with the place invariants $\vec{i}$ found by solving $I^I_{\overline{\mathbf{P}}} \cdot \vec{i} = \vec{0}$. Both $\vec{i}_1$ and $\vec{i}_2$ cover p_E as well as p_A, hence $\mathbf{P}$ will *terminate*. The number of invariants (2) obtained simultaneously expresses the degree of exploitable parallelism; $\mathbf{P}$ can (potentially) execute on two different PE's.

[2]Because of the free choice property one could also prove termination with the deadlock-trap method, but this would cause higher computational complexity.

	t_{fork}	t_{E1}	$C_{1.1}$	$C_{1.2}$	t_{A1}	SR_{12}	t_{E2}	$C_{2.1}$	$C_{2.2}$	t_{A2}	t_{join}	t_e	i_1	i_2
P_E	−1											1	1	1
P_{E1}	1	−1											1	
P_{Loop1}		1	−1	1	−1								1	
P_{A1}					1						−1		1	
BSR_1			1			−1							1	
ESR_1				−1		1							1	
P_{E2}	1						−1							1
P_{Loop2}							1	−1	1	−1				1
P_{A2}										1	−1			1
BSR_2						−1		1						1
ESR_2						1			−1					1
P_A											1	−1	1	1

Fig. 5, Incidence matrix of $\overline{\mathbf{P}}$ for the P-net in Fig. 1(*a*).

2.2 PRM-Nets

When modeling the performance of a parallel application the analyst should not have to change the formalism when modeling different aspects of the whole system. To this end we propose to use the Petri net formalism also for modeling resources and usage of resources by software processes. The characteristic of parallel processing systems is that resources like memory, processing elements and communication devices appear with some multiplicity (thus forming pools of resources) allowing their concurrent usage. The pool of resources, their connectivity and interactivity as well as their potential performance are modeled by R-nets (resource nets). We assume that for every resource in a parallel processing environment one can identify its *type* and a set of *services* (primitive processes) offered to applications.

Definition 2.8 *Let* $\Phi = \{\rho_1, \rho_2, \ldots, \rho_{n_\Phi}\}$ *be the set of resources and* $\Pi_\rho \subseteq \Pi$ *the set of primitive processes that can be executed by* ρ. *A R-net is a resource graph* $\mathbf{R} = (P^{\mathbf{R}}, T^{\mathbf{R}}, R^{\mathbf{R}}, M_{init}, \mathcal{T})$ *where:*

(*i*) $P^{\mathbf{R}} = \{h_1, h_2, \ldots, h_{n_\Phi}\}$ *is a finite set of 'home' places for the resources* $\rho \in \Phi$.

(*ii*) $T^{\mathbf{R}} = \{t_1, t_2, \ldots, t_{n_T}\}$ *is a finite set of transitions and* $R^{\mathbf{R}} \subseteq (P^{\mathbf{R}} \times T^{\mathbf{R}}) \cup (T^{\mathbf{R}} \times P^{\mathbf{R}})$ *is a flow relation.*

(*iii*) $M_{init} : P^{\mathbf{R}} \mapsto \Phi$ *initially assigns resource tokens* $\rho_i \in \Phi$ *to 'home' places* $h_i \in P^{\mathbf{R}}$.

(*iv*) $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_{n_\Phi}\}$ *is a set of functions, each* $\tau_i : \Pi_{\rho_i} \mapsto \mathbb{Z}$ *assigning deterministic timing information to primitive processes executable by* ρ_i.

Every resource in the system is modeled by a token in the R-net having its proper *home place*. Presence of the (resource-) token in that place indicates the availability of the resource (idle to serve). Arcs in $R^{\mathbf{R}}$ describe the direction of resource flows and help, together with transitions in $T^{\mathbf{R}}$, to model interactions and interdependencies between resources. With every resource ρ is associated a set of primitive processes Π_ρ along with timing information $\tau(\pi)$ for each $\pi \in \Pi_\rho$. $\tau(\pi)$ is the time it would take ρ to serve π.

The assignment of parallel (software) processes to resources is expressed by a set of arcs combining P-nets and R-nets to a single Petri net which we call PRM-net (program-resource-mapping-net).

Definition 2.9 *A mapping is a set of arcs* $\mathcal{M} \subseteq (P^{\mathbf{R}} \times T^{\mathbf{P}}) \cup (T^{\mathbf{P}} \times P^{\mathbf{R}})$ *where*

(*i*) $P^{\mathbf{R}} \times T^{\mathbf{P}}$ *is the set of arcs leading from home places to processes such that if* $(h_i, t_j) \in P^{\mathbf{R}} \times T^{\mathbf{P}}$ *and the type of* ρ_i *is* ω, *then* $\pi \in \Pi_{\rho_i}\ \forall \pi \in \sigma$ *for all tuples* $(\sigma, \omega) \in \varrho_j$.

(*ii*) $T^{\mathbf{P}} \times P^{\mathbf{R}}$ *is the set of arcs leading from processes to home places with* $(t_j, h_i) \in T^{\mathbf{P}} \times P^{\mathbf{R}} \Rightarrow \exists (h_i, t_j) \in P^{\mathbf{R}} \times T^{\mathbf{P}}$ *as in* (*i*).

Assigning home places to process transitions is allowed only if all the primitive processes required by the transition are offered as services by the resource (the resource also has to have the desired type). We finally call the triple $PRM = \{\mathbf{P}, \mathbf{R}, \mathcal{M}\}$, the combination of a P-net and a R-net by a mapping to a single, *integrated* net model, the PRM-net model.

See *Fig. 6* for sample mappings of the P-net in *Fig. 1*(*a*) to a set of resources. Assume that the compound processes $C_{1.1}$, $C_{1.2}$, $C_{2.1}$ and $C_{2.2}$ all require resources of type **p** (processor) for being served, and Processor 1 and Processor 2 being resources of that type, then the process transitions are allowed to get mapped to them if they can be served completely. *Fig. 1*(*a*) shows the assignment of the parallel program to two processors connected to each other by a communication link, fully exploiting the inherent parallelism. During one iteration step $C_{1.1}$ and $C_{2.1}$ can be executed concurrently by Processor 1 and Processor 2. The communication process SR_{12} synchronizes the two processes when being executed by a link type resource. (The link resource is made available only if both processes are ready for communication, i.e. there is a flow token both in BSR_1 as well as in BSR_2. This is expressed by bidirectional arcs between BSR_1 (BSR_2) and the transition preceding place Link, enabling this transition only if BSR_1 and BSR_2 are marked. Both are not drawn in *Fig. 1*(*a*) for sake of readability.) Finally $C_{1.2}$ and $C_{2.2}$ are executed concurrently. In the sec-

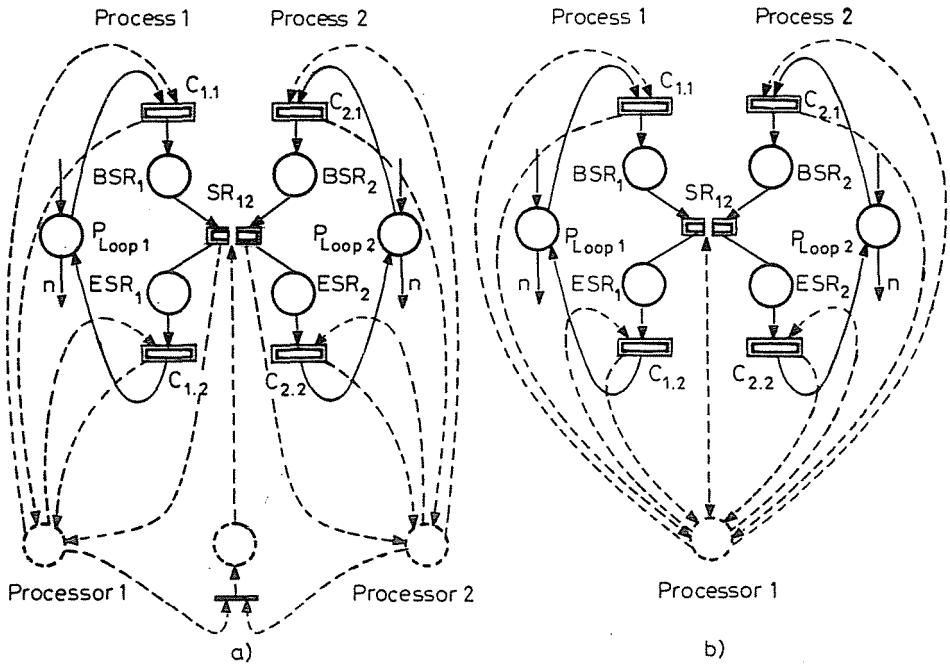


Fig. 6. PRM-nets for two processor (a), and one processor (b) mapping

ond case (*Fig. 1*(b)) the program is mapped (without change in the software structure) to a single processor capable of serving a communication process. All the processes $C_{1.1}$, $C_{1.2}$, $C_{2.1}$ and $C_{2.2}$, as well as SR_{12} are executed sequentially. Processes are scheduled according to their flow precedence in the P-net: When starting a new loop iteration only $C_{1.1}$ and $C_{1.2}$ are ready to get active. This is due to $C_{1.1}$ and $C_{1.2}$ being enabled because of $m_{Loop1} = 1$, $m_{Loop2} = 1$ and the residence of the resource token of Processor 1 in its home-place ($m_{Processor_1} = 1$). The conflict among $C_{1.1}$ and $C_{1.2}$ is resolved as in ordinary Petri nets by nondeterministic selection of one transition (process) to fire (execute). Assume $C_{1.1}$ being chosen to execute first, then, after replacing the resource used in its home-place, only $C_{1.2}$ is enabled and gets fired. After that control flow in the P-net forces the communication SR_{12} to happen ($m_{BSR_1} = 1$, $m_{BSR_2} = 1$ and $m_{Processor_1} = 1$), etc. (We neglect the overhead of scheduling for simplicity.)

To conclude, a transition (process) assigned to some resource is enabled (ready to get active) if both, all its inputplaces in the P-net bear a (flow-) token, and the required resource token is in its home place. The transition (process) fires (executes) by removing all the flow tokens from the inputplaces in the P-net and the resource token from the R-net, making the resource unavailable for other processes. After a certain firing period flow tokens are placed to outputplaces in the P-net, while the resource token is placed back in its home place (R-net), making the resource available again.

2.2.1 Timing in PRM-nets

To support independent modeling of programs and resources we introduce the notion of *interactive timing* in the evaluation of PRM-nets. At the time a parallel program is being developed the configuration of the target hardware is generally not known. The number, type and arrangement of processing elements for example is often determined on the basis of the parallel program so as to achieve optimum performance. To this end a model of the program has to reveal the amount and services of resources required, independently of an actual resource constellation. The P-net provides all these informations: the amount of resources required is implicitely specified by the number of processes and the resource types required by them. The services and amount of resource usage are explicitly in the model by the resource requirements associated to process transitions. The resource requirements are expressed in terms of multisets of primitive processes during the parametrization of the P-net, and aggregated in the case of process compostions according to the rules given above.

The actual execution time of some process can only be determined in terms of performance characteristics of an assigned resource; these characteristics are explicitly in the R-net model in the shape of timing functions for primitive processes, the assignment information is explicitly in the PRM-net model. Given now a process transition t_i with resource requirements $\varrho_i = (\sigma, \omega)$, where $\sigma = \sum_{k|\pi_k \in \Pi_i} n_k \cdot \pi_k$ is a multiset of primitive processes out of $\Pi_i \subseteq \Pi$ (n_k denotes the multiplicity of π_k in Π_j), assigned to a resource ρ_j with services $\Pi_j \subseteq \Pi$ and service times $\tau_j(\pi)$, $\pi \in \Pi_j$ ($\Pi_i \subseteq \Pi_j$). The (deterministic) firing time for that transition is *interactively* (at the firing instant) calculated as

$$\sum_{k|\pi_k \in \Pi_i} n_k \cdot \tau_j(\pi_k).$$

The compound process $C_{1.1}$ in *Fig. 6(b)* with resource requirements as in *Fig. 1 (b)*, assigned to resource Processor 1 with execution times $\tau_1(\pi_1)$, $\tau_1(\pi_2)$ and $\tau_1(\pi_3)$ for the primitive processes π_1, π_2 and π_3 would take $2 \cdot \tau_1(\pi_1) + 2 \cdot \tau_1(\pi_2) + 4 \cdot \tau_1(\pi_3)$ time steps to execute.

2.2.2 Evaluation of PRM-nets

Overall execution time of the parallel program is evaluated by means of deterministic simulation of the PRM-net model. Existing tools (CHIOLA, 1987) have proven useful for this task, although preprocessing is necessary to simulate interactive timing (a coloured timed Petri net simulator will help for evaluating PRM-nets in the future). With the simulation of the PRM-net one simultaneously observes token distributions of home-places in the R-net, representing the basis for the derivation of figures describing resource usage.

3 Example: Systolic Matrix Multiplication

The common principle to *pipelined* parallel algorithms is that data is flowing through a cascade of *processing cells* (pipeline stages) being modified by process activities. The major characteristic of *systolic computation* in contrast to *pipelining* in general is the *homogeneity* of processing cells (LI and JAYAKUMAR, 1986), (NELSON, 1987) i.e. the set of operations applied to the incoming data stream(s) is the same in each cell. All cells operate synchronously in parallel (in that every cell causes (approximately) the same processing time) in *compute-communicate cycles.* In the compute phase all the cells are busy operating, while in the communicate phase cells propagate and receive data to and from direct neighbouring cells (GANNON, et al 1985.) ('locality of communication'). In the following section a PRM-net for a skeleton of a systolic computation is developed and illustrated in terms of a very simple example: matrix multiplication.

3.1 PRM-net Module for Systolic Computations

A simple matrix multiplication algorithm can be expressed in a systolic fashion, as is illustrated in Fig. 7 for the problem instance $C = B \cdot A$, A and B being 2×2 matrices: columns of A and rows of B are flowing through a grid of inner product cells in a west-to-east and north-to-south lock-step manner.

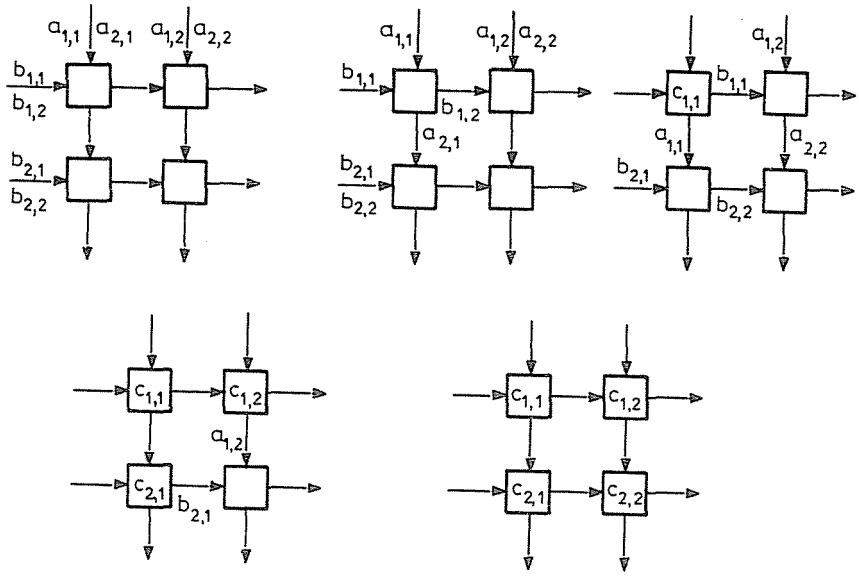


Fig. 7. Systolic matrix multiplication

The building block of a systolic computation is (as already mentioned) a *cell*, represented by a process with input ports for accepting data from previous cells, a process body comprising a set of operations to be applied to the input stream and output ports to pass (possibly modified) data on to succeeding cells. *Fig. 8* (*a*) shows a cell (concurrently) accepting two input streams a and b, applying the functionality $c := c + a \cdot b$ and outputting the streams a and b. The corresponding P-net representation for a cell performing the receive-compute-send cycle N times is given in *Fig. 8*(*b*). The process first performs two parallel receive (?) operations from north and west, then executes a multiset of primitive processes and finally sends (!) data to the east and south concurrently.

In *Fig. 9a* PRM-net model for an $N \times N$ systolic *grid* of processes of the kind of *Fig. 8* (*b*) is given. (For brevity we do not draw the 'boarder processes' at the top and the left hand side of the grid acting as data stream sources. Also not drawn are the data stream sink processes to the left and at the bottom of the grid.)

The process grid is assigned to a grid of PE's on the R-net level, while synchronisation transitions between cells are mapped to hardware links.

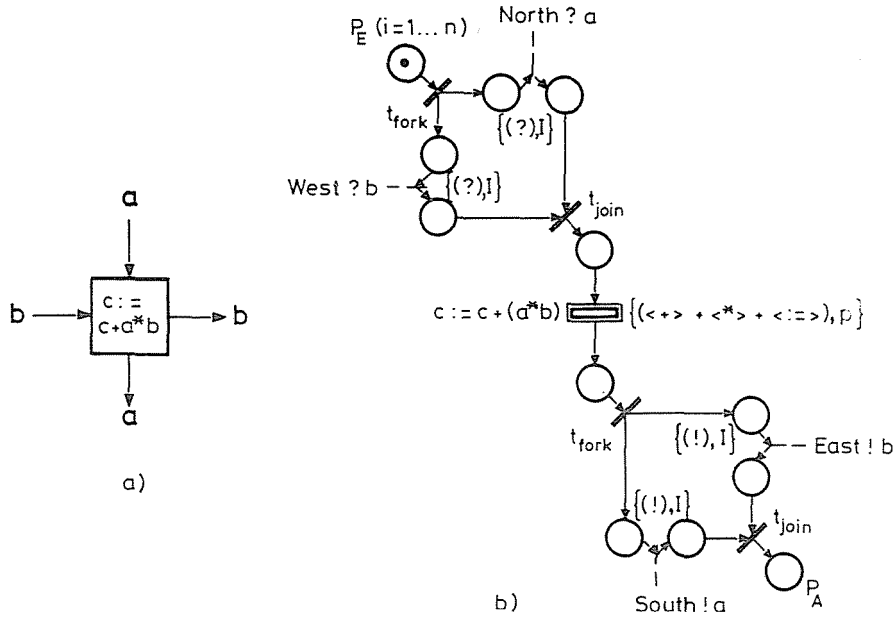


Fig. 8. Systolic cell (a), P-net representation (b)

The home place for the PE's is modeled by the dynamic relation Ppq and initially marked with all PE's $\langle i, j\rangle$, the home place for the link elements is modeled by the dynamic relation $Labcd$ (drawn twice in the figure, on the left for west-east links, on the right for north-south links). PE's $\langle i, j\rangle$ are used to execute cells $\langle x, y\rangle$ in iteration z. (compute is concurrently enabled in several modes of z, as the transition selector $(x = i) \wedge (y = j)$ binds PE's to cells independently of z. The concurrent enabling in compute expresses the (temporal) degree of parallelism in the systolic array!) A link is made available for west-east communication if neighbouring processes $\langle r, s, t\rangle$ and $\langle u, v, w\rangle$ are ready for communication at the same time (P-net) and the assigned PE's $\langle a, b\rangle$ $(a = r, b = s)$ and $\langle c, d\rangle$ $(c = u, d = v)$ are in (removable from) their home places (R-net), etc.

3.2 Performance Considerations and Optimal Placement

The systolic algorithm shown with the simple example in *Fig. 7* can generally be considered for the problem $C = A \cdot B$, given that A has dimension $n_1 \times n_2$, B has dimension $n_2 \times n_3$, and processing elements are arranged in a $N \times M$ grid, if:

(*i*) $\exists l_A \mid n_1 \mathrm{div} N = l_A$, i.e. A can be partitioned into N equal sized row blocks.

(*ii*) $\exists l_B \mid n_3 \mathrm{div} M = l_B$, i.e. B can be partitioned into M equal sized column blocks.

(*iii*) $\exists s \mid n_2 \mathrm{div} s = l$, with $1 \leq l \leq n_2$ and $l \in I\!N$ is a partition of columns of A (rows of B) into s iterations such that

(*iv*)

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n_2} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n_2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_1,1} & a_{n_1,2} & \cdots & a_{n_1,n_2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,s} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N,1} & A_{N,2} & \cdots & A_{N,s} \end{bmatrix},$$

where $A_{i,j}$ is a $l_A \times l$ submatrix of A with elements

$$A_{i,j} = \begin{bmatrix} a_{l_A(i-1)+1,l(j-1)+1} & a_{l_A(i-1)+1,l(j-1)+2} & \cdots & a_{l_A(i-1)+1,lj} \\ a_{l_A(i-1)+2,l(j-1)+1} & a_{l_A(i-1)+2,l(j-1)+2} & \cdots & a_{l_A(i-1)+2,lj} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l_A i,l(j-1)+1} & a_{l_A i,l(j-1)+2} & \cdots & a_{l_A i,lj} \end{bmatrix},$$

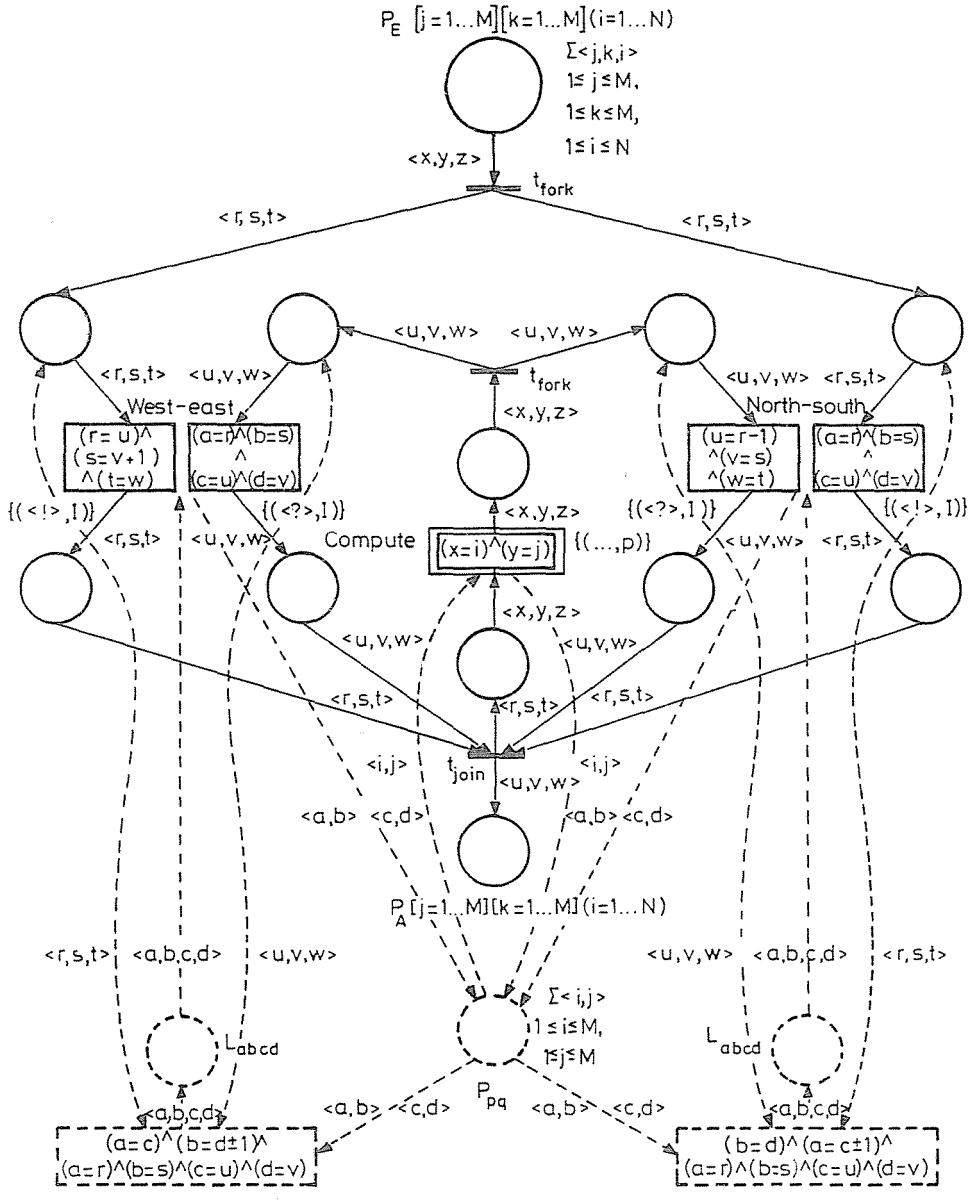
and B is analogously partitioned into sM submatrices of dimension $l \times l_B$.

Every systolic cell performs multiplication of $l_A \times l$ submatrices of A and $l \times l_B$ submatrices of B *locally* according to

$$c_{i,j} = c_{i,j} + \sum_{h=1}^{l} a_{i,h} \cdot b_{h,j} \qquad \forall i,j \mid (1 \leq i \leq l_A),\ (1 \leq j \leq l_B),$$

thus yielding $l_A \times l_B$ submatrices $C_{i,j}$ of the solution C in every cell after s iterations.

To evaluate the minimum execution time (and possible speedup) for matrix multiplication applying this algorithm we have to consider two data streams (pipelines). Let the firing time of a north-south synchronisation


P_E [j=1...M][k=1...M](i=1...N)
Σ<j,k,i>
1≤j≤M,
1≤k≤M,
1≤i≤N
<x,y,z>
t_fork
<r,s,t>
<r,s,t>
<u,v,w>
<u,v,w>
t_fork
<r,s,t>
<u,v,w>
West-east
<x,y,z>
North-south
(r=u)^(s=v+1)^(t=w)
(a=r)^(b=s)^(c=u)^(d=v)
(u=r-1)^(v=s)^(w=t)
(a=r)^(b=s)^(c=u)^(d=v)
{(<!>,1)}
{(<?>,1)}
<x,y,z>
{(<?>,1)}
{(<!>,1)}
<r,s,t>
<u,v,w>
Compute
(x=i)^(y=j)
{(...,p)}
<u,v,w>
<r,s,t>
<x,y,z>
<u,v,w>
<u,v,w>
<r,s,t>
<r,s,t>
<r,s,t>
<i,j>
t_join
<u,v,w>
<i,j>
<a,b>
<c,d>
<a,b>
<c,d>
P_A[j=1...M][k=1...M](i=1...N)
<r,s,t>
<a,b,c,d>
<u,v,w>
<u,v,w>
<a,b,c,d>
<r,s,t>
Σ<i,j>
1≤i≤M,
1≤j≤M
L_abcd
L_abcd
<a,b,c,d>
<a,b>
<c,d>
P_pq
<a,b>
<c,d>
<a,b,c,d>
(a=c)^(b=d±1)^(a=r)^(b=s)^(c=u)^(d=v)
(b=d)^(a=c±1)^(a=r)^(b=s)^(c=u)^(d=v)

transition be $\zeta^{\text{n\_s}}$ and $\zeta^{\text{w\_e}}$ for a west-east transition, respectively. The firing time of a process cell transition performing multiplication of submatrices is denoted by θ[3]. Then under the assumption of concurrent send and receive operations the execution time in a cell is $\theta + 2 \cdot \max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}})$ because of inhomogeneous submatrix transfer times. The *fill-time* of the first (leftmost) n_s pipeline (i.e. the time the first submatrix of A uses to reach the last cell in the array) with N stages (cells) is

$$t^{\text{n\_s}}_{fill_N}(1) = \max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}}) + N \cdot (\theta + \zeta^{\text{n\_s}}),$$

as computation in the first cell can start at time $\max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}})$ at the earliest. The second north-to-south pipeline can start computation at $\max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}}) + (\theta + \zeta^{\text{w\_e}})$, and the *fill-time* of pipeline i is hence

$$t^{\text{n\_s}}_{fill_N}(i) = \max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}}) + (i-1) \cdot (\theta + \zeta^{\text{w\_e}}) + N \cdot (\theta + \zeta^{\text{n\_s}}).$$

Analogously the *fill-time* for the i-th M-stage w-e pipeline is

$$t^{\text{w\_e}}_{fill_M}(i) = \max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}}) + (i-1) \cdot (\theta + \zeta^{\text{n\_s}}) + M \cdot (\theta + \zeta^{\text{w\_e}}).$$

After filling both groups of pipelines, submatrices are released in time intervals of

$$t^{\text{n\_s}}_{rel_N}(i) = t^{\text{w\_e}}_{rel_M}(i) = \theta + 2 \cdot \max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}}).$$

As s submatrices have to be propagated through every pipeline, the execution time is

$$t^{\text{n\_s}}_{exec_N}(i,s) = t^{\text{n\_s}}_{fill_N}(i) + (s-1) \cdot t^{\text{n\_s}}_{rel_N}(i) =$$

$$\max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}}) + (i-1) \cdot (\theta + \zeta^{\text{w\_e}}) + N \cdot (\theta + \zeta^{\text{n\_s}}) + (s-1) \cdot (\theta + 2 \cdot \max(\zeta^{\text{n\_s}}, \zeta^{\text{w\_e}}))$$

[3] Given n_1, n_2, n_3 and a partition l_A, l_B and s. Let $\widehat{c}$ be the computation time for one inner product then

$$\theta = \frac{n_1}{N} \cdot \frac{n_3}{M} \cdot \frac{n_2}{s} \cdot \widehat{c} = l_A \cdot l_B \cdot l \cdot \widehat{c},$$

and

$$\zeta^{\text{w-e}} = \widehat{\sigma} + \frac{n_1}{N} \cdot \frac{n_2}{s} \cdot \widehat{\tau} = \widehat{\sigma} + l_A \cdot l \cdot \widehat{\tau},$$

$$\zeta^{\text{n-s}} = \widehat{\sigma} + \frac{n_2}{s} \cdot \frac{n_3}{M} \cdot \widehat{\tau} = \widehat{\sigma} + l \cdot l_B \cdot \widehat{\tau},$$

respectively ($\widehat{\sigma}$ denotes the data transfer setup time and $\widehat{\tau}$ is the propagation time for a single number).

and

$$t^{\mathtt{w\_e}}_{exec_M}(i,s) = t^{\mathtt{w\_e}}_{fill_M}(i) + (s-1)\cdot t^{\mathtt{w\_e}}_{rel_M}(i) =$$

$$\max(\zeta^{\mathtt{n\_s}}, \zeta^{\mathtt{w\_e}}) + (i-1)\cdot(\theta + \zeta^{\mathtt{n\_s}}) + M\cdot(\theta + \zeta^{\mathtt{w\_e}}) + (s-1)\cdot$$

$$(\theta + 2\cdot\max(\zeta^{\mathtt{n\_s}}, \zeta^{\mathtt{w\_e}}))$$

respectively. The execution time $T_{N\times M}$ of a systolic computation on a $N \times M$ grid with a data stream length of s is therefore

$$T_{N\times M}(s) = \max(t^{\mathtt{n\_s}}_{exec_N}(M,s), t^{\mathtt{w\_e}}_{exec_M}(N,s)).$$

Example To compare analytical performance prediciton with PRM-net simulations we define a small example resulting from an experiment investigated on multitransputer hardware (T414, 17MHz) with $\widehat{c} = 3.001$ μsec, $\widehat{\sigma} = 2.903$ μsec and $\widehat{\tau} = 3.999$ μsec for transmitting a single integer.

Assume A to be 6×8, B be 8×12 and the availability of 2×3 ($N = 2$, $M = 3$) PE's to calculate $C = A \cdot B$. Choosing $s = 4$ implies A to be partitioned into $2 \cdot 4 = 8$ submatrices with dimension $\frac{6}{2} \times \frac{8}{4}$, whereas B is partitioned into $4 \cdot 3 = 12$ submatrices with dimension $\frac{8}{4} \times \frac{12}{3}$. In every cell submatrices of A and B are multiplied to $\frac{6}{2} \times \frac{12}{3}$ submatrices of C, which are cumulated in $s = 4$ iterations.

The resource requirements for the synchronisation transitions $s_{\mathtt{w\_e}}$ are $\varrho = \{(\langle![6]\rangle + \langle?[6]\rangle, \mathbf{l})\}$, as in one single communication step a submatrix of A ($\frac{6}{2} \cdot \frac{8}{4} = 6$ integers) has to be transmitted, taking $6 \cdot 3.999 + 2.903 = 26.897$ μsec if assigned to a resource of type $\mathbf{l}$. Transition $s_{\mathtt{n\_s}}$ requires $\varrho = \{(\langle![8]\rangle + \langle?[8]\rangle, \mathbf{l})\}$ ($\frac{8}{4} \times \frac{12}{3} = 8$ integers) and takes $8 \cdot 3.999 + 2.903 = 34.895$ μsec to fire. The computation transition requires $\varrho = \{(24(\langle*\rangle + \langle+\rangle + \langle:=\rangle), \mathbf{p})\}$ for $\frac{6}{2} \cdot \frac{12}{3} \cdot 2 = 24$ inner products yielding in a firing delay of $24 \cdot 3.001 = 72.024$ μsec if assigned to a resource of type $\mathbf{p}$. The overall execution time $T_{2\times3}(4)$ is because of $\theta = 72.024$ μsec and $\max(\zeta^{\mathtt{n\_s}}, \zeta^{\mathtt{w\_e}}) = \max(34.895\ \mu\text{sec}, 26.897\ \mu\text{sec})$:

$$T_{2\times3}(4) = \max(34.895 + (3-1)\cdot 98.921 + 2\cdot 106.919 + (4-1)\cdot 141.814,$$

$$34.895 + (2-1)\cdot 106.919 + 3\cdot 98.921 + (4-1)\cdot 141.814) =$$

$$\max(872.017, 864.019) = 872.017\ \mu\text{sec}.$$

A		B		Topology	Size Subm.		IP	Exec. Time	Proc.	Speedup
n_1	n_2	n_2	n_3	$N \times M$	s	n_s	w_e	i	$T^{PRM}_{N\times M}(s)$	P	S_P
6	8	8	12	2×3	1	32	24	96	1742.755	6	1.436
6	8	8	12	2×3	2	16	12	48	1156.457	6	2.164
6	8	8	12	2×3	4	8	6	24	872.017	6	2.869
6	8	8	12	2×3	8	4	3	12	747.215	6	3.349
6	8	8	12	3×2	1	48	16	96	1998.691	6	1.252
6	8	8	12	3×2	2	24	8	48	1348.409	6	1.856
6	8	8	12	3×2	4	12	4	24	1031.977	6	2.425
6	8	8	12	3×2	8	6	2	12	891.179	6	2.808
6	8	8	12	6×1	1	96	8	96	4436.225	6	0.564
6	8	8	12	6×1	2	48	4	48	2762.031	6	0.906
6	8	8	12	6×1	4	24	2	24	1933.643	6	1.294
6	8	8	12	6×1	8	12	1	12	1536.867	6	1.628
6	8	8	12	1×6	1	16	48	96	3092.561	6	0.809
6	8	8	12	1×6	2	8	24	48	1898.247	6	1.318
6	8	8	12	1×6	4	4	12	24	1309.799	6	1.910
6	8	8	12	1×6	8	2	6	12	1032.993	6	2.422

Fig. 10: Speedup for Matrix Multiplication using 6 PE's

Simulating the PRM-net in *Fig. 9* revealed the following execution times:
$T^{PRM}_{2\times3}(4) = 872.017029$ μsec, $T^{PRM}_{2\times3}(3) = 730.203003$ μsec, $T^{PRM}_{2\times3}(2) = 588.389038$ μsec, $T^{PRM}_{2\times3}(1) = 446.575012$ μsec, which are identical to analytical results. Moreover PRM-net simulations (see *Fig. 10*) showed that there is an even better partitioning into submatrices for 2×3 PE's (i.e. $s = 8$), and discovered that one cannot gain higher speedup when rearranging the systolic array. In fact an inappropriate arrangement (e.g. 6×1) of PE's can make parallel matrix multiplication even slower than in the sequential case using a single PE.

4 Conclusion and Perspectives

A Petri net based model for multiprocessor hardware executing a parallel program has been developed, integrating all the performance influencing factors (performance characteristics of hardware resources, structure and resource requirements of parallel programs and the process-to-processor mapping). The model allows investigations into structural properties of the parallel program by applying methods known from the Petri net theory (e.g. the invariant method). Performance is evaluated by deterministic or stochastic simulation of the derived timed Petri net model as well as

analytically. In any case existing computerized tools are used (CHIOLA, 1987) for automated analysis.

Synthesis and decomposition rules for hierarchical (parallel) program net models along with the specification power of Coloured Petri nets (JENSEN, 1987), will be used further for the performance oriented, graphical development of parallel programs. The clear Petri net formalism as well as the general methods developed so far in theory give rise for an integration into a set of tools assisting the development process of parallel software in all phases of the software life cycle. The architecture of the toolset will comprise graphical editing facilities using a Petri net metaphor, tools for automatic generation of the performance model from high level program specifications along with its evaluation and result interpretation for performance prediction, tools for automated mapping support, program code generation, measurement and monitoring facilities and a set of tools for visualising multiprocessor performance and runtime behaviour of algorithms.

References

CHIMENTO, P. F. – TRIVEDI, K. S. (1988): The Performance of Block Structured Programs on Processors Subject to Failure and Repair. In: E. Gelenbe, Ed., *High Performance Computer Systems,* pp. 269–280, North-Holland, Amsterdam.

CHIOLA, G. (1987): GreatSPN Users Manual. Version 1.3, September 1987. Tech. Rep., Dipartimento di Informatica, corso Svizzera 185, 10149 Torino, Italy.

DUGAN, J. B. – TRIVEDI, K. S. – GEIST, R. M. – NICOLA, V. F. (1984): Extended Stochastic Petri Nets: Applications and Analysis. In: *Proc. of the 10th Int. Symp. on Computer Performance (Performance 84), Paris, France, Dec 19-21, 1984,* pp. 507–519.

FERSCHA, A. (1990): *Modellierung und Leistungsanalyse Paralleler Systeme mit dem PRM-Netz Modell.* PhD Thesis, University of Vienna, Institute of Statistics and Computer Science, May.

GANNON, D. – BECHTOLSHEIM, S. et al. (1985 Apr): The Systolic BLAS: An Experiment in Parallel Algorithm Design. In: *Proc. of the 30th IEEE COMPCON'85 Int. Spring Conf., pp. 66–70, Apr.*

GELENBE, E. – MONTAGNE, E. – SUROS, R. (1988): A Performance Model of Block Structured Parallel Programs. In: M. Cosnard, P. Quinton, Y. Robert, and M. Tchuente, Eds., *Parallel Algorithms and Architectures*, pp. 127–138, North-Holland, Amsterdam.

GENRICH, H. J. (1987): Predicate/Transition Nets. In: W. Brauer, W. Reisig, and G. Rozenberg, Eds., *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986. LNCS* Vol. 254, pp. 207–247, Springer Verlag.

GENRICH, H. (1988 a): Equivalence Transformations of PrT nets. In: *Proc. of the 9th European Workshop on Applications and Theory of Petri Nets, June 22 - 24, 1988, Venice, Italy. (to appear),* pp. 229–248.

HOARE, C. A. R. (1978 Aug): Communicating Sequential Processes. *Communications ACM,* Vol. 21, No. 8, Aug.

Jensen, . (1987 a): Coloured Petri Nets. In: W. Brauer, W. Reisig, and G. Rozenberg, Eds., *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986. LNCS* Vol. 254, pp. 248–299, Springer Verlag.

Lazowska, E. D. – ahoran, J. – Graham, S. G. – Sevcik, . (1984): Quantitative System Performance. Computer System Analysis using Queueing Network Models. Prentice-Hall, Englewood Cliffs, New Jersey.

Li, H. . – Jayakumar, R.) (1986 Sep): Systolic Structures: A Notion and Characterization. *Journal of Parallel and Distributed Computing,* Vol. 3, No. 3, pp. 373–397, 1986.

Murata, T. (1989 Apr): Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE,* Vol. 77, No. 4, pp. 541–580, Apr.

Nelson, P. A. – Snyder, P. (1987): Programming Paradigms for Nonshared Memory Parallel Computers. In: L. H. Jamieson, D. B. Gannon, and R. J. Douglass, Eds., *The Characteristics of Parallel Algorithms,* pp. 3–20, MIT Press, 1987.

Reisig, W. (1985 a): Petrinetze. Eine Einführung. *Studienreihe Informatik,* Springer-Verlag, Berlin, 2nd Ed.

Reisig, W. (1987 a): Place/Transition Systems. In: W. Brauer, W. Reisig, and G. Rozenberg, Eds., *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986. LNCS* Vol. 254, pp. 117–141, Springer Verlag.

Trivedi, . S. (1984): Probability and Statistics with Reliability, Queueing and Computer Science Applications. Prentice-Hall, Englewood Cliffs, New Jersey.

Vernon, M. . – Holliday, M. A. (1985 Jul): A Generalized Timed Petri Net Model for Performance Analysis. In: *Proc. Int. Workshop on Timed Petri Nets,* pp. 181–190, IEEE Comp. Soc. Press, July. 1985.

Vernon, M. . – Holliday, M. A. (1987 Jan): Exact Performance Estimates for Multiprocessor Memory and Bus Interference. *IEEE Transactions on Computers,* Vol. C-36, No. 1, pp. 76–85, Jan. 1987.

Address:

A. FERSCHA, – G. HARING
Institut für Statistik und Informatik, Universität Wien
Lenaugasse 2/8, A-1080 Wien
Austria