GSPN Models of Concurrent Architectures with Mesh Topology*

Stefano Caselli, Gianni Conte

Dipartimento di Ingegneria dell'Informazione
Università di Parma
Viale delle Scienze, 43100 Parma - Italy

Abstract

The paper describes a GSPN-based approach to performance evaluation of parallel MIMD architectures with mesh interconnection schemes. The analysis considers regular computational structures in which data exchange and inter-process synchronization require communication only among processors that are physically connected. The effectiveness of the modeling technique for this class of architectures is shown through examples considering both the effect of the interaction policies among processes and the benefit of multitasking on the global efficiency of the architecture.

1 Introduction

The recent advances in VLSI technologies are drawing an increasing attention to Multiple–Instruction stream, Multiple–Data stream (MIMD) massively parallel architectures, enabled as viable and practical computer architectures. A large number of applications has been experimented in many different fields, from real–time process control to signal processing and discrete–event simulation. MIMD architectures can be classified according to the computational model they directly support: the global memory model or the message–passing model. Particular emphasis is currently being placed on message–passing architectures, for the enhanced reliability and fault–tolerance attainable, as well as their compatibility with VLSI technologies, high modularity and scalability [2, 5, 7, 19, 20]. In this class of architectures no shared memory space exists, and cooperation among processes is achieved by sending and receiving messages only. Nevertheless, price/performance ratios better than on conventional systems can be obtained, provided that a good exploitation of the architectural peculiarities is realized by the workload distribution and the underlying programming model.

Design methodologies able to cope with massively parallel architectures, however, are not well established, yet. One of the major challenges faced by designers deals with matching the workload profile of an application with the hardware parallel architecture in order to maximize performance and resources utilization, taking into account both the characteristics of the processing elements and the interconnection topology. At the current edge of technology, such a matching problem may arise from the opposite perspective as well, namely in the development of application–driven architectures, optimized for a given workload. An efficient mapping of processes to processors is especially crucial in message–passing architectures, in which synchronization among processes typically entails blocking semantics (e.g., an extended rendez–vous). Low overall throughput and deadlock are among the potential effects of an unproperly–chosen workload distribution in a message–passing architecture.

Several techniques are currently being investigated to achieve automatic mapping of processes to processors, including static techniques, as compile–time creation of a task graph where the interactions among processes are weighted [6], and dynamic load balancing by means of process migration. These approaches aim to extract, either statically or dynamically, the process structure embedded in a given workload in order to efficiently map it onto the available processor network. Since this mapping problem is known to be NP–hard, it appears also useful to investigate *a priori* general qualitative and quantitative properties exhibited by abstract computational structures. In this paper we discuss the application of Petri net-based models to performance analysis of concurrent computer architectures. In particular, we develop GSPN models for meshes of identical processing elements, with uniform workload and process structure, and communication requirements that can be satisfied by the processors belonging to the immediate neighborhood, such those involved in convolution–type or systolic algorithms. Features explicitly modeled include the interaction policy among processes and the level of multitasking within each processing element. Availability of GreatSPN [12], a well–assessed tool for the analysis of GSPN models, made this work possible.

The rest of the paper is organized as follows. After a brief overview of the GSPN approach to the modeling of distributed architectures, GSPN models of message–passing concurrent architectures are developed. First, design features are illustrated by developing partial models describing the behavior of the single processor. Next, these models are used as building blocks to generate complete models for meshes of various size, which are then analyzed to assess their properties and overall performance.

*This work was partially supported by C.N.R., Italy, under contracts 8900246.12 and 9004095.12.

2 GSPN Modeling of Distributed Architectures

Any performance analysis task requires a preliminary decision – taken on the basis of the objective of the analysis – about the level of detail at which the system has to be seen, and, in turn, about the most appropriate modeling tool. In many cases the complex phenomena that take place in the system being modeled can be synthetically and effectively described by means of probabilistic assumptions. In these cases several advantages may be achieved: first, not all the details of the system behavior need to be known to perform the analysis, so that it may be applied even in the early design stages; second, performance indices can be obtained using analytical or numerical techniques instead of simulation; third, architectural decisions are taken with reference to classes of applications rather than specific cases. Such an approach might be called *conceptual* or *high-level* modeling. Petri net models tend to be more appropriate at this level.

Petri net's ability to precisely and unambiguously describe concurrent operations makes this model a natural choice for analysis of distributed systems. When dealing with concurrent computer architectures, Petri net models offer an additional peculiar advantage: they allow a uniform and consistent representation to be used for both hardware (e.g., number of processors, interconnection topology) and software (e.g., level of multiprocessing, process interaction policies) features, thus leading to an easier modeling of the interactions between the two levels. This capability is essential in the case of MIMD architectures, since complex interactions can occur when a multitasking approach must coexist with the physical distribution of the resources.

From a methodological viewpoint, Petri nets offer the possibility of using the same paradigm in several phases of system design, namely, specification, verification, performance analysis, and automatic implementation. This fact has the advantage of integrating the performance evaluation step into the design process, as is not always the case at present.

The use of Petri net-based techniques for performance evaluation implies the addition of the notion of time to the original Petri net model, where no time concept is defined. In this work the Generalized Stochastic Petri Net (GSPN) [3] approach is followed. GSPN have been adopted as a modeling technique in many different application fields [2, 4, 8, 11, 13, 15].

2.1 Definition of GSPN

GSPN structural analysis is based on a revised definition of the original model in [3] that simplifies model development and reduces the complexity of the solution computation. This revised definition, fully reported in [1, 17], is here only briefly summarized.

GSPN are obtained by introducing a temporal specification in the class of Petri nets augmented with inhibitor arcs and priorities. Timing is associated with transitions, and specifies the amount of time the transition must remain enabled before firing.

Two classes of temporal specifications are possible: delays are exponentially distributed random variables for transitions with priority zero, that are consequently called *timed*, and are deterministically zero for transitions with priority $n \geq 1$; these transitions are called *n-immediate*. When only two priority levels exist, transitions are simply referred to as timed and immediate. Markings that enable timed transitions only are said to be *tangible*, whereas markings that enable n-immediate transitions are said to be *vanishing*.

Formally, a GSPN is an 8-tuple:

$$\text{GSPN} = (P, T, \Pi(.), I(.), O(.), H(.), W(.), M_0) , \quad (1)$$

where $\text{PN} = (P, T, \Pi(.), I(.), O(.), H(.), M_0)$ is the marked PN underlying the GSPN, and constitutes the structural component of a GSPN model, whereas $W(.)$ is a function defined on the set of transitions allowing the definition of the stochastic component of a GSPN model. In particular, it maps transitions into real positive numbers. We use the simpler notation w_k to indicate $W(t_k)$, for any transition $t_k \in T$. The quantity w_k is called the "rate" of transition t_k if t_k is timed, and the "weight" of transition t_k if t_k is n-immediate.

When a vanishing marking is entered, the weights of the enabled immediate transitions are used to probabilistically select the (immediate) transition(s) to fire. The time spent in any vanishing marking is deterministically equal to zero.

Due to the memoryless property of the exponential distribution of firing delays, it has been shown that the reachability graph of a bounded SPN is isomorphic to a finite Markov Chain (MC) [18]; the same has been shown true in the case of GSPN [3].

We address to other papers [1, 17] for a complete and formal description of the GSPN properties and characteristics. Rather, we remark here the typical steps followed in the GSPN analysis of a system:

- model development (both top–down and bottom–up techniques are applicable);
- model validation (by means of interactive simulation and structural analysis, including formal proving of expected behavioral properties of the model);
- assessment of qualitative properties (also based on the structural analysis of the Petri net);
- definition of performance indices of interest in terms of GSPN markings and transition firings;
- reachability set and reachability graph computation (a preliminary step to obtain the continuous–time Markov chain associated with the GSPN model; deserves special consideration since it is the most (computationally) expensive step and determines feasibility of the analysis);
- Markov chain solution;
- performance indices computation (from the Markov chain solution).

GreatSPN provides integral support to all the steps in the outlined analysis procedure.

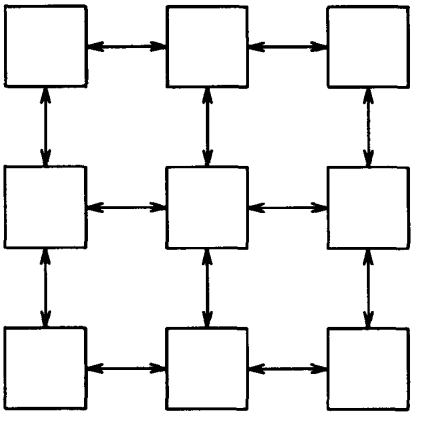

Figure 1: Regular mesh of processing elements.

3 Analysis of concurrent architectures

We consider MIMD architectures that use regular interconnection patterns and distributed memory schemes [19]. The advantage and the success of this class of architectures are related to the availability of VLSI subsystems that can be used as building blocks in powerful massively parallel structures. In particular, different kinds of Transputer-based architectures are now available as commercial products [16]. These architectures support the CSP (Communicating Sequential Processes) concurrent model of computation [14], from which the Occam language was directly derived.

The possibility of modeling Occam-based systems using GSPN was already proved in [2], where a parallel architecture was analyzed consisting of 16 Transputers communicating by means of a multistage Delta interconnection network. Our goal now is to investigate the effectiveness of the same approach in the case of regular meshes of processing elements, taking into account the effects of the workload distribution, the communication patterns, and the synchronization schemes available. As a side goal, we also aim to verify the efficiency of the technique against the dimension of the problem, and the capacity of affording both the quantitative problems (bottleneck analysis) and the logical ones (deadlock analysis).

In Figure 1 a 3x3 regular mesh of processing elements is shown in which each processor is directly connected to its local neighbours. GSPN models for this class of architectures are developed under the assumptions that the workload is evenly distributed across all the processing elements and the computations assigned to each processor entail interaction with the local neighbours only (thus, we neglect any routing problem).

3.1 Intra–processor concurrency models

In this section we develop basic models of the behavior of a single processing unit within a concurrent architecture. Models for multiprocessor architectures will be obtained combining, according to the specific

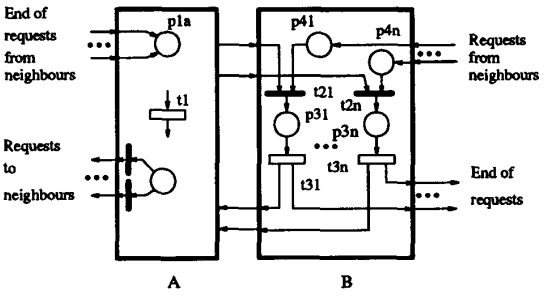

Figure 2: Basic model of the processing elements.

interconnection topology, the building blocks here described.

The computation assigned to each processor is modeled, in general, as a set of concurrent tasks performing a cyclic activity. This activity includes a local operation phase, whose duration is a stochastic variable, exponentially distributed with parameter λ, followed by a request for service addressed with equal probability to one of the neighbours. Service time is a stochastic variable too, exponentially–distributed with parameter μ. Thus, each processor must provide service to requests coming from any of its immediate neighbours. In these models we do not explicitly represent communication activities: the time required to perform sending and receiving operations is considered part of the service requesting and service giving tasks.

Figure 2 shows the basic structure of the GSPN model of each processing unit. It is composed of two different subnets. The first one, denoted with A, describes the local activity of the task (modeled by the timed transition t_1 with rate λ) along with the issuing of the service requests towards the neighbours. The second one, denoted with B, models the service provided by the processor to the requests from the neighbours.

In subnet B, for any direct neighbour the model must include a place to describe the pending requests (p_{4i} for the i-th neighbour), an immediate transition (t_{2i}) to model the granting operation, and a place (p_{3i}) followed by a timed transition (t_{3i}) with rate μ describing the service operation. Once service has been completed, a token is returned to the requesting neighbour, where the task is inserted in the ready–task pool (place p_{1a} of subnet A).

The specific process interaction and scheduling policies determine whether a local computation is resumed or further external requests (if pending) are served next. These policies are modeled by subnet A (not yet specified) and by the priority of transitions t_{2i}. In any case, the activity of each processor alternates between execution of local tasks (subnet A) and service of external requests (subnet B).

Two basic interaction policies will be considered in the following:

- *preemptive interaction*: requests coming from the neighbours are served with higher priority than

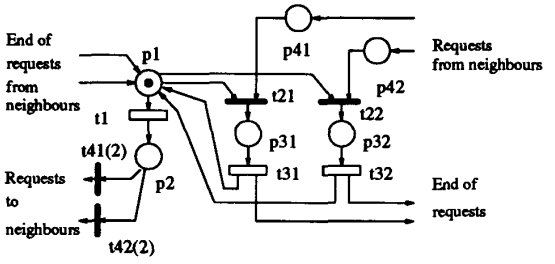

Figure 3: Simplified GSPN model of the behavior of a processor in the case of preemptive interaction and no multitasking.

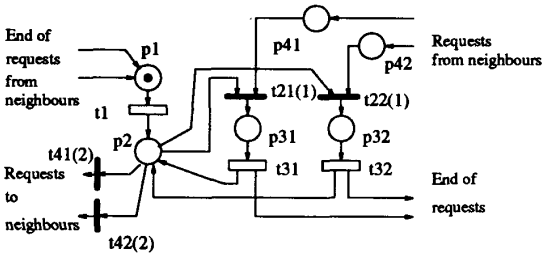

Figure 4: Simplified GSPN model of the behavior of a processor in the case of non–preemptive interaction and no multitasking.

the local activity, i.e. requests are preemptive;

- *non–preemptive interaction*: requests are not preemptive but are served, instead, at the end of the local operation.

All models in this section refer, for ease of representation, to a processor with two neighbours. We first discuss two simplified models considering only one task per processor. Figure 3 shows the GSPN model of the behavior of a processor in the case in which the requests are preemptive, whereas Figure 4 represents the case in which requests are not preemptive. In both cases the local computation is modeled by a place p_1 and a transition t_1 with rate λ, the termination of the local computation is modeled by place p_2, and the request for service towards the neighbours is modeled by the transition set t_{4i}.

Under the preemptive policy Ffigure 3), the external requests are immediately served (firing of transition t_{2i}), preempting any local computation. The local computation may resume only when no further external requests are pending.

In the non–preemptive case (Figure 4), service is given to neighbours only at the end of the local computation. In this case, at the end of the local operation priority is given to accepting and serving requests from the neighbours rather than to issuing the local request, in order to avoid deadlock (t_{2i} priority is greater than t_{4i} priority). This deadlock potential (arising if priority is reversed) can be easily recognized performing

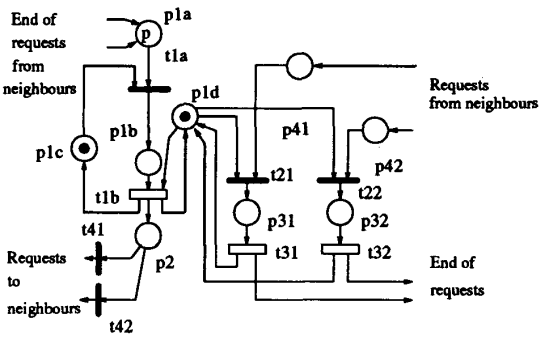

Figure 5: GSPN model of the behavior of a processor in the case of preemptive interaction and multitasking.

structural analysis on the net describing a multiprocessor configuration.

Models in Figures 3 and 4, though fairly intuitive, fail to appropriately distinguish between *processor* state and *task* state. As a consequence, when a task has completed its local execution phase and has been dispatched to one of the neighbours by the owner processor, the processor is not able to provide service to incoming requests. In actual concurrent architectures, efficiency in use of the processing power dictates the following constraints for any reasonable scheduling policy:

1. if no external request is pending and the ready–task pool is not empty, a local task is immediately scheduled for execution;
2. if the ready–task pool is empty but there are external requests pending, service of an external request is immediately scheduled for execution;
3. if the ready–task pool is not empty and there are external requests pending, either a local task or service of an external request is immediately scheduled for execution.

The more elaborate GSPN models in Figures 5 and 6 show the behavior of a processor taking into account multitasking and efficient use of the processing power. Figure 5 represents the case in which requests are preemptive, whereas Figure 6 represents the case in which requests are not preemptive. It can be seen that both these models satisfy the aforementioned constraints on the scheduling policy: when a task terminates its local activity and sends a service request to a neighbour, the processor is immediately freed to serve incoming requests or to execute another task.

In Figure 5 the local computation is modeled by place p_{1b} and transition t_{1b} with rate λ, the termination of the local computation moves a token in p_2, and the requests for service towards the neighbours are modeled by the transition set t_{4i}. The initial marking of place p_{1a} (p) expresses the number of tasks active on the processor (multitasking level). A token in place p_{1d} indicates that either the processor is executing a

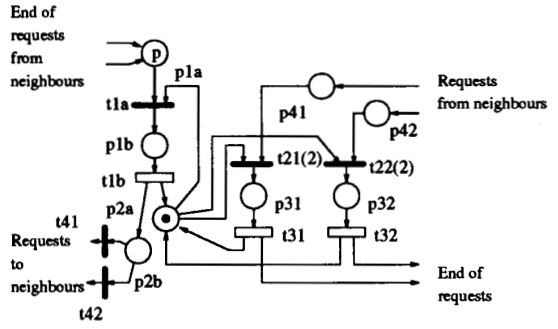

Figure 6: GSPN model of the behavior of a processor in the case of non–preemptive interaction and multitasking.

local task or is idle. Since the policy described is preemptive, in both cases an external request may be immediately served. Multiple incoming requests are not allowed to preempt each other.

The following property must be verified as a place invariant on the net describing the MIMD architecture: $M(p_{1d}) + M(p_{31}) + M(p_{32}) = 1$. This relationship implies that the processor is in one of the following states:

- executing a local task: $[M(p_{1d}) = 1] \wedge [M(p_{1b}) = 1]$,
- idle: $[M(p_{1d}) = 1] \wedge [M(p_{1c}) = 1]$,
- serving an external request: $[M(p_{31}) = 1] \vee [M(p_{32}) = 1]$.

The model of Figure 6 shows the case of non–preemptive interaction. Firing of transition t_{1b}, which as in the previous case models task execution, moves a token into both p_{2a} and p_{2b}. The token in place p_{2a} can enable transition t_{1a}, thus making active a ready task. Pending requests of service from neighbours have priority over the execution of new local tasks, therefore transitions t_{2i} have priority greater than transition t_{1a}.

The following property must be verified on the net: $M(p_{1b}) + M(p_{2a}) + M(p_{31}) + M(p_{32}) = 1$. This relationship implies that the processor is in one of the following states:

- executing a local task: $M(p_{1b}) = 1$,
- idle: $M(p_{2a}) = 1$,
- serving an external request: $[M(p_{31}) = 1] \vee [M(p_{32}) = 1]$.

3.2 Concurrency models of mesh structures

Once the behavior of each processor has been defined by a suitable submodel, the model of a complete MIMD architecture can be obtained by proper interconnection of these submodels. GSPN analysis can now be applied on the model to assess deadlock potential and other structural properties, along with performance measures as processor idle time and overall processing power. We report results for both the cases of single task and multitask activity within each processor.

Mesh size and interaction policy	# of states
2x2, preemptive	57
2x3, preemptive	833
3x3, preemptive	58681
2x2, not preemptive	117
2x3, not preemptive	2176
3x3, not preemptive	215712

Table 1: Number of tangible states for different mesh sizes and interaction policies (simplified models).

Mesh size and interaction policy	# of states
2x2, preemptive	57
2x3, preemptive	939
3x3, preemptive	81893
2x2, not preemptive	189
2x3, not preemptive	4692
3x3, not preemptive	$\gg$ 300000

Table 2: Number of tangible states for different mesh sizes and interaction policies (models in Figures 5 and 6 with p=1).

3.2.1 Models without multitasking

Meshes of size 2x2, 2x3, and 3x3 have been considered and analyzed considering both the simplified processor models and the more detailed ones with p set to 1. Structural analysis shows that, in all models, all places are covered by P-invariants and the net is bounded. Furthermore, the number of T-invariants is given by the sum over all processors of the number of neighbours, all transitions are covered by T-invariants and the net is live.

Table 1 reports the number of tangible states for the simplified models in Figures 3 and 4, whereas Table 2 reports the number of states for the models in Figures 5 and 6 with p=1. With the simpler model, reachability graph construction for the 3x3 mesh with non–preemptive interaction among processors takes about 1 hour of CPU time on a SUN4 SPARCstation 1+ equipped with 16 Mbyte RAM, thus approaching the computational limits of GreatSPN on this workstation. With the more elaborate model, the analysis of the 3x3 mesh exceeds these limits.

Tables 3 and 4 show some performance figures obtained from the quantitative analysis assuming $\lambda = 1$ and $\mu = 10$. For all these models we can distinguish the processor figures according to the number of neighbours. Given the uniform workload, within each model all processors with the same number of neighbours exhibit the same performance figures. Specifically, the idle time (time spent by the processors waiting for service) is reported in Tables 3 and 4, along with the overall effective processing power. In all mod-

Mesh size and interaction policy	processing power	idle time		
		2-neigh. pr.	3-neigh. pr.	4-neigh. pr.
2x2, preemptive	3.64(90.95%)	9.0%	-	-
2x3, preemptive	5.46(90.93%)	9.3%	8.9%	-
3x3, preemptive	8.16(90.67%)	9.6%	9.1%	9.2%
2x2, not preempt.	1.70(42.62%)	57.4%	-	-
2x3, not preempt.	2.44(40.67%)	60.1%	57.8%	-
3x3, not preempt.	3.54(39.33%)	58.6%	56.0%	55.6%

Table 3: Performance results: idle time and effective processing power (simplified models).

Mesh size and interaction policy	processing power	idle time		
		2-neigh. pr.	3-neigh. pr.	4-neigh. pr.
2x2, preemptive	3.66(91.52%)	8.5%	-	-
2x3, preemptive	5.49(91.47%)	8.7%	8.2%	-
3x3, preemptive	8.23(91.43%)	9.0%	8.2%	8.4%
2x2, not preempt.	2.66(66.39%)	33.6%	-	-
2x3, not preempt.	3.96(66.03%)	34.6%	32.7%	-

Table 4: Performance results: idle time and effective processing power.

els each processor can be in one of the following states:

- active executing its local computation,
- active serving a request from one of the neighbours,
- idle, with the local task waiting for service from a neighbour,
- idle, but with the local task being served by a neighbour.

Only the active states of the processors are considered in the effective processing power index reported in Tables 3 and 4.

The results in Tables 3 and 4 confirm that non–preemptive interaction policies achieve a low throughput if multitasking is not allowed, whereas interrupt–like policies are an effective replacement for multitasking. Thus, in this somewhat unfair comparison the preemptive interaction policy is dramatically more efficient than the non–preemptive one.

3.2.2 Multitasking models

Figure 7 shows the model of a 2x2 concurrent architecture with preemptive interaction policy when the multitasking behavior of Figure 5 is adopted on each processor. Figure 8 shows the model of an architecture with the same topology when the behavior of each processor is modeled as in Figure 6.

Tables 5 and 6 report some results that can be obtained by the GSPN analysis of these architectures with respect to the multitasking level. The throughput indicates the number of tasks that are executed by each processor in the unit time. This performance parameter is obtained (with reference to Figures 5 and 6) multiplying the rate of transition t_{1b} by the probability that $M(p_{1b}) \geq 1$. Considering the values used to derive these results ($\lambda = 1$, $\mu = 10$), in the case of preemptive interaction the asymptotic throughput ($1/1.1 = 0.90...$) is quickly reached with a multitasking level of only 3 tasks per processor. When a non–preemptive policy is used, the performance obtained at a reasonable multitasking level (4) is very close to the asymptotic value (96.7%). Therefore, multitasking allows this simpler policy to be adopted with a minimal penalty on performance.

The cycle time in Tables 5 and 6 is simply the reciprocal of the throughput and is reported for sake of clarity and to derive task latency. Latency represents the time between two successive rescheduling of a given task and is computed multiplying the cycle time by the number of tasks. Latency corresponds to the total delay that a task experiences in the following steps: execution on its local processor, waiting for service from the remote processor, receiving service from the remote processor, waiting in the ready–task queue until the next rescheduling. Individual waiting times may also be readily computed by applying Little's formula, since mean number of tokens in each place and transition rates are directly reported by GreatSPN.

The analysis of the 2x2 models seems to reach the computational limits of the available tools when the multitasking level is in the range of 4 to 5 (see Table 6), therefore the direct analysis of meshes with larger dimensions does not seem feasible with the present tools. However, the analysis of the single task models shows that the results obtained in the case of the 2x3 and 3x3 architectures do not differ significantly from the results obtained in the 2x2 case; moreover, results seem to move towards asymptotic values. This conjecture, if validated by further experiments, may suggest that analysis of large–size meshes is not necessary, and general results may be derived from models of reasonable size. Results along this direction were also obtained in [2].

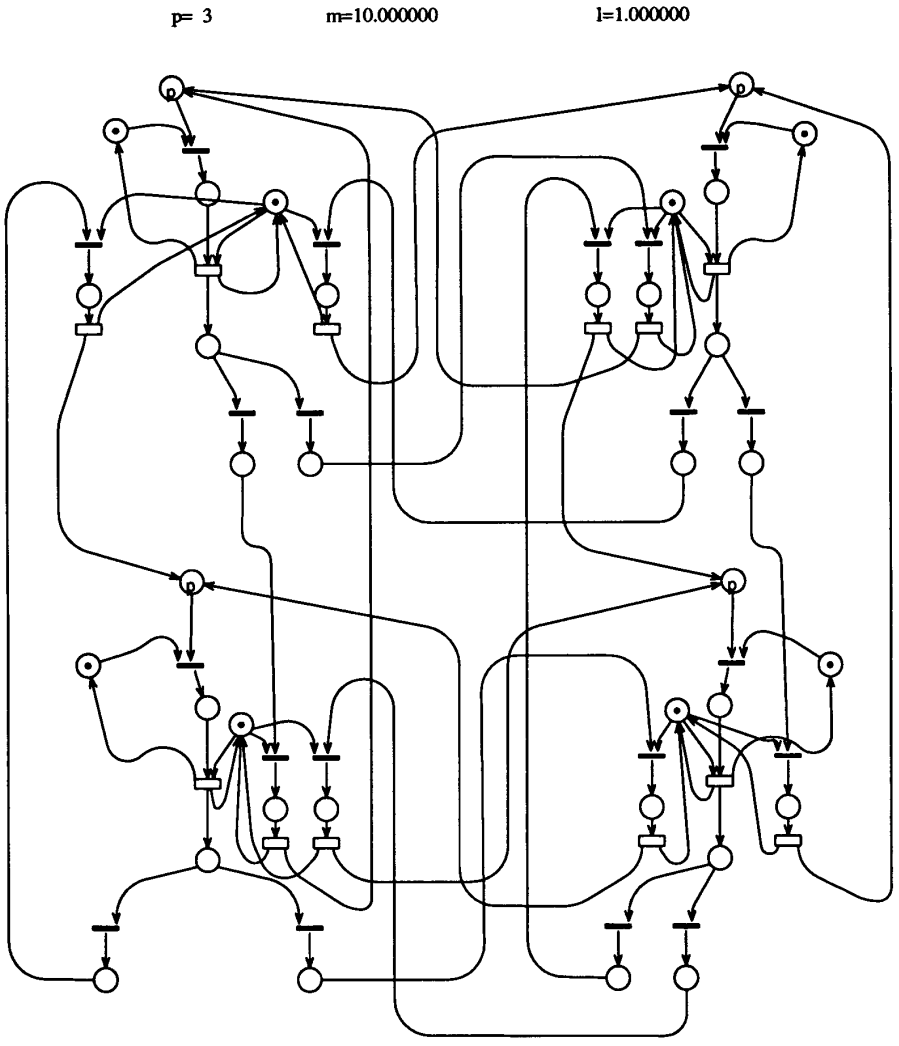

Figure 7: GSPN model of a 2x2 mesh in the case of preemptive interaction and multitasking.

multitasking level (p)	throughput	cycle time	%busy	latency	# of states
1	0.827	1.209	91.0%	1.209	57
2	0.903	1.107	99.3%	2.214	461
3	0.909	1.101	99.9%	3.301	1837
4	0.909	1.100	99.9%	4.400	5147
5	0.909	1.100	100%	5.500	11681

Table 5: Performance results: 2x2 mesh, preemptive interaction with multitasking.

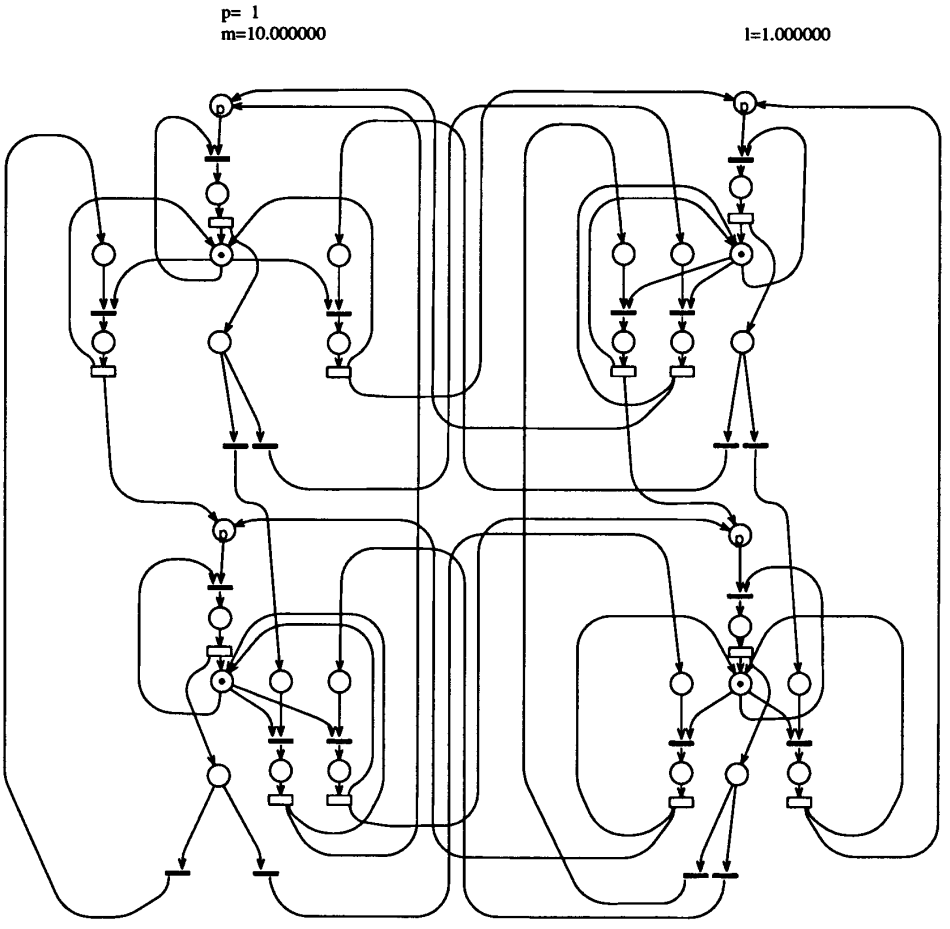

Figure 8: GSPN model of a 2x2 mesh in the case of non–preemptive interaction and multitasking.

multitasking level (p)	throughput	cycle time	%busy	latency	# of states
1	0.604	1.657	66.4%	1.657	189
2	0.756	1.324	83.1%	2.648	5327
3	0.838	1.193	92.2%	3.579	37527
4	0.879	1.138	96.7%	4.552	141537

Table 6: Performance results: 2x2 mesh, non–preemptive interaction with multitasking.

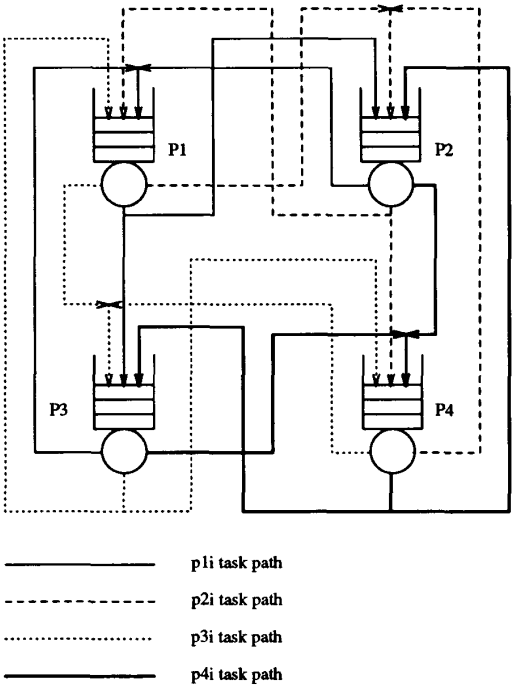

Figure 9: Queueing network model of a 2x2 mesh.

multitasking level	throughput	cycle time	%busy	latency
1	0.780	1.282	85.8%	1.477
2	0.882	1.134	97.0%	2.570
3	0.902	1.108	99.3%	3.728
4	0.907	1.102	99.8%	4.913
5	0.909	1.100	99.9%	6.108

Table 7: Performance results: 2x2 mesh, processor sharing model.

3.3 Queueing Network Models

Concurrent architectures as those analyzed in this paper can also be modeled using queueing network techniques. We discuss in this section the effectiveness and the main limits of this technique in comparison with the GSPN-based one. Figure 9 shows the basic closed multiclass queueing network model for the 2x2 mesh. Each processor is modeled by a service station that can give service to different classes of customers:

- the local processes, whose service requirements correspond to the amount of needed computation,
- the external requests, whose service time corresponds to the needs of external computation.

The history of each process is therefore an activity on the local processor followed by an activity on one of the neighbour processors (randomly choosen). The number of classes corresponds to the number of processing elements.

The model of Figure 9 only specifies the customer flow; to describe the details of the system behavior, the service policy of each service station must be defined. Let's denote with p_{ij} the $i-th$ process running on the $j-th$ processor P_j. In the case of the model described by the GSPN of Figure 7 the following rules must be applied to each service station:

- a random selection policy is adopted for the customer class p_{ij} on processor (queue) P_j,
- a random selection policy is adopted for the customer class p_{ij} on processors (queues) P_k with $k \neq j$,
- on processor P_k, processes p_{ij} with $j \neq k$ preempt processes p_{ij} with $k = j$.

A similar set of rules can be developed to model a system with a behavior as the one described by the GSPN model of Figure 8.

Any required policy can be analyzed, indeed, if one can refer to a simulation tool able to describe the specific rules for each service station. Of course, the same is true in the case of GSPN models, since any simulation approach is not limited by the state explosion. As a bonus, GSPN modeling allows an explicit formal definition of the service policy at each service station and thus some formal verification.

The advantage of the queueing network modeling technique is evident if the network falls into one of those cases for which a product form solution holds [9]. As far as the modeling of concurrent architectures is concerned, the only reasonable case is the one in which each service station is a single server with PS (Processor Sharing) policy. The results for this case (Table 7) have been obtained using SUPERNET [10], a Mean Value Analysis-based queueing network solution package. The computing time in this case becomes neglegible.

4 Conclusions

We have presented a GSPN-based approach to the modeling of concurrent architectures with mesh interconnection topology and uniform workload. Different behaviors have been discussed and analyzed first at the single processor level. Models for (massively) parallel architectures are obtained by replication and simple interconnection. Performance results show that non-preemptive interaction policies, such those implemented by advanced concurrent programming paradigms (e.g., CSP), may yield also good performance, provided that a minimum multitasking level is implemented (in our models only 3 tasks per processor). Though this work has reported results for mesh-connected processor arrays, investigation of alternative physical arrangements (e.g., toroidal and hypercube) is also in progress.

In this paper we have focused on how the GSPN model can be effectively used for the analysis of the interactions between the high-level process cooperation and the processor interconnection topology. In

MIMD structures the relevance of synchronization mechanisms and level of multiprocessing at the processing elements is very high, and the use of modeling techniques to tackle both the logical and quantitative facets of these problems is mandatory.

It must be emphasized that the complexity of the analysis requires the use of sophisticated tools for GSPN description, structural analysis, and computation of the numerical results. Only the smallest toy examples can be handled without adequate software support, and even in those cases errors are very likely to occur. The software package GreatSPN [12] provides an advanced environment for description and analysis of GSPN models, including both structural analysis capabilities and stochastic solution algorithms. GreatSPN has demonstrated to enable tackling useful problems in modeling of massively parallel MIMD architectures. To further expand the range of problems that can be analyzed, we also plan to exploit a CM-2 Connection Machine as a back–end for reachability graph construction and GSPN solution. According to preliminary estimates, a CM-2 in a 8K-processor configuration and a total memory of 256 Mbyte should increase the range of solvable nets of 2 or 3 orders of magnitude with respect to the cardinality of the state space that can be constructed and analyzed on a workstation. Other possible approaches can try to exploit the network simmetry inherent in many large–scale models or look for reasonable approximations.

References

[1] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Generalized stochastic Petri nets revisited: Random switches and priorities. In *Proc. Int. Workshop on Petri Nets and Performance Models*, pages 44–53, Madison, WI, USA, August 1987. IEEE-CS Press.

[2] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Modeling the software architecture of a prototype parallel machine. In *Proc. 1987 SIGMETRICS Conference*, Banf, Alberta, Canada, May 1987. ACM.

[3] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

[4] H. H. Ammar, S. M. R. Islam, and S. Deng. Performability analysis of parallel and distributed algorithms. In *Proc. 3rd Int. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

[5] M. Annaratone, M. Fillo, K. Nakabayashi, and M. Viredaz. The K2 parallel processor: Architecture and hardware implementation. Technical Report K2 Project – 1989 Progress Report, Integrated Systems Laboratory, Federal Institute of Technology, Zurich, Swiss, December 1989.

[6] S. Antonelli, F. Baiardi, S. Pelagatti, and M. Vanneschi. Communication cost and process mapping in massively parallel system: A static approach. Technical Report TR/12-89, Dip. di Informatica, Univ. di Pisa, Pisa, Italy, March 1989.

[7] W. C. Athas and C. L. Seitz. Multicomputers: Message–passing concurrent computers. *Computer*, 21(8), August 1988.

[8] G. Balbo, G. Chiola, G. Franceschinis, and G. Molinar Roet. Generalized stochastic Petri nets for the performance evaluation of FMS. In *Proc. 1987 Int. Conference on Robotics and Automation*, pages 1013–1018, Raleigh, NC, USA, April 1987. IEEE-CS Press.

[9] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[10] S. C. Bruell, G. Balbo, S. Ghanta, and P. V. Afshari. A mean value analysis based package for the solution of product-form queueing network models. In *Proc. Int. Conf. on Modelling Techniques and Tools for Performance Analysis*, Paris, May 1984. INRIA.

[11] G. Bruno and P. Biglia. Performance evaluation and validation of tool handling in flexible manufacturing systems using Petri nets. In *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[12] G. Chiola. A software package for the analysis of generalized stochastic Petri net models. In *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[13] E. Gressier. A stochastic Petri net model for Ethernet. In *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[14] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[15] O. C. Ibe, A. Sathaye, R. C. Howe, and K. S. Trivedi. Stochastic Petri net modeling of VAXCluster system availability. In *Proc. Int. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

[16] Ltd INMOS. *The Transputer Databook*. Bristol, 1989.

[17] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Generalized stochastic Petri nets: Definition at the net level. Submitted for publication to the Journal of the ACM, 1989.

[18] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transaction on Computers*, 31(9):913–917, September 1982.

[19] C. L. Seitz. Concurrent VLSI architectures. *IEEE Transactions on Computers*, 33(12), December 1984.

[20] C. J. Wang and V. P. Nelson. Petri net performance modeling of a modified mesh–connected parallel computer. *Parallel Computing*, 17, 1991.