Lecture Notes in Computer Science 616

K. Jensen (Ed.)

Application and Theory of Petri Nets 1992

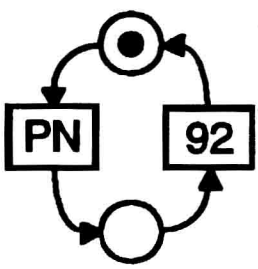
13th International Conference
Sheffield, UK, June 1992
Proceedings

Springer-Verlag

K. Jensen (Ed.)

Application and Theory of Petri Nets 1992

13th International Conference
Sheffield, UK, June 22-26, 1992
Proceedings



Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
W-7500 Karlsruhe, FRG

Juris Hartmanis
Department of Computer Science
Cornell University
5149 Upson Hall
Ithaca, NY 14853, USA

Volume Editor

Kurt Jensen
Computer Science Department, Aarhus University
Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark

CR Subject Classification (1991): F.1-3, C.1-2, D.4, I.6

ISBN 3-540-55676-1 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-55676-1 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author/editor
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Lecture Notes in Computer Science 616
Edited by G. Goos and J. Hartmanis

Advisory Board: W. Brauer D. Gries J. Stoer

Lecture Notes in Computer Science

For information about Vols. 1–529
please contact your bookseller or Springer-Verlag

Vol. 530: D. H. Pitt, P.-L. Curien, S. Abramsky, A. M. Pitts, A. Poigné, D. E. Rydeheard (Eds.), Category Theory and Computer Science. Proceedings, 1991. VII, 301 pages. 1991.

Vol. 531: E. M. Clarke, R. P. Kurshan (Eds.), Computer-Aided Verification. Proceedings, 1990. XIII, 372 pages. 1991.

Vol. 532: H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Grammars and Their Application to Computer Science. Proceedings, 1990. X, 703 pages. 1991.

Vol. 533: E. Börger, H. Kleine Büning, M. M. Richter, W. Schönfeld (Eds.), Computer Science Logic. Proceedings, 1990. VIII, 399 pages. 1991.

Vol. 534: H. Ehrig, K. P. Jantke, F. Orejas, H. Reichel (Eds.), Recent Trends in Data Type Specification. Proceedings, 1990. VIII, 379 pages. 1991.

Vol. 535: P. Jorrand, J. Kelemen (Eds.), Fundamentals of Artificial Intelligence Research. Proceedings, 1991. VIII, 255 pages. 1991. (Subseries LNAI).

Vol. 536: J. E. Tomayko, Software Engineering Education. Proceedings, 1991. VIII, 296 pages. 1991.

Vol. 537: A. J. Menezes, S. A. Vanstone (Eds.), Advances in Cryptology – CRYPTO '90. Proceedings. XIII, 644 pages. 1991.

Vol. 538: M. Kojima, N. Megiddo, T. Noma, A. Yoshise, A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems. VIII, 108 pages. 1991.

Vol. 539: H. F. Mattson, T. Mora, T. R. N. Rao (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Proceedings, 1991. XI, 489 pages. 1991.

Vol. 540: A. Prieto (Ed.), Artificial Neural Networks. Proceedings, 1991. XIII, 476 pages. 1991.

Vol. 541: P. Barahona, L. Moniz Pereira, A. Porto (Eds.), EPIA '91. Proceedings, 1991. VIII, 292 pages. 1991. (Subseries LNAI).

Vol. 542: Z. W. Ras, M. Zemankova (Eds.), Methodologies for Intelligent Systems. Proceedings, 1991. X, 644 pages. 1991. (Subseries LNAI).

Vol. 543: J. Dix, K. P. Jantke, P. H. Schmitt (Eds.), Nonmonotonic and Inductive Logic. Proceedings, 1990. X, 243 pages. 1991. (Subseries LNAI).

Vol. 544: M. Broy, M. Wirsing (Eds.), Methods of Programming. XII, 268 pages. 1991.

Vol. 545: H. Alblas, B. Melichar (Eds.), Attribute Grammars, Applications and Systems. Proceedings, 1991. IX, 513 pages. 1991.

Vol. 546: O. Herzog, C.-R. Rollinger (Eds.), Text Understanding in LILOG. XI, 738 pages. 1991. (Subseries LNAI).

Vol. 547: D. W. Davies (Ed.), Advances in Cryptology – EUROCRYPT '91. Proceedings, 1991. XII, 556 pages. 1991.

Vol. 548: R. Kruse, P. Siegel (Eds.), Symbolic and Quantitative Approaches to Uncertainty. Proceedings, 1991. XI, 362 pages. 1991.

Vol. 549: E. Ardizzone, S. Gaglio, F. Sorbello (Eds.), Trends in Artificial Intelligence. Proceedings, 1991. XIV, 479 pages. 1991. (Subseries LNAI).

Vol. 550: A. van Lamsweerde, A. Fugetta (Eds.), ESEC '91. Proceedings, 1991. XII, 515 pages. 1991.

Vol. 551:S. Prehn, W. J. Toetenel (Eds.), VDM '91. Formal Software Development Methods. Volume 1. Proceedings, 1991. XIII, 699 pages. 1991.

Vol. 552: S. Prehn, W. J. Toetenel (Eds.), VDM '91. Formal Software Development Methods. Volume 2. Proceedings, 1991. XIV, 430 pages. 1991.

Vol. 553: H. Bieri, H. Noltemeier (Eds.), Computational Geometry - Methods, Algorithms and Applications '91. Proceedings, 1991. VIII, 320 pages. 1991.

Vol. 554: G. Grahne, The Problem of Incomplete Information in Relational Databases. VIII, 156 pages. 1991.

Vol. 555: H. Maurer (Ed.), New Results and New Trends in Computer Science. Proceedings, 1991. VIII, 403 pages. 1991.

Vol. 556: J.-M. Jacquet, Conclog: A Methodological Approach to Concurrent Logic Programming. XII, 781 pages. 1991.

Vol. 557: W. L. Hsu, R. C. T. Lee (Eds.), ISA '91 Algorithms. Proceedings, 1991. X, 396 pages. 1991.

Vol. 558: J. Hooman, Specification and Compositional Verification of Real-Time Systems. VIII, 235 pages. 1991.

Vol. 559: G. Butler, Fundamental Algorithms for Permutation Groups. XII, 238 pages. 1991.

Vol. 560: S. Biswas, K. V. Nori (Eds.), Foundations of Software Technology and Theoretical Computer Science. Proceedings, 1991. X, 420 pages. 1991.

Vol. 561: C. Ding, G. Xiao, W. Shan, The Stability Theory of Stream Ciphers. IX, 187 pages. 1991.

Vol. 562: R. Breu, Algebraic Specification Techniques in Object Oriented Programming Environments. XI, 228 pages. 1991.

Vol. 563: A. Karshmer, J. Nehmer (Eds.), Operating Systems of the 90s and Beyond. Proceedings, 1991. X, 285 pages. 1991.

Vol. 564: I. Herman, The Use of Projective Geometry in Computer Graphics. VIII, 146 pages. 1992.

Vol. 565: J. D. Becker, I. Eisele, F. W. Mündemann (Eds.), Parallelism, Learning, Evolution. Proceedings, 1989. VIII, 525 pages. 1991. (Subseries LNAI).

Vol. 566: C. Delobel, M. Kifer, Y. Masunaga (Eds.), Deductive and Object-Oriented Databases. Proceedings, 1991. XV, 581 pages. 1991.

Vol. 567: H. Boley, M. M. Richter (Eds.), Processing Declarative Kowledge. Proceedings, 1991. XII, 427 pages. 1991. (Subseries LNAI).

Vol. 568: H.-J. Bürckert, A Resolution Principle for a Logic with Restricted Quantifiers. X, 116 pages. 1991. (Subseries LNAI).

Vol. 569: A. Beaumont, G. Gupta (Eds.), Parallel Execution of Logic Programs. Proceedings, 1991. VII, 195 pages. 1991.

Vol. 570: R. Berghammer, G. Schmidt (Eds.), Graph-Theoretic Concepts in Computer Science. Proceedings, 1991. VIII, 253 pages. 1992.

Vol. 571: J. Vytopil (Ed.), Formal Techniques in Real-Time and Fault-Tolerant Systems. Proceedings, 1992. IX, 620 pages. 1991.

Vol. 572: K. U. Schulz (Ed.), Word Equations and Related Topics. Proceedings, 1990. VII, 256 pages. 1992.

Vol. 573: G. Cohen, S. N. Litsyn, A. Lobstein, G. Zémor (Eds.), Algebraic Coding. Proceedings, 1991. X, 158 pages. 1992.

Vol. 574: J. P. Banâtre, D. Le Métayer (Eds.), Research Directions in High-Level Parallel Programming Languages. Proceedings, 1991. VIII, 387 pages. 1992.

Vol. 575: K. G. Larsen, A. Skou (Eds.), Computer Aided Verification. Proceedings, 1991. X, 487 pages. 1992.

Vol. 576: J. Feigenbaum (Ed.), Advances in Cryptology - CRYPTO '91. Proceedings. X, 485 pages. 1992.

Vol. 577: A. Finkel, M. Jantzen (Eds.), STACS 92. Proceedings, 1992. XIV, 621 pages. 1992.

Vol. 578: Th. Beth, M. Frisch, G. J. Simmons (Eds.), Public-Key Cryptography: State of the Art and Future Directions. XI, 97 pages. 1992.

Vol. 579: S. Toueg, P. G. Spirakis, L. Kirousis (Eds.), Distributed Algorithms. Proceedings, 1991. X, 319 pages. 1992.

Vol. 580: A. Pirotte, C. Delobel, G. Gottlob (Eds.), Advances in Database Technology – EDBT '92. Proceedings. XII, 551 pages. 1992.

Vol. 581: J.-C. Raoult (Ed.), CAAP '92. Proceedings. VIII, 361 pages. 1992.

Vol. 582: B. Krieg-Brückner (Ed.), ESOP '92. Proceedings. VIII, 491 pages. 1992.

Vol. 583: I. Simon (Ed.), LATIN '92. Proceedings. IX, 545 pages. 1992.

Vol. 584: R. E. Zippel (Ed.), Computer Algebra and Parallelism. Proceedings, 1990. IX, 114 pages. 1992.

Vol. 585: F. Pichler, R. Moreno Díaz (Eds.), Computer Aided System Theory – EUROCAST '91. Proceedings. X, 761 pages. 1992.

Vol. 586: A. Cheese, Parallel Execution of Parlog. IX, 184 pages. 1992.

Vol. 587: R. Dale, E. Hovy, D. Rösner, O. Stock (Eds.), Aspects of Automated Natural Language Generation. Proceedings, 1992. VIII, 311 pages. 1992. (Subseries LNAI).

Vol. 588: G. Sandini (Ed.), Computer Vision – ECCV '92. Proceedings. XV, 909 pages. 1992.

Vol. 589: U. Banerjee, D. Gelernter, A. Nicolau, D. Padua (Eds.), Languages and Compilers for Parallel Computing. Proceedings, 1991. IX, 419 pages. 1992.

Vol. 590: B. Fronhöfer, G. Wrightson (Eds.), Parallelization in Inference Systems. Proceedings, 1990. VIII, 372 pages. 1992. (Subseries LNAI).

Vol. 591: H. P. Zima (Ed.), Parallel Computation. Proceedings, 1991. IX, 451 pages. 1992.

Vol. 592: A. Voronkov (Ed.), Logic Programming. Proceedings, 1991. IX, 514 pages. 1992. (Subseries LNAI).

Vol. 593: P. Loucopoulos (Ed.), Advanced Information Systems Engineering. Proceedings. XI, 650 pages. 1992.

Vol. 594: B. Monien, Th. Ottmann (Eds.), Data Structures and Efficient Algorithms. VIII, 389 pages. 1992.

Vol. 595: M. Levene, The Nested Universal Relation Database Model. X, 177 pages. 1992.

Vol. 596: L.-H. Eriksson, L. Hallnäs, P. Schroeder-Heister (Eds.), Extensions of Logic Programming. Proceedings, 1991. VII, 369 pages. 1992. (Subseries LNAI).

Vol. 597: H. W. Guesgen, J. Hertzberg, A Perspective of Constraint-Based Reasoning. VIII, 123 pages. 1992. (Subseries LNAI).

Vol. 598: S. Brookes, M. Main, A. Melton, M. Mislove, D. Schmidt (Eds.), Mathematical Foundations of Programming Semantics. Proceedings, 1991. VIII, 506 pages. 1992.

Vol. 599: Th. Wetter, K.-D. Althoff, J. Boose, B. R. Gaines, M. Linster, F. Schmalhofer (Eds.), Current Developments in Knowledge Acquisition - EKAW '92. Proceedings. XIII, 444 pages. 1992. (Subseries LNAI).

Vol. 600: J. W. de Bakker, K. Huizing, W. P. de Roever, G. Rozenberg (Eds.), Real-Time: Theory in Practice. Proceedings, 1991. VIII, 723 pages. 1992.

Vol. 601: D. Dolev, Z. Galil, M. Rodeh (Eds.), Theory of Computing and Systems. Proceedings, 1992. VIII, 220 pages. 1992.

Vol. 602: I. Tomek (Ed.), Computer Assisted Learning. Proceedigs, 1992. X, 615 pages. 1992.

Vol. 603: J. van Katwijk (Ed.), Ada: Moving Towards 2000. Proceedings, 1992. VIII, 324 pages. 1992.

Vol. 604: F. Belli, F.-J. Radermacher (Eds.), Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. Proceedings, 1992. XV, 702 pages. 1992. (Subseries LNAI).

Vol. 605: D. Etiemble, J.-C. Syre (Eds.), PARLE '92. Parallel Architectures and Languages Europe. Proceedings, 1992. XVII, 984 pages. 1992.

Vol. 606: D. E. Knuth, Axioms and Hulls. IX, 109 pages. 1992.

Vol. 607: D. Kapur (Ed.), Automated Deduction – CADE-11. Proceedings, 1992. XV, 793 pages. 1992. (Subseries LNAI).

Vol. 608: C. Frasson, G. Gauthier, G. I. McCalla (Eds.), Intelligent Tutoring Systems. Proceedings, 1992. XIV, 686 pages. 1992.

Vol. 609: G. Rozenberg (Ed.), Advances in Petri Nets 1992. VIII, 472 pages. 1992.

Vol. 610: F. von Martial, Coordinating Plans of Autonomous Agents. XII, 246 pages. 1992. (Subseries LNAI).

Vol. 612: M. Tokoro, O. Nierstrasz, P. Wegner (Eds.), Object-Based Concurrent Computing. Proceedings, 1991. X, 265 pages. 1992.

Vol. 613: J. P. Myers, Jr., M. J. O'Donnell (Eds.), Constructivity in Computer Science. Proceedings, 1991. X, 247 pages. 1992.

Vol. 614: R. G. Herrtwich (Ed.), Network and Operating System Support for Digital Audio and Video. Proceedings, 1991. XII, 403 pages. 1992.

Vol. 615: O. Lehrmann Madsen (Ed.), ECOOP '92. European Conference on Object Oriented Programming. X, 425 pages. 1992.

Vol. 616: K. Jensen (Ed.), Application and Theory of Petri Nets 1992. Proceedings, 1992. VIII, 398 pages. 1992.

Preface

This volume contains the proceedings of the 13th International Conference on Application and Theory of Petri Nets. The aim of the Petri net conferences is to create a forum for discussing progress in the application and theory of Petri nets. Typically, the conferences have 150-200 participants – one third of these coming from industry while the rest are from universities and research institutions. The conferences always take place in the last week of June.

The previous conferences were held in Strasbourg, France (1980), Bad Honnef, Germany (1981), Varenna, Italy (1982), Toulouse, France (1983), Aarhus, Denmark (1984), Espoo, Finland (1985), Oxford, United Kingdom (1986), Zaragoza, Spain (1987), Venice, Italy (1988), Bonn, Germany (1989), Paris, France (1990), Aarhus, Denmark (1991).

The conferences and a number of other Petri net activities are coordinated by a steering committee with the following members: M. Ajmone Marsan (Italy), J. Billington (Australia), H.J. Genrich (Germany), C. Girault (France), K. Jensen (Denmark), G. de Michelis (Italy), T. Murata (USA), C.A. Petri (Germany; Honorary Member), W. Reisig (Germany), G. Roucairol (France), G. Rozenberg (The Netherlands; Chairman), M. Silva (Spain),

The 1992 conference is organized by the School of Computing and Management Sciences at Sheffield City Polytechnic, England. In addition to the conference there is a tool exhibition and there are four different tutorials (both at the introductory and at the more advanced levels).

We would like to thank very much all those who submitted papers to the Petri net conference. 77 papers and project presentations were submitted, and 25 have been accepted for presentation. Invited lectures are given by G. Balbo (Italy), M. Hennessy (United Kingdom) and W. Reisig (Germany).

The submitted papers were evaluated by a programme committee with the following members: W. Brauer (Germany), G. Chiola (Italy), G. Cutts (United Kingdom), F. De Cindio (Italy), R. Devillers (Belgium), M. Diaz (France), U. Goltz (Germany), T. Hildebrand (France), R. Hopkins (United Kingdom), N. Husberg (Finland), K. Jensen (Denmark; Chairman), H.C.M. Kleijn (The Netherlands), J. Martínez (Spain), T. Murata (USA), K. Onaga (Japan), D. Simpson (United Kingdom), G. Wheeler (Australia), J. Winkowski (Poland), W. Zuberek (Canada). The program committee meeting took place at GMD, Bonn. We would like to express our gratitude to the members of the programme committee, and to all the referees who assisted them. The names of the referees are listed on the following page.

We also thank the organizers at Sheffield City Polytechnic: P. Collingwood, G. Cutts (Chairman), I. Jelly, M. Love, R. MacMillan, S. Morton, G. Redfearn and G. Roberts. Finally, we would like to mention the excellent cooperation with Springer-Verlag and to thank A. Paysen who handled all the submissions and compiled the final manuscript.

Aarhus, Denmark
April 1992

Kurt Jensen

List of Referees for Petri Nets 1992

M. Agoulmine
M. Ajmone Marsan
M. Akatsu
C. Anglano
M. Aoyama
P. Azema
G. Balbo
K. Barkaoui
D. Barnard
E. Battiston
F. Bause
A. Beltramini
S. Ben Ahmed
L. Bernardinello
G. Berthelot
A. Bertoni
E. Best
J. Billington
R. Bosworth
O. Botti
A. Bourguet
J. Brown
J. Campos
H. Carstensen
H. Chehaibar
A. Cheng
P. Chrzastowski-Wachtel
S. Christensen
R. Coelho
J.M. Colom
G. Conte
J.P. Courtiat
G. de Michelis
P. Degano
R. DeLemos
J. Desel
S. Donatelli
C. Dutheillet
J. Engelfriet
S. English
J. Esparza
P. Estraillier
J. Ezpeleta
G. Findlow
G. Franceschinis
H. J. Genrich
C. Ghezzi
C. Girault
H.J.M. Goeman
R. Gold
M. Goldwurm
D. Gomm
A. Guia
N. Guelfi
N. Götz
J. Hall
C. Hanen
N.D. Hansen
H. Hasegawa
I. Hatono
X. He
A. Heise
B. Henderson
K. Hiraishi
S. Honiden
H.J. Hoogeboom
P.W. Hoogers
H. Hußmann
J.M. Ilie
C. Johnson
G. Juanole
B. Keck
P. Kemper
E. Kindler
T. Kobayashi
K. Komota
F. Kordon
M. Koutny
P. Lee
D.L. Lee
M. Leuschel
J. Lilius
J. Lobo
A. Mader
T. Massart
T. Matsumoto
G. Mauri
D. McCue
A. Moslemie
M. Mukund
T. Murata
D. Murphy
J. Murphy
H. Müller
M. Nielsen
K. Parker
E. Pelz
W. Penczek
L. Petrucci
M. Pezze
M. Pinna
J. Pitrel
A. Poigné
L. Pomello
B. Pradin-Chezalviel
P. Pulli
A. Ramirez
L. Rapanotti
M. Rauhamaa
W. Reisig
M. Ribaudo
B. Sanchez
V. Sassone
M. Sereno
S. M. Shatz
M. Shields
T. Shimura
C. Simone
A. P. Sistla
M. Silva
V. Sliva
E. Smith
N. Speirs
R. Stroud
I. Suzuki
H. Tamura
P. Taylor
E. Teruel
M. Tiusanen
N. Treves
N. Uchihira
T. Ushio
R. Valette
A. Valmari
K. Varpaaniemi
F. Vernadat
W. Vogler
K. Voss
R. Walter
T. Watanabe
J. Whitworth
A. Yakovlev
B. Zouari
P. Østergaard

Table of Contents

Invited Papers

Submitted Papers

Project Papers

Performance Issues in Parallel Programming

Gianfranco Balbo

Dipartimento di Informatica, Università di Torino,
corso Svizzera 185, 10149 Torino, Italy
e-mail: balbo@di.unito.it

Abstract. The development of parallel applications requires the availability of tools that support their debugging and tuning. GSPN represent a formalism that is well suited for the construction of formal models of parallel programs that can be used for both validation and evaluation purposes. The analysis of GSPN models of parallel programs provides the information that is needed for deciding whether the objectives contained in the specifications of an application are met and for distributing the computation on a parallel architecture. In this paper we discuss a methodology for directly constructing a GSPN model of an application from its code and for deriving the parameters that are needed for obtaining the optimal allocation of the components of a parallel application on the computational units of a parallel architecture. A simple example is used throughout the paper to illustrate the different steps of the methodology and to show how these GSPN models can be used to check the efficiency of a parallel application.

1 Introduction

Parallel computers are widely recognized as the equipments capable of meeting the demands of high performance computing posed by new scientific and real time applications. Parallel programming is however still difficult because of the lack of tools that help in developing and debugging new implementations. Computer architectures and language characteristics restrict the class of applications that can be easily implemented [15] with parallel programs; indeed, concurrency, communication, synchronization and nondeterminism make the manual assessment of the correctness and of the efficiency of parallel programs particularly difficult.

The study of the characteristics of an application both from the point of view of correctness and performance, can be done at different stages of the software life cycle [27]. For instance, an analysis can be performed at the specification stage to ensure that the implementation will meet certain real time constraints [19] or to support the results of rapid prototyping [8]; alternatively, an evaluation can be carried on after the completion of the implementation to verify whether the results conform to the original specifications or to assess its efficiency through the computation of performance indices such as resource consumption indicators [28,16].

* This work has been supported in part by Ministero dell'Universita' e della Ricerca Scientifica e Tecnologica - 40% Project - and by the Italian National Research Council - Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, Grant N. 91.00879.PF69.

In order to perform these tasks efficiently, tools must work on models of the real application that differ depending on the goals of the analyses. Different representations allow to characterize the behaviour of a program with different levels of detail [34,30,20]. It is however important that these representations be compatible so that abstract models can be augmented with more detailed descriptions of specific components to allow a modular and efficient analysis of large programs. In any case, the choice of the modelling formalism that is used throughout the software life cycle must easily integrate within the programming environment and must allow the characterization of both the static and dynamic behaviour of the program by means of analytic as well as simulation techniques.

Parallel programs are developed to obtain high-performance computing and is thus a major aspect of their implementations that of allocating their components on the computational units of parallel architectures in order to maximize their efficiency. Real parallel architectures are however characterized by a limited number of processors and by a limited degree of connectivity (not every computational unit can directly communicate with any other unit of the architecture) that constraint their capabilities. Intuition suggests that processes that are concurrently active should be allocated on different nodes of a parallel computer in order to exploit parallelism. Communication among processes can however modify this picture. Indeed, processes allocated on the same processor communicate through common memory in a very fast manner. Communications among processes allocated on different processors, on the other hand, take place through relatively slow links. It follows that when mapping a parallel program on a parallel architecture, several counteracting effects must be taken into accounts. For instance, processes that are simultaneously active and that interact very strongly may be better allocated on the same processor trading the loss of parallelism with the reduction of communication latency. On the contrary, processes with very loose interactions can be easily allocated on separate processors.

These considerations are usually formalized as an optimization problem whose objective function accounts for the communication and processing costs. The form of the objective function depends on the structure of the parallel program and the coefficients depend on the amount of data exchanged among processes, on their mutual distance, and on the amount of local processing performed by each processor [23].

To solve this problem, many different models of parallel programs have been presented in the literature, typical examples being the *task graph* [18] and the *communication graph* [33] in which nodes represent processes and arcs correspond to communications and/or synchronizations. A great effort has been devoted to devise methods for obtaining the optimal allocation strategy [7,31], but little or no effort has been devoted to the problem of constructing these models starting from real programs.

In this paper we show how Generalized Stochastic Petri Net (GSPN) [3] models can be used to estimate characteristic parameters of parallel programs, and thus to construct the corresponding communication graphs, with the possibility of exploiting all the classical analysis results based on GSPN for validating and evaluating these applications. In particular, we address the problem of constructing a GSPN model of the program flow, the possibility of doing this automatically, and the trade-off between automatic generation of the model, efficiency of the analysis and usefulness

of the results. We consider the impact of including control variables in the models and the way of representing communication among processes. Finally we discuss the choice of which features to include in the model, i.e. the abstraction level of the representation, that strongly depends on the type of analysis we want to perform.

To apply these techniques, we focus our attention on parallel languages that allow applications to be organized as sets of cooperating tasks using a message passing paradigm of the rendez-vous type. Major examples of languages of this type are Occam, CSP and Ada. We shall also take into account the case of communications of the non-blocking type, like those allowed by CsTools which is a programming environment available on. Meiko's [25] parallel computers.

The work reported in this paper is part of a project for the definition and the implementation of an integrated programming environment for the development of parallel applications organized as sets of parallel processes that interact by message passing following the CSP [21] paradigm. A single formalism based on GSPN is used whithin this environment for specifying, designing, implementing, testing, and evaluating parallel programs. Detailed information on this project can be found in [5,6,22].

The balance of the paper is the following. Section 2 discusses the possibility of using static analysis techniques for characterizing the behaviour of parallel programs. Section 3 describes the transformation steps that must be undertaken to produce the desired model starting from the code of parallel programs. Section 4 overviews the graph models that are used to solve the problem of mapping parallel programs on parallel architectures. Section 5 describes how the communication graph of a parallel program can be constructed starting from the solution of the GSPN representation of the same application. Section 6 indicates how the model of a parallel program together with the indications of its allocation on a parallel architecture can be used to check the efficiency of the implementation. Section 7 concludes the paper with indications on the problems that are still open and with a discussion of future research efforts that will be undertaken in this field.

2 Static Analysis of Parallel Programs

Two types of strategies may be followed to infer the properties of parallel programs and to obtain their optimal execution. Programs that exhibit very dynamic behaviours may be run on representative sets of input data and under the control of dynamic allocation policies. The results obtained from these sample executions are used to identify the properties of these programs and the observed balance of workloads and communications may be interpreted as a measure of the quality of the allocation policies. Programs that are instead characterized by an internal structure may be analyzed independently of their inputs to identify their properties and may be optimized a priori (i.e., statically) in order to make the best use of the capabilities of the architecture.

Although appealing, the first strategy may be impractical because of the difficulty of choosing representative test cases, of measuring the behaviour of a parallel application, and of the necessity of devising fast decision policies that allocate resources and restructure applications without forcing the programs to wait for long

times while the choices are made. It is thus quite conceivable to follow a static approach in which high optimization costs are justified by the time-critical aspects of the applications and in which a careful characterization of the computation may yield important advantages when the program is repeatedly executed.

The static analysis of an application consists in characterizing a program with a formal model that is subsequently studied to infer the properties of the program itself. This model is used first to analyze the correctness of the program and subsequently to assess its efficiency. Since during static analysis nothing is known about the run-time behaviour of the program (for example its input data), no assumption is made on which of the possible execution paths is actually followed by the program. Static analysis thus tends to account for many program behaviours that would never occur in real executions. Three types of anomalies may be detected by static analysis in a distributed/parallel environment [29]: unconditional faults, conditional faults, and nonfaults. Unconditional faults correspond to errors that will definitely occur during program executions. Conditional faults represent instead errors whose occurrence either depends on non deterministic choices or on specific sets of input data. Nonfaults, finally, are errors which are reported by the static analyzer although they will never occur during program executions: nonfaults are detected when the correct behaviours of programs are ensured by control variables that are ignored during static analysis. Static analysis thus provides a pessimistic picture of program behaviour that must be kept under control to avoid that issuing too many warnings for unverified events make the programmers completely disregard the whole result of the study.

Once a parallel program is considered correct it must also run fast. It is thus of paramount importance to be able to estimate the performance of the adopted solution. Again in the case of a static approach, this optimization must relay on the formal model of the program and must disregard any information on the input data. A probabilistic interpretation of the formal model can be convenient to concisely account for the many possible execution of the program using probability distributions.

GSPN have been chosen as the formalism for the definition of the formal model that is used for static analysis because of their capability of supporting both validation and performance evaluation of real systems using basically the same model [26,4]. Moreover, GSPN are also particularly well suited for the study considered in this paper because of the possibility that they offer of representing both the characteristics of the architecture (the hardware) and the peculiarities of the program (the software) of parallel computers [2,1].

3 Modelling CSP-like programs

The parallel programs we consider in this paper conform to a CSP style; a typical application is organized as a set of procedures that include statements equivalent to the **SEQ**, **ALT**, **PAR**, **if**, **while**, **repeat**, **for**, **?** and **!** of CSP. The two operators **?** and **!** are used to indicate inputs and outputs; communications are of the rendezvous style. Communication among processes is assumed to happen through declared common channels. This is the facility provided by Occam [24] and it represents a generalization of the basic mechanism of CSP.

We assume that there is a process that initiates the concurrent execution of a certain number of processes. Processes can be activated by a **PLACED PAR** instruction or by any **PAR** instruction contained in an already activated process. In the command **PAR A B ENDPAR**, A and B are either simple **SEQ** commands or whole processes. In the second case, the execution of the command corresponds to instantiating and activating a copy of each process. All these features are present in the language that we used for the implementation of the examples. It represents a parallel extension of C and is called DISC [22].

Having discussed the type of languages we are interested in, we must decide which features of a program we want to represent in the model. It is important to remark that GSPN could account for all the details of a real parallel program. This approach would lead to enormous nets, cluttered with lots of details, with huge reachability graphs, difficult to analyze, and providing results problematic to interpret. The choice of what to model must be driven by the type of results we want to obtain. Each program can be represented with different levels of abstraction yielding different consequences in terms of complexity and accuracy of their analysis. In particular, it is clear that, if we want to study deadlocks, all process synchronizations must be carefully described; similarly, if the objective of our study is the analysis of the communications among processes, all the possible rendez-vous must be taken into consideration. In what follows we shall describe different possible choices of the level of abstraction at which we want to represent our programs.

3.1 Modelling process schemes

A first choice is that of including in the model control flow, process activation and communication statements only [34,30,17,13]. In particular we model every single communication statement as well as each **PAR**, **SEQ**, **ALT**, **if**, **while**, **repeat**, and **for** statement that includes in its body a communication. All the sequences that do not include any of these instructions are represented in the GSPN as timed transitions whose associated delays depend on the length of their executions.

Using this level of abstraction, parallel programs are translated into GSPN models according to the following procedure:

1. A set of process schemes is derived from a parallel program consisting of procedures by coalescing into single macrostatements all those sequences of statements that do not include any communication or any **PAR** with named processes.

2. Each process scheme yields a GSPN model: named processes are represented in the net with single transitions (i.e., they are not yet substituted with their translations), communication statements are represented as immediate transitions (thus disregarding any type of synchronization connected with the rendez-vous), and macrostatements are substituted by timed transitions (whose associated delay is an estimate of their execution time).

3. Starting from the initial process, all process names are expanded with a copy of their corresponding GSPN models. The substitution continues in depth-first mode until all the names have been replaced.

4. Pairs of communication transitions that belong to different processes and that represent their (mutual) communication are fused to concretely represent the synchronization deriving from the rendez-vous protocol.

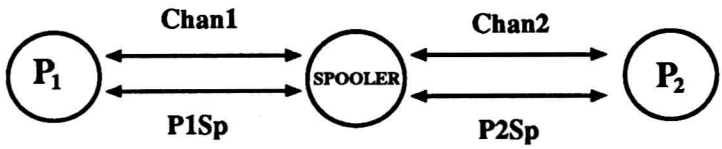


Fig. 1. A "printer-spooler" system

We shall present this translation process in a rather informal way with the help of a simple example. The program used as an example in this section is a "printer-spooler" that accepts the print requests of two user programs P_1 and P_2. A scheme of the structure of the *Spooler* and of the user programs P_1 and P_2 is depicted in Fig. 1, where $Chan_1$ and P_1Sp ($Chan_2$ and P_2Sp) are the channels used for communication between the *Spooler* and process P_1 (P_2).

The corresponding Occam-like code is presented in Fig. 2. Process P_1 (P_2) executes a loop of computation and requests to print the results. If process P_1 (P_2) wants to send a sequence of packets to be printed to the *Spooler*, it first sends a message on channel $Chan_1$ ($Chan_2$) containing a request to transmit k (n) packets and then enters a loop where at each iteration one of the k (n) packets is sent on channel P_1Sp (P_2Sp). The *Spooler* process executes a loop (the external one) looking for print requests coming from the two programs. When it receives a request for $k(n)$ packets from $Chan_1$ ($Chan_2$) it enters a reception loop of $k(n)$ messages from P_1Sp (P_2Sp).

```
Main
while true
   ALT
      Chan1 ? x :
         for i=x-1 downto 0
            P1Sp? pckt
            ⟨ Print pckt ⟩
         endfor
      Chan2 ? x :
         for i= x-1 downto 0
            P2Sp? pckt
            ⟨ Print pckt ⟩
         endfor
endwhile
```

```
P1
while true
   ⟨ P1 computes pckt[1..k] ⟩
   Chan1 ! k
   for i=k-1 downto 0
      P1Sp? pckt[i]
   endfor
endwhile

P2
while true
   ⟨ P2 computes pckt[1..n] ⟩
   Chan2 ! n
   for i=n-1 downto 1
      P2Sp? pckt[i]
   endfor
endwhile
```

Fig. 2. A spooler system code

The first translation step consists of generating the net representing the control structure. This task requires an abstraction phase that eliminates all the parts

that are irrelevant with respect to control. This can be a by-product of the compiler of the language. For instance, in our case we use the process scheme generated by the DISC compiler [22]. The code presented in Fig. 2 is already in a form where only control structures that include PAR and communications have been kept. All other operations are summarized within angle brackets.

The second translation step consists of producing a first GSPN structure from the process scheme. Each process is considered separately. The basic translation rules are shown in Fig. 3, where A, B and A_i are used to indicate either named processes or sequences of statements. All comments that follow refer to this figure. Each process can be characterized by a GSPN with one input place and one output place; the same is true for any construct of our CSP-like application.

The translation of the process definition is shown in (a). In (b) it is shown the translation of an **if** statement: the two alternatives are represented by a non deterministic choice (two immediate transitions with a common input place). If nothing is known about the relative frequencies of the outcomes of the condition evaluation, equal weights are assigned to the two immediate transitions. The **case** statement can be translated in the same way, yielding a structure with as many parallel sequences as there are cases. The **while** statement is instead depicted in (c); the same discussion as for the **if** statement applies. In the **PAR** statement (d) the timed transition represents the time needed to activate the individual branches of the **PAR**. The final transition of this subnet represents the fact that the **PAR** completes its execution only when this is true for all branches: it is an immediate transition because it expresses a logical condition. The translation of the **ALT** statement is shown in (e), **ALT** is modeled as a non deterministic choice among possible communications; the G_i are communication statements (input guards) or simple conditions and are thus represented as immediate transitions. In the **SEQ** statement (f) the output place of the translation of A_i is superposed with the input place of the translation of A_{i+1}. Composite statements (corresponding to angle brackets in the program scheme) are represented by timed transitions with one input and one output place (g). The presence of a communication is captured in the model by introducing an immediate transition (h).

The automatic implementation of these rules is quite straightforward as it corresponds to simple modifications of any algorithm that translates a program into a flowchart. Applying these translation rules mechanically, many useless immediate transitions and places are generated. For example in (b) the subnet comprising the output places of $\tau(A)$ and $\tau(B)$ the output place of the **if** statement and the two immediate transitions connected to them can be actually substituted by a single place which is the output place of the **if**. The same is true also in the case of the **ALT** statement. Moreover, all sequences of immediate transitions of the 1-in 1-out type can be coalesced into a single one (of course this is not true if a transition represents communication, since it will be manipulated during the fourth step of the procedure). Removing useless transitions corresponds to the application of well known reduction techniques [32,9].

The third translation step consists of replacing all transitions which represent named processes with their net representations. The substitution can be easily performed by superimposing the input (output) place of the process equivalent subnet