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

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

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

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