A Hierarchical Approach to Workload Characterization for Parallel Systems[⋆]

M. Calzarossa[1], G. Haring[2], G. Kotsis[2], A. Merlo[1], D. Tessera[1]

[1] Dipartimento di Informatica e Sistemistica, Università di Pavia
Via Abbiategrasso 209, I-27100 Pavia, Italy
e-mail: {mcc,merlo,tessera}@gilda.unipv.it
[2] Institut für Angewandte Informatik und Informationssysteme, Universität Wien
Lenaugasse 2/8, A-1080 Vienna, Austria
e-mail: {haring,gabi}@ani.univie.ac.at

Abstract. Performance evaluation studies are to be an integral part of the design and tuning of parallel applications. We propose a hierarchical approach to the systematic characterization of the workload of a parallel system, to be kept as modular and flexible as possible. The methodology is based on three different, but related, layers: the application, the algorithm, and the routine layer. For each of these layers different characteristics representing functional, sequential, parallel, and quantitative descriptions have been identified. These characteristics are specified in a system independent way to clearly separate between the workload description and the architecture description. Taking also architectural and mapping features into consideration, the hierarchical workload characterization can be applied to any type of performance studies.

1 Introduction

The main reason to use parallel systems is to get more and better performance, i.e. either to be able to solve larger problems or to solve given problems in a shorter time. So, in fact, performance is the driving force to develop new applications for parallel systems. The performance of such systems depends on many more aspects than that of a uniprocessor system, due to the more complex behavior of its hardware and software components. Therefore, from the performance point of view, there are more challenges but also deeper pitfalls [CS93].

Performance evaluation studies are to be an integral part of the design and tuning of applications, i.e. the workload, to reduce the development and performance debugging costs [CMM$^+$94]. More or less for all these studies, but especially for performance predictions of applications under development, a monolithic approach is not appropriate. Due to the fact that workload, architecture, and mapping are the features with the largest impact on the performance of parallel systems, various performance prediction frameworks have been developed, where these aspects can be specified and modified in structured and

⋆ This work was partially supported by the Austrian-Italian Cooperation 1994-96 Project Number 32, and by the 40% MURST Project.

flexible ways, as independent as possible from each other. Examples are the PAPS tool set [WKH93, WH94], the PRM [Fer90, Fer92] and the Mitchele–Thiel [Her92, MT93] approaches. But even these approaches use a rather “flat” description of the workload, reducing the understandability by increasing the representation complexity.

In this paper we propose a hierarchical approach to the systematic characterization of the workload of parallel systems able to manage the complexity of such a workload following the same principles as used in software engineering. Hierarchical descriptions of workloads have already proven their usability for representing workloads of interactive [Har86] and distributed [CHS88, RVH95] systems. Our hierarchical methodology is general enough to be easily adapted according to the objectives of the performance study. These objectives reach from getting “performance figures” for an existing application (or algorithm), which is composed of smaller components with known behavior, to predicting the performance of an application under development before implementing and executing it. In all these cases, a hierarchical approach is natural and helps to manage the complexity of the workload characterization and to drive the performance studies.

The methodology we present here can be applied both to functional and data parallel programming paradigms. From the architecture point of view, we can deal with distributed and shared memory, as well as virtual shared memory systems. Our characterization approach, which represents a logical view of the workload, is system independent, in the sense that the architectural characteristics come in when the final performance model is built. At that stage, either the communication or the memory access behavior will be modeled using explicit timings or hidden delays, according to the load and data distributions.

The paper is organized as follows. In Section 2 we introduce the underlying methodology of the proposed hierarchical approach and we define each layer in terms of its components. The characterization of each layer is discussed in Section 3. Section 4 concludes the paper with an outlook on future activities.

2 Methodology

Workload characterization studies are widely applied in the software design process and when a synthetic description of existing programs is required. Basic principles of software engineering technology, like abstraction and hierarchy, are taken to manage the complexity of the software engineering process or to better understand existing applications. In this section, the basic features of our hierarchical workload modeling methodology are pointed out. This approach corresponds to the user’s or developer’s mental model (view) of the software system.

Figure 1 (a) shows the mental view corresponding to the user’s natural perception of a software system. Three possible levels of granularity are identified. Solving a specific “problem” requires the utilization of a few “methods” and

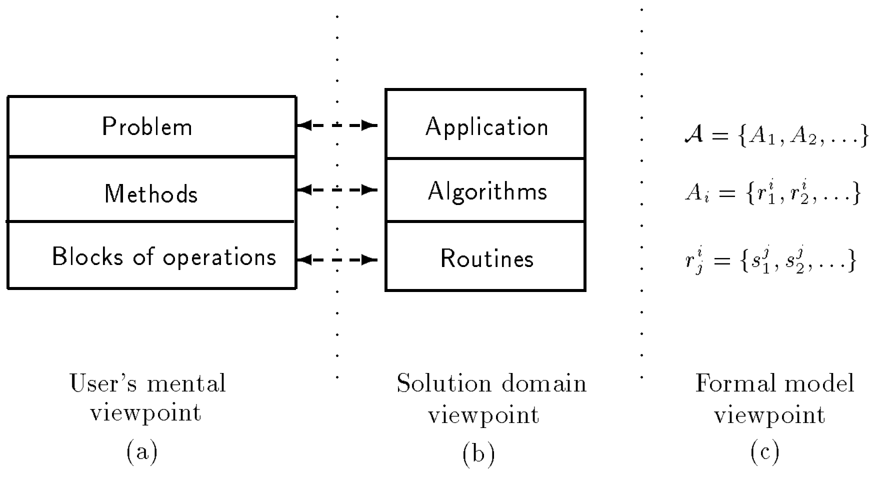
Problem
Methods
Blocks of operations
Application
Algorithms
Routines
$\mathcal{A} = \{A_1, A_2, \ldots\}$
$A_i = \{r_1^i, r_2^i, \ldots\}$
$r_j^i = \{s_1^j, s_2^j, \ldots\}$
User's mental viewpoint
(a)
Solution domain viewpoint
(b)
Formal model viewpoint
(c)

Fig. 1. Correspondences between user's mental view, solution domain, and formal model

these methods can be described by means of "blocks of operations" they are built with.

As can be seen from Fig. 1 (b), there exists a conceptual correspondence between these software levels and potential layers of characterization of the load itself in a so called solution domain. The solution domain preserves the logical subdivision of the mental view and translates it in terms of "application", "algorithms", and "routines".

The workload description adopted in our methodology is based on these hierarchical layers. At each of them, a generic but more formal specification of the load is given as follows (see Fig. 1 (c)).

Application Layer At the very top layer the load is characterized by a coarse granularity, in that, a whole application corresponds to the problem to be solved (e.g., biomolecular modeling, fluid dynamics). As solving a problem requires a set of methods to be applied, an application $\mathcal{A}$ can be described in terms of the underlying algorithms A_i $(i = 1, 2, \ldots)$ in the solution domain. Formally:

$$\mathcal{A} = \{A_1, A_2, \ldots\}$$

Algorithm Layer At the intermediate layer of our methodology, the granularity is refined in order to characterize the components of an application. For numerical applications, algorithms solving systems of nonlinear equations are examples for the methods used at this layer. An algorithm A_i can be described by means of the routines r_j^i $(j = 1, 2, \ldots)$ it consists of; more precisely:

$$A_i = \{r_1^i, r_2^i, \ldots\}$$

Routine Layer At the bottom layer, the routines used to implement an algorithm are considered. A routine r_j^i is composed of code segments s_k^j ($k = 1, 2, \ldots$), e.g., subroutines, functions, loops. At the very finest granularity, a routine can even be a single statement. Formally, a routine is specified as:

$$r_j^i = \{s_1^j, s_2^j, \ldots\}$$

According to the application under study and to the objectives of the analysis to be performed, the granularity identified for the components at each layer may vary. Sometimes a few methods can be considered as a single algorithm component because they are logically related. Furthermore, characterization layers can be omitted. A small problem, which might require only one solution method, can be characterized starting at the algorithm layer, hence omitting the application one.

3 Characterization of the Layers

After having identified and defined three possible layers and the corresponding components, we investigate the characteristics for each layer. Following the concept of an adaptive methodology, we keep the characterization modular and system independent.
Such a characterization starts from a functional description of the load, where its basic components are identified. Their sequences are then analyzed and the exploitation of potential structural parallelism is then considered. The characterization ends with a quantitative description. It is possible to omit certain characteristics if they are irrelevant for the particular system under test or if they do not provide any information useful in the performance evaluation study. Table 1 summarizes the characterization of the three layers.

	Descriptions			
Layer	Functional	Sequential	Parallel	Quantitative
Application	set of algorithms	application behavior graph	structural parallelism graph	multiplicity of algorithms, volume of data
Algorithm	set of routines	algorithm behavior graph	structural parallelism graph	multiplicity of routines, volume of data
Routine	set of statements	routine behavior graph	structural parallelism graph	multiplicity of statements, volume of data

Table 1. Characterization at the three hierarchical layers

Functional Description The identification of the basic components for each layer is the starting point for any further description. The statements at the routine layer, the routines at the algorithm layer, and finally the algorithms at the application layer are the basic building blocks. As already mentioned in the previous section, the granularity at each layer depends on the workload under study and on the objective of the analysis. For example, it is possible to lump two algorithms together into a single algorithm component A_i at the application layer. This lumping implies that all the other characteristics can only be specified in terms of the A_i component.

Since, at this layer, we neither introduce any order among the components, nor their possible multiplicities, it is sufficient to identify the set of components.

Sequential Description Once we have defined the set of components at each layer (algorithms, routines, statements), we can represent their sequences by the corresponding models. Directed graph models, called behavior graphs, are adopted. The components at each layer correspond to the nodes of such graphs. If there is a positive (i.e., non-zero) probability, that component i is followed by component j, then an arc is drawn from node i to node j annotated with the associated probability.

A behavior graph could also be interpreted as a state transition diagram, where the components correspond to the states, and the arcs denote the transitions annotated with their probabilities.

As an example we consider an application which consists of seven algorithms (represented by the nodes $A_1, A_2, \ldots, A_7$). As can be seen from Fig. 3 (a) nodes A_1, A_2, A_3, A_4 and A_5 are in a sequential order, while A_5 is either followed by algorithm A_6 (with a probability equal to 0.2) or by algorithm A_7. Then, from A_6 there is a loop back to algorithm A_3.

Moving one layer down in our hierarchy would mean to derive the corresponding behavior graphs for the interactions of the routines within each algorithm. Each routine can in turn be represented in terms of the graph of its statements.

Parallel Description Graph models have proven to be a convenient method for specifying the characteristics of parallel applications (e.g., synchronization or concurrency).

We introduce Structural Parallelism Graphs (SPGs), which allow a clear and concise characterization of the potential parallelism, i.e. the full parallelism that can be exploited within a workload component without any constraints with respect to the number of available processors.

A Structural Parallelism Graph is a directed graph $SPG = (V, D, W)$, where V is a set of vertices corresponding to the workload components and $D = (D^P \cup D^A)$ is a set of directed arcs[3]. W is a set of weight distributions for outgoing arcs. If no distribution is specified, all the outgoing paths are taken. If a distribution is given, the corresponding arc is taken with the associated probability.

[3] An arc can either represent an activation signal and/or an explicit data exchange.

Two types of arcs, precedence and activation arcs, are introduced in our methodology. A precedence arc, belonging to D^P, represents a precedence relation between two components, such that the second component is started after the first one has finished. The two components can never run in parallel, since they are strictly sequential. Graphically such an arc is depicted as a thin arrow (see Fig. 3 (b)). An activation arc, belonging to D^A, expresses a relation between two components such that the second component is activated by the first one before its completion. Thus, the two components may run in parallel, but the second one must not start before the first one. The graphical representation is a thick arrow (see Fig. 3 (b)). A communication or synchronization activity among two components, that have already been activated or started, can be neglected, since it does not change the potential parallelism.

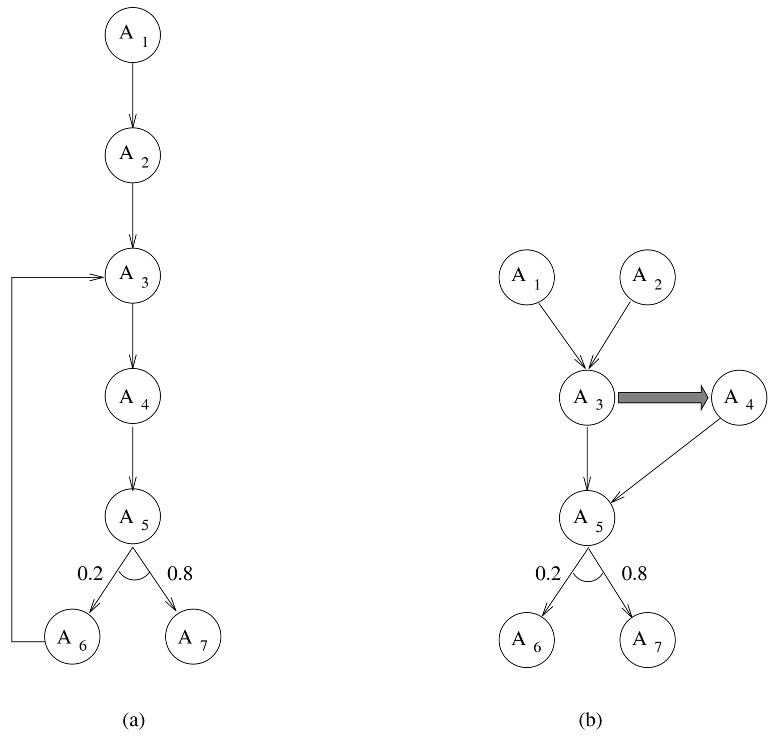

Fig. 2. Examples of a sequential (a) and a parallel (b) description

For example, the application behavior graph depicted in Fig. 3 (a) has been transformed in the corresponding SPG (see Fig. 3 (b)). By looking at the internal behavior of the algorithms A_1 and A_2 we find out that they are completely independent of each other. To represent this in the SPG, we simply draw two nodes A_1 and A_2. The results that are produced by A_1 and A_2 are both needed by A_3. So we add the node A_3 and connect A_1 and A_2 to A_3 using precedence arcs. A_4 has to wait for data computed by algorithm A_3 at an early stage, i.e. A_4 does not have to wait for A_3 to finish. We therefore connect A_3 and A_4 with an activation arc. A_5 can start as soon as A_3 and A_4 have finished (precedence arcs) and depending on the output of A_5 either A_6 or A_7 follows (precedence arcs with associated weights). The loop back to A_3 does not have any influence on the potential parallelism and it is therefore not represented.

Quantitative Description All previous characteristics have specified the type of components and their possible interactions. To derive a performance model, we also need information about the internal characteristics of the components in terms of their multiplicity as well as the volume of data used and/or exchanged. Note that this description is still system independent and it does not consider the characteristics of the system. For example, we do not specify the duration of the components and the architecture dependent communication or memory access patterns.

The proposed characterization is general enough such that it can be customized according to the objective of the performance studies. In the case of the existing software, measurements are basic information for bridging the gap between the logical description of the workload and the system characteristics [RS94]. In such a case, a bottom–up approach of the methodology is advisable. From measurements we can also infer the information to be used for performance prediction of applications under design. In this case, a top down approach, which starts from a coarse grain representation of the workload, seems more appropriate.

4 Conclusions

A hierarchical methodology for the systematic characterization of the workload of parallel systems has been proposed. Such a methodology is based on a layered approach, which takes into consideration various types of descriptions of the workload components. The characterization obtained by means of this methodology focuses on the load, while discarding specific information on the architectures and mapping strategies. In this sense, our characterization is system independent, modular, and flexible and supports independent specifications of workload, architectures, and mapping.

The system dependent characteristics are to be considered at the final stage, where the relations and transformations between the various characterizations on the different layers have also to be analyzed, to obtain the final performance model. Our future work will concentrate on the study of these interactions and on corresponding performance modeling techniques.

References

[CHS88] M. Calzarossa, G. Haring, and G. Serazzi. Workload Modelling for Computer Networks. In U. Kastens and F. J. Rammig, editors, *Architektur und Betrieb von Rechnersystemen*, pages 324–339. Springer Verlag, 1988.

[CMM$^+$94] M. Calzarossa, L. Massari, A. Merlo, D. Tessera, and A. Malagoli. Performance Debugging of Parallel Programs. In *Proc. AICA Annual Conference*, pages 541–556, Palermo, Italy, 1994.

[CS93] M. Calzarossa and G. Serazzi. Workload Characterization: a Survey. *Proc. of the IEEE*, 81(8):1136–1150, 1993.

[Fer90] A. Ferscha. The PRM-Net Model - An Integrated Performance Model for Parallel Systems. Technical report, Austrian Center for Parallel Computation, University of Vienna, 1990.

[Fer92] A. Ferscha. A Petri Net Approach for Performance Oriented Parallel Program Design. *Journal of Parallel and Distributed Computing*, 15(3):188–206, 1992.

[Har86] G. Haring. Fundamental Principles of Hierarchical Workload Description. In G. Serazzi, editor, *Workload Characterization of Computer Systems and Computer Networks*, pages 101–110. North Holland, 1986.

[Her92] U. Herzog. Performance Evaluation as an Integral Part of System Design. In M. Becker et al., editors, *Proceedings of the Transputers'92 Conference*. IOS Press, 1992.

[MT93] A. Mitschele-Thiel. Automatic Configuration and Optimization of Parallel Transputer Applications. In *Proceedings of the World Transputer Congress*, Aachen, Germany, 1993. IOS Press.

[RS94] E. Rosti and G. Serazzi. Workload Characterization for Performance Engineering of Parallel Applications. In *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing*, pages 457–462. IEEE Computer Society Press, 1994.

[RVH95] S. V. Raghavan, D. Vasukiammaiyar, and G. Haring. Hierarchical approach to building generative network load models. *Computer Networks and ISDN*, 1995. (to appear).

[WH94] H. Wabnig and G. Haring. PAPS - The Parallel Program Performance Prediction Toolset. In *Proc. of the 7^{th} Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 284–304. Springer-Verlag, 1994.

[WKH93] H. Wabnig, G. Kotsis, and G. Haring. Performance Prediction of Parallel Programs. In G. Haring and G. Kotsis, editors, *Proc. of the 7th GI/ITG Conference on Measurement, Modelling and Performance Evaluation of Computer Systems*, pages 64–76. Springer Verlag, 1993.

This article was processed using the LaTeX macro package with LLNCS style