Towards the Automatic Mapping of ASSIST Applications for the Grid

Marco Aldinucci[1] and Anne Benoit[2]

[1] Inst. of Information Science and Technologies
National Research Council (ISTI–CNR)
Via Moruzzi 1, Pisa I-56100, Italy
aldinuc@di.unipi.it
[2] LIP, Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Anne.Benoit@ens-lyon.fr

Abstract. One of the most promising technical innovations in present-day computing is the invention of grid technologies which harness the computational power of widely distributed collections of computers. However, the programming and optimisation burden of a low level approach to grid computing is clearly unacceptable for large scale, complex applications. The development of grid applications can be simplified by using high-level programming environments. In the present work, we address the problem of the mapping of a high-level grid application onto the computational resources. In order to optimise the mapping of the application, we propose to automatically generate performance models from the application using the process algebra PEPA. We target in this work applications written with the high-level environment ASSIST, since the use of such a structured environment allows us to automate the study of the application more effectively.

Key words: high-level parallel programming, grid, ASSIST, PEPA, automatic model generation, skeletons.

1 Introduction

A grid system is a geographically distributed collection of possibly parallel, interconnected processing elements, which all run some form of common grid middleware (e.g. Globus services) [15]. The key idea behind grid-aware applications is to make use of the aggregate power of distributed resources, thus benefiting from a computing power that falls far beyond the current availability threshold in a single site. However, developing programs able to exploit this potential is highly programming intensive. Programmers must design concurrent programs that can execute on large-scale platforms that cannot be assumed to be homogeneous, secure, reliable or centrally managed. They must then implement these programs correctly and efficiently. As a result, in order to build efficient

grid-aware applications, programmers have to address the classical problems of parallel computing as well as grid-specific ones:

1. Programming: code all the program details, take care about concurrency exploitation, among the others: concurrent activities set up, mapping/scheduling, communication/synchronisation handling and data allocation.

2. Mapping & Deploying: deploy application processes according to a suitable mapping onto grid platforms. These may be highly heterogeneous in architecture and performance. Moreover, they are organised in a cluster-of-clusters fashion, thus exhibiting different connectivity properties among all pairs of platforms.

3. Dynamic environment: manage resource unreliability and dynamic availability, network topology, latency and bandwidth unsteadiness.

Hence, the number and quality of problems to be resolved in order to draw a given QoS (in term of performance, robustness, etc.) from grid-aware applications is quite large. The lesson learnt from parallel computing suggests that any low-level approach to grid programming is likely to raise the programmer's burden to an unacceptable level for any real world application.

Therefore, we envision a layered, high-level programming model for the grid, which is currently pursued by several research initiatives and programming environments, such as ASSIST [21], eSkel [10], GrADS [19], ProActive [7], Ibis [20], Higher Order Components [12, 13]. In such an environment, most of the grid specific efforts are moved from programmers to grid tools and run-time systems. Thus, the programmers have only the responsibility of organising the application specific code, while the programming tools (i.e. the compiling tools and/or the run-time systems) deal with the interaction with the grid, through collective protocols and services [14].

In such a scenario, the QoS and performance constraints of the application can either be specified at compile time or varying at run-time. In both cases, the run-time system should actively operate in order to fulfil QoS requirements of the application, since any static resource assignment may violate QoS constraints due to the very uneven performance of grid resources over time. As an example, ASSIST applications exploit an autonomic (self-optimisation) behaviour. They may be equipped with a QoS contract describing the degree of performance the application is required to provide. The ASSIST run-time environment tries to keep the QoS contract valid for the duration of the application run despite possible variations of platforms' performance at the level of grid fabric [6, 5]. The autonomic features of an ASSIST application rely heavily on run-time application monitoring, and thus they are not fully effective for application deployment since the application is not yet running. In order to deploy an application onto the grid, a suitable mapping of application processes onto grid platforms should be established, and this process is quite critical for application performance.

This problem can be addressed by defining a performance model of an ASSIST application in order to statically optimise the mapping of the application onto a heterogeneous environment, as shown in [1]. The model is generated from the source code of the application, before the initial mapping. It is expressed with the process algebra PEPA [17], designed for performance evaluation. The

use of a stochastic model allows us to take into account aspects of uncertainty which are inherent to grid computing, and to use classical techniques of resolution based on Markov chains to obtain performance results. This static analysis of the application is complementary with the autonomic reconfiguration of ASSIST applications, which works on a dynamic basis. In this work we concentrated on the static part to optimise the mapping, while the dynamic management is done at run-time. It is thus an orthogonal but complementary approach.

Structure of the paper. The next section introduces the ASSIST high-level programming environment and its run-time support. Section 3 introduces the Performance Evaluation Process Algebra PEPA, which can be used to model ASSIST applications. These performance models help to optimise the mapping of the application. We present our approach in Section 4, and give an overview of future working directions. Finally, concluding remarks are given in Section 5.

2 The ASSIST environment and its run-time support

ASSIST (A Software System based on Integrated Skeleton Technology) is a programming environment aimed at the development of distributed high-performance applications [21, 3]. ASSIST applications should be compiled in binary packages that can be deployed and run on grids, including those exhibiting heterogeneous platforms. Deployment and run is provided through standard middleware services (e.g. Globus) enriched with the ASSIST run-time support.

2.1 The ASSIST coordination language

ASSIST applications are described by means of a coordination language, which can express arbitrary graphs of modules, interconnected by typed streams of data. Each stream realises a one-way asynchronous channel between two sets of endpoint modules: sources and sinks. Data items injected from sources are broadcast to all sinks. All data items injected into a stream should match the stream type.

Modules can be either sequential or parallel. A sequential module wraps a sequential function. A parallel module *(parmod)* can be used to describe the parallel execution of a number of sequential functions that are activated and run as *Virtual Processes* (VPs) on items arriving from input streams. The VPs may synchronise with the others through barriers. The sequential functions can be programmed by using a standard sequential language (C, C++, Fortran). A *parmod* may behave in a data-parallel (e.g. SPMD/for-all/apply-to-all) or task-parallel (e.g. farm) way and it may exploit a distributed shared state that survives the VPs lifespan. A module can nondeterministically accept from one or more input streams a number of input items, which may be decomposed in parts and used as function parameters to instantiate VPs according to the input and distribution rules specified in the parmod. The VPs may send items or parts of items onto the output streams, and these are gathered according to the output rules. Details on the ASSIST coordination language can be found in [21, 3].

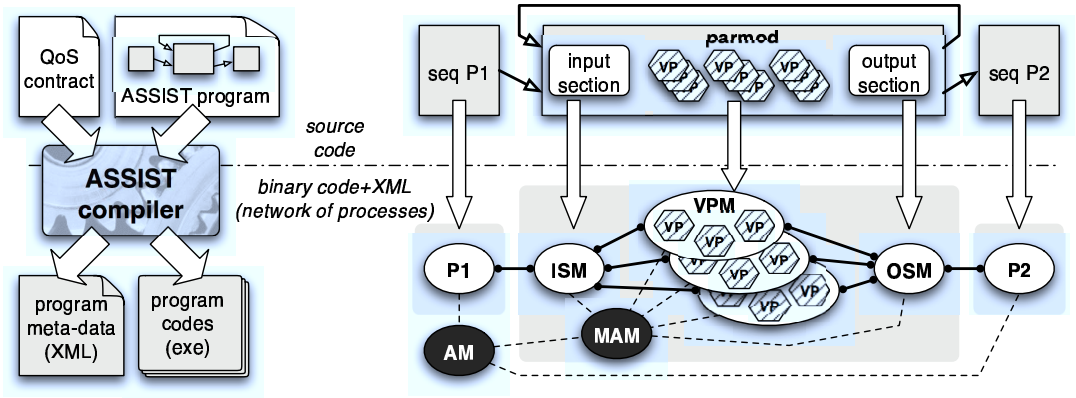

Fig. 1. An ASSIST application and a QoS contract are compiled in a set of executable codes and its meta-data [3]. This information is used to set up a processes network at launch time.

2.2 The ASSIST run-time support

The ASSIST compiler translates a graph of modules into a network of processes. As sketched in Fig. 1, sequential modules are translated into sequential processes, while parallel modules are translated into a parametric (w.r.t. the parallelism degree) network of processes: one *Input Section Manager* (ISM), one *Output Section Manager* (OSM), and a set of *Virtual Processes Managers* (VPMs, each of them running a set of Virtual Processes). The number of VMPs gives the actual parallelism degree of a parmod instance. Also, a number of processes are devoted to application QoS control, e.g. a *Module Adaptation Manager* (MAM), and an *Application Manager* (AM) [6].

The processes that compose an ASSIST application communicate via ASSIST support channels. These can be implemented on top of a number of grid middleware communication mechanisms (e.g. shared memory, TCP/IP, Globus, CORBA-IIOP, SOAP-WS). The suitable communication mechanism between each pair of processes is selected at launch time depending on the mapping of the processes.

2.3 Towards fully grid-aware applications

ASSIST applications can already cope with platform heterogeneity [2], either in space (various architectures) or in time (varying load) [6]. These are definite features of a grid, however they are not the only ones. Grids are usually organised in sites on which processing elements are organised in networks with private addresses allowing only outbound connections. Also, they are often fed through job schedulers. In these cases, setting up a multi-site parallel application onto the grid is a challenge in its own right (irrespectively of its performance). Advance reservation, co-allocation, multi-site launching are currently hot topics of research for a large part of the grid community. Nevertheless, many of these problems should be targeted at the middleware layer level and they are largely

independent of the logical mapping of application processes on a suitable set of resources, given that the mapping is consistent with deployment constraints.

In our work, we assume that the middleware level supplies (or will supply) suitable services for co-allocation, staging and execution. These are actually the minimal requirements in order to imagine the bare existence of any non-trivial, multi-site parallel application. Thus we can analyse how to map an ASSIST application, assuming that we can exploit middleware tools to deploy and launch applications [11].

3 Introduction to performance evaluation and PEPA

In this section, we briefly introduce the Performance Evaluation Process Algebra PEPA [17], with which we can model an ASSIST application. The use of a process algebra allows us to include the aspects of uncertainty relative to both the grid and the application, and to use standard methods to easily and quickly obtain performance results.

The PEPA language provides a small set of combinators. These allow language terms to be constructed defining the behaviour of components, via the activities they undertake and the interactions between them. We can for instance define constants, express the sequential behaviour of a given component, a choice between different behaviours, and the direct interaction between components. Timing information is associated with each activity. Thus, when enabled, an activity $a = (\alpha, r)$ will delay for a period sampled from the negative exponential distribution which has parameter r. If several activities are enabled concurrently, either in competition or independently, we assume that a *race condition* exists between them.

The dynamic behaviour of a PEPA model is represented by the evolution of its components, as governed by the operational semantics of PEPA terms [17]. Thus, as in classical process algebra, the semantics of each term is given via a labelled *multi-transition* system (the multiplicity of arcs are significant). In the transition system a state corresponds to each syntactic term of the language, or *derivative*, and an arc represents the activity which causes one derivative to evolve into another. The complete set of reachable states is termed the *derivative set* and these form the nodes of the *derivation graph*, which is formed by applying the semantic rules exhaustively. The derivation graph is the basis of the underlying Continuous Time Markov Chain (CTMC) which is used to derive performance measures from a PEPA model. The graph is systematically reduced to a form where it can be treated as the state transition diagram of the underlying CTMC. Each derivative is then a state in the CTMC. The *transition rate* between two derivatives P and Q in the derivation graph is the rate at which the system changes from behaving as component P to behaving as Q. Examples of derivation graphs can be found in [17].

It is important to note that in our models the rates are represented as random variables, not constant values. These random variables are exponentially distributed. Repeated samples from the distribution will follow the distribution

and conform to the mean but individual samples may potentially take any positive value. The use of such distribution is quite realistic and it allows us to use standard methods on CTMCs to readily obtain performance results. There are indeed several methods and tools available for analysing PEPA models. Thus, the PEPA Workbench [16] allows us to generate the state space of a PEPA model and the infinitesimal generator matrix of the underlying Markov chain. The state space of the model is represented as a sparse matrix. The PEPA Workbench can then compute the steady-state probability distribution of the system, and performance measures such as throughput and utilisation can be directly computed from this.

4 Performance models of ASSIST applications

PEPA can easily be used to model an ASSIST application since such applications are based on stream communications, and the graph structure deduced from these streams can be modelled with PEPA. Given the probabilistic information about the performance of each of the ASSIST modules and streams, we then aim to find information about the global behaviour of the application, which is expressed by the steady-state of the system. The model thus allows us to predict the run-time behaviour of the application in the long time run, taking into account information obtained from a static analysis of the program. This behaviour is not known in advance, it is a result of the PEPA model.

4.1 PEPA model and performance results

The technical report [1] exposes in details how to model an ASSIST application with PEPA in order to optimise the static mapping of the application. We give in this paper only the general ideas of the approach, and we focus on the on-going and future work, while technical results can be found in the report.

Each ASSIST module is represented as a PEPA component, and the different components are synchronised through the streams of data to model the overall application. The performance results obtained are the probabilities to be in either of the states of the system. From this information, we can determine the bottleneck of the system and decide the best way to map the application onto the available resources.

The PEPA model is generated automatically from the ASSIST source code, during a pre-compilation phase. This task is simplified thanks to some information provided by the user directly in the source code, and particularly the rates associated to the different activities of the PEPA model.

This approach has been introduced on an example of Data Mining classification algorithm [18]. The structure of the application can be represented as a graph, where the ASSIST modules are the nodes and the data streams the arcs. The graph representing this application is displayed in Fig. 2 ❶. The algorithm is designed according to the Divide&Conquer paradigm; all algorithms following the same paradigm could be similarly implemented. It is implemented by means

of four modules: the `start` module is generating the inputs, while the `end` module is collecting outputs. Modules `DC_c45` and `CS_c45` represent the core of the algorithm. The `DC_c45` drives both Divide and Conquer phases: it outputs data items, which are obtained from the split (Divide) or join (Conquer) of input stream items. The `CS_c45` module behaves as "co-processor" of the first module: it receives a data item and sorts it out in such a way that it can be split trivially. Notice that this last module is the only computationally intensive module, while the three others are not.

We have studied the behaviour of the application when mapped in two different ways.

- In the first case (Fig. 2 ❷), we map the loop onto the same cluster, and the `start` and `end` modules onto another. This means that communications on streams s1 and s4 are slow, while they are fast on s2 and s3.
- In the second case (Fig. 2 ❸), we split the loop into two parts, thus `start`, `end` and `DC_c45` are on the same cluster while the computational part `CS_c45` is on the other cluster by itself. In this case, communications on s1 and s4 are fast, while they are slow on s2 and s3.

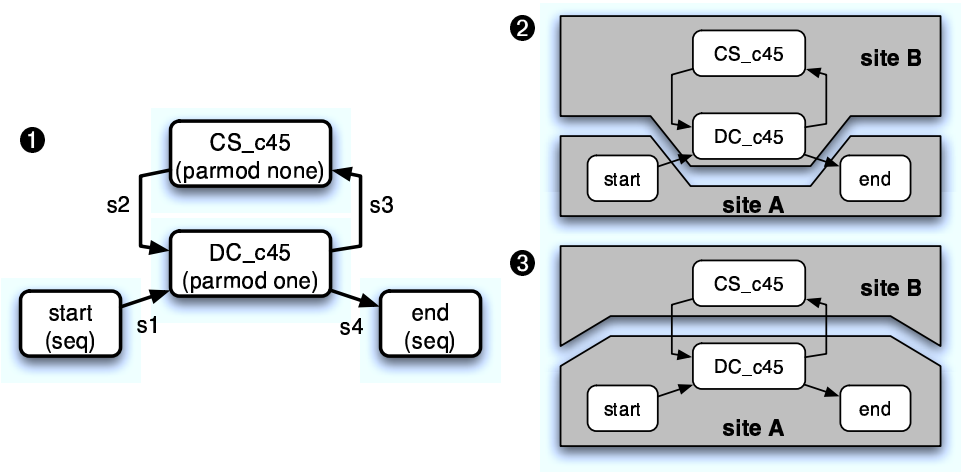

Fig. 2. Graph representation of the classification algorithm (❶) and two different multi-site deployments (❷, ❸).

The results allow us to determine the most efficient mapping, which is the first one, since in this case most of the time is spend in the computational part of the application. The performance results in the second case show that lot of time is spent in the non-computationally intensive modules. This shows that the computational intensive module is constantly waiting for data.

Notice however that it is up to the user to decide which mappings to study, and an exhaustive study of all mappings is probably useless and may cost a lot. The performance models help then to determine, between the mappings chosen by the user, which one is the best.

4.2 Alternative approach to this analysis

The main aim of the analysis performed on the classification algorithm was to compare alternative mappings. In fact, communication and computation rates already include mapping peculiarities (speed of individual links and processors). With the same technique it is also possible to conduct a *predictive analysis*.

Rates are assigned to the PEPA model solely on the basis of the application logical behaviour, assuming uniform speed of connections and processors. In this way, the result of the analysis is not representing a particular mapping, but it rather highlights individual resources (links and processors) requirements, that are used to label the application graph.

These labels represent the expected relative requirements of each module (stream) with respect to other modules (streams) during the application run. In the case of a module the described requirement can be interpreted as the aggregate power of the site on which it will be mapped. On the other hand, a stream requirement can be interpreted as the bandwidth of the network link on which it will be mapped. The relative requirements of parmods and streams may be used to implement mapping heuristics which assign more demanding parmods to more powerful sites, and more demanding streams to links exhibiting higher bandwidths. Whether a fully automatic application mapping is not required, modules and streams requirements can be used to drive a user-assisted mapping process.

Moreover, each parmod exhibits a structured parallelism pattern (a.k.a. skeleton). In many cases, it is thus possible to draw a reliable relationship between the site fabric level information (number and kind of processors, processors and network benchmarks) and the expected aggregate power of the site running a given parmod exhibiting a parallelism pattern [5, 4, 9]. This may enable the development of a mapping heuristic, which needs only information about sites fabric level information, and can automatically derive the performance of a given parmod on a given site.

4.3 Future work

The approach described here considers the ASSIST modules as blocks and does not model the internal behaviour of each module. A more sophisticated approach might be to consider using known models of individual modules and to integrate these with the global ASSIST model, thus providing a more accurate indication of the performance of the application. At this level of detail, distributed shared memory and external services (e.g. DB, storage services, etc) interactions can be taken into account and integrated to enrich the network of processes with dummy nodes representing external services. PEPA models have already been developed for pipeline or deal skeletons [8, 9], and we could integrate such models when the parmod module has been adapted to follow such a pattern.

Analysis precision can be improved by taking into account historical (past runs) or synthetic (benchmark) performance data of individual modules and their communications. This kind of information should be scaled with respect to the

expected performances of fabric resources (platform and network performances), which can be retrieved via the middleware information system (e.g. Globus GIS).

5 Conclusions

In this paper we have presented a way to automatically generate PEPA models from an ASSIST application with the aim of improving the mapping of the application. This is an important problem in grid application optimisation.

It is our belief that having an automated procedure to generate PEPA models and obtain performance information may significantly assist in taking mapping decisions. However, the impact of this mapping on the performance of the application with real code requires further experimental verification. This work is ongoing, and is coupled with further studies on more complex applications.

This ongoing research should be performed between two CoreGRID partners: the ISTI CNR in Pisa, Italy (WP3 - Programming Model), and the ENS (CNRS) in Lyon, France (WP6 - Institute on Resource Management and Scheduling).

Acknowledgments

This work has been partially supported by Italian national FIRB project no. RBNE01KNFP *GRID.it*, by Italian national strategic projects *legge 449/97* No. 02.00470.ST97 and 02.00640.ST97, and by the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

References

1. M. Aldinucci and A. Benoit. Automatic mapping of ASSIST applications using process algebra. Technical report TR-0016, CoreGRID, October 2005.
2. M. Aldinucci, S. Campa, M. Coppola, S. Magini, P. Pesciullesi, L. Potiti, R. Ravazzolo, M. Torquati, and C. Zoccolo. Targeting heterogeneous architectures in ASSIST: Experimental results. In M. Danelutto, M. Vanneschi, and D. Laforenza, editors, *10th Intl Euro-Par 2004: Parallel and Distributed Computing*, volume 3149 of *LNCS*, pages 638–643, Pisa, Italy, August 2004. Springer Verlag.
3. M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, and C. Zoccolo. ASSIST as a research framework for high-performance Grid programming environments. In J. C. Cunha and O. F. Rana, editors, *Grid Computing: Software environments and Tools*. Springer Verlag, January 2006.
4. M. Aldinucci, M. Danelutto, J. Dünnweber, and S. Gorlatch. Optimization techniques for skeletons on grid. In L. Grandinetti, editor, *Grid Computing and New Frontiers of High Performance Processing*, volume 14 of *Advances in Parallel Computing*. Elsevier, October 2005.
5. M. Aldinucci, M. Danelutto, and M. Vanneschi. Autonomic QoS in ASSIST Grid-aware components. In *Proc. of Euromicro PDP 2006: Parallel Distributed and network-based Processing*. IEEE, 2006. To appear.

6. M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo. Dynamic reconfiguration of grid-aware applications in ASSIST. In *11th Intl Euro-Par 2005: Parallel and Distributed Computing*, volume 3648 of *LNCS*, pages 771–781, Lisboa, Portugal, August 2005. Springer Verlag.
7. F. Baude, D. Caromel, and M. Morel. On hierarchical, parallel and distributed components for Grid programming. In V. Getov and T. Kielmann, editors, *Workshop on component Models and Systems for Grid Applications*, ICS '04, Saint-Malo, France, June 2005.
8. A. Benoit, M. Cole, S. Gilmore, and J. Hillston. Evaluating the performance of skeleton-based high level parallel programs. In M. Bubak, D. van Albada, P. Sloot, and J. Dongarra, editors, *The International Conference on Computational Science (ICCS 2004), Part III*, LNCS, pages 299–306. Springer Verlag, 2004.
9. A. Benoit, M. Cole, S. Gilmore, and J. Hillston. Scheduling skeleton-based grid applications using PEPA and NWS. *The Computer Journal*, 48(3):369–378, 2005.
10. M. Cole. Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Computing*, 30(3):389–406, 2004.
11. M. Danelutto, M. Vanneschi, C. Zoccolo, N. Tonellotto, R. Baraglia, T. Fagni, D. Laforenza, and A. Paccosi. Hpc application execution on grids. In *Dagstuhl Seminar Future Generation Grid 2004*, CoreGRID series. Springer, 2005. To appear.
12. J. Dünnweber and S. Gorlatch. HOC-SA: A grid service architecture for higher-order components. In *IEEE International Conference on Services Computing, Shanghai, China*, pages 288–294. IEEE Computer Society Press, September 2004.
13. J. Dünnweber, S. Gorlatch, M. Aldinucci, S. Campa, and M. Danelutto. Behavior customization of parallel components application programming. Technical Report TR-0002, Institute on Programming Model, CoreGRID - Network of Excellence, April 2005.
14. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The Intl. Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.
15. I. Foster and C. Kesselmann, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, December 2003.
16. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proc. of the 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in LNCS, pages 353–368, Vienna, May 1994. Springer-Verlag.
17. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
18. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.
19. S. Vadhiyar and J. Dongarra. Self adaptability in grid computing. *International Journal Computation and Currency: Practice and Experience*, 2005. To appear.
20. R. V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. Hofman, C. Jacobs, T. Kielmann, and H. E. Bal. Ibis: a flexible and efficient Java-based grid programming environment. *Concurrency & Computation: Practice & Experience*, 2005.
21. M. Vanneschi. The programming model of ASSIST, an environment for parallel and distributed portable applications. *Parallel Computing*, 28(12):1709–1732, December 2002.