Future Generation Computer Systems 10 (1994) 305–308

The CAMAS workbench: Computer Aided Migration of Applications System

J.F. de Ronde *, P.M.A. Sloot, M. Beemster, L.O. Hertzberger

Parallel Scientific Computing and Simulation Group, Department of Computer Science - University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Abstract

A simulation methodology for predicting the performance of sequential programs as well as data parallel (SPMD) programs on parallel and distributed platforms is presented. The proposed methodology comprises a parameterised description of the applications as well as the target machines. An integrated simulation procedure estimates the time complexity of the (SPMD) application given a well-defined hardware platform and predicts the execution behaviour. The methodology has been actualised in terms of a toolset currently under development at the University of Amsterdam.

Key words: Performance estimation; Massively parallel platforms; SPMD

1. Introduction

Migration of large sequential applications to parallel and distributed platforms is yet still in its infancy. This is largely due to the fact that industry (that has many large applications of great social and economical relevance) is reserved towards spending great amounts of money and labour on adjusting their codes to parallel distributed memory machines (DMM), while it is not clear that the effort spent will pay off eventually. A simulation environment that can predict the performance of SPMD applications on parallel DMM is of crucial importance in the decision phase of such large projects. In addition it can assist in choosing the most suitable parallel platform for a specific application (e.g. a workstation network or a massively parallel monolith). We have developed a technique that can predict data parallel program performance by means of parameterisation of the distributed memory machine as well as the data parallel application. The Amsterdam Simulation Toolkit (AST) consists of several tools to obtain these parameterisations and to perform simulations using formal machine and application descriptions (i.e. mapping the parameterised application to the parameterised machine). The AST is partially developed within the CAMAS (ESPRIT III) project [5]. The aim of this project is to develop an integrated workbench that can help in migrating existing sequen-

* Corresponding author.

tial Fortran 77 programmes to massively parallel platforms.

Apart from the AST there are three tools being developed within the CAMAS workbench by the University of Southampton:

(i) An interprocedural dependency analyser (IDA) which aids in unraveling dusty deck Fortran 77 codes.
(ii) A domain decomposition tool (DDT) explicitly usable to decompose arbitrary finite element meshes as for instance used in dynamical car crash simulations.
(iii) A static performance estimator (SPE) which statically estimates execution times of (subset) High Performance Fortran data parallel programs.

Below the AST will be discussed in more detail.

2. The Amsterdam Simulation Toolkit (AST)

The AST consists of several tools (Fig. 1):

(1) A machine database that to a high level of detail describes DMM characteristics.
(2) A symbolic application description language in which the SPMD code of interest can be expressed.
(3) A mapping procedure to find an optimal mapping of segmented data to a specific processor topology.
(4) A simulation environment that integrates these three items to obtain an estimation of both the time complexity of the application and the execution behaviour.

An application simulator is used to support the development of the several tools [4].

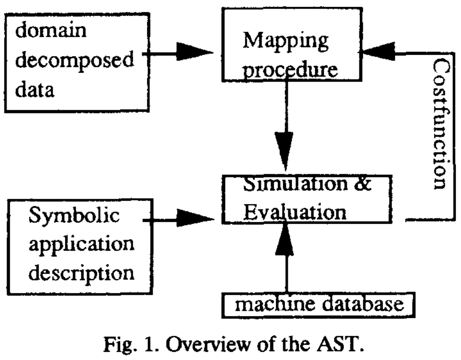

Fig. 1. Overview of the AST.

2.1. The machine database

A model that can represent (up to a predefined level of detail) the characteristics of (virtual) DMM has been developed. It is validated by means of timings on real machines (e.g. a Parsytec 512 T800 transputer platform [3]). For the development of the machine model the machine parameters that influence the performance of data parallel applications had to be identified:

(1) Communication parameters: e.g. hardware and virtual topologies, latencies, throughput, start-up time for a message and routing (allocating) processes.
(2) Processor parameters: e.g. time constants for addition, multiplication, division etc. for different numeric types and flow control performance (branch costs → pipeline efficiency).
(3) Memory parameters and caching: in the model it is assumed that enough memory is available. This in order to avoid modelling of I/O during execution. Memory speed (instruction and operand fetch) is incorporated in other parameters. Cache speeds are possible to time but cache size is impossible to be used without execution of target applications.
(4) 'Programming models': we consider SPMD programming models supported by PVM [6], EXPRESS [2] and MPI [1]. Of these message passing environments the most cost intensive primitives must be modelled: *spawn* (create a process), *synchronise* (synchronisation between processors), *send / receive* (message to/from neighbours), *multi-cast* (send a message to a group).
(5) I/O is not modelled.

2.2. The symbolic application description

We describe SPMD programs by the following functional hierarchy:

(1) statement block level,
(2) control flow level,
(3) data locality level.

(1) The time complexity of a so called statement block is given by cumulation of all the individual time complexities occurring in the block. For example an expression like:

```
a = a + b*c
```

has a time complexity of Tassign + Tmultiplication + Taddition. The corresponding actual time measures in such formal expressions are filled in by the machine database (the parameterised machine description).

(2) The control flow level introduces indeterminability into the time complexity description. In general the execution path taken, given a specific set of input parameters, is only determinable by means of explicit execution. We approach this level in a (quasi) statistic manner by describing the possibility of branching in some specific direction along the execution graph. Branch directions in *if...then...else* constructs are specified by probabilities ($P1, P2, \ldots, PN$ (where we assume the number of branching directions = N), whereas in *loop* constructs the number of unknown iterations is presented by a stochastic variable that behaves according to a (user defined) probability density function.

So a parameterised description on these two levels leads to a mixture of the time-complexities of basic statement blocks, probabilities and stochastic variables.

(3) The locality level describes the fact that some fraction of data is involved in communication and the remaining part is not. In case of static domain decomposition these fractions are constant. In case of programs where these fractions are forced to change dynamically it is necessary to introduce a stochastic description.

2.3. The mapping procedure

The mapping procedure aims to allocate data parallel processes on a multiprocessor system such that the time complexity function T is minimised. In other words, T is a cost function that has to be optimised using some generic optimisation algorithm such as simulated annealing or genetic algorithms. Assuming that an optimal workload balancing over the processes has already been achieved this implies that the communication effort between processors has to be minimised: neighbouring processes should be on (physically) neighbouring processors.

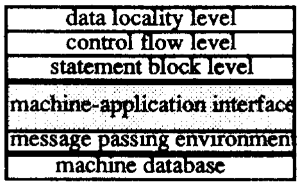

Fig. 2. Parameterised application mapped on parameterised machine.

Fig. 2 depicts the various layers out of which the simulation toolkit is constructed.

2.4. The simulation environment

The symbolic description of the program of interest consists of stochasts and statement block time complexities. A specific actualisation of all the parameters present will result in a number that represents the time consumption for the specific parameter actualisation.

It is quite probable that parameters that were initially described as stochasts turn out to be dependent on input parameters in a simple straightforward manner, e.g. loop boundaries. It is necessary to identify these parameters and replace them by their absolute behaviour instead of describing them in a stochastic manner. Furthermore, identification of dominant parameters in the time complexity formula and the actual program part they originate from is required. If necessary the program expert can supply background information on the behaviour of such parts. For example some loop boundary can turn out to have a well determinable stochastic behaviour dependent on several input parameters.

The symbolic description therefore has to allow for reversibility: given a certain parameter the program part from which it originates must be tractable.

3. Conclusions

The simulation methodology introduced offers the possibility to estimate the time consumption of SPMD applications on arbitrary parallel distributed memory machines. All tools are in their final stage of development and validation. Pilot experiments indicate that this technique can supply us with reasonable estimates of data parallel programs performance. We feel that this toolset can have a significant contribution in lowering the barrier for industry to use DMM. Currently the toolset is being validated by porting a very large complex crashworthiness code to a massively parallel machine.

Acknowledgements

Part of this research is funded by the Commission of European Communities within the Esprit Framework under project number: NB 6756 (CAMAS). The excellent contributions of Berry van Halderen and Arjan de Mes to the implementation of the toolset and furthermore the helpful suggestions of the CAMAS consortium (ESI-Paris, ESI-Germany, ACE-Netherlands, University of Southampton, FEGS-England, Parsytec-Aachen) are gratefully acknowledged.

References

[1] R. Lusky and B. Knighten, Minutes of the message passing interface forum, Dallas, TX (May 12–14, 1993).

[2] Parasoft, Express user guide version 3.2, 1992.

[3] Parsytec, Parsytec GC, technical summary version 1.0, 1991.

[4] J.F. de Ronde and P.M.A. Sloot, The application simulator, Technical report: CAMAS-TR-2.1.1.2 ESPRIT III, May 1993.

[5] P.M.A. Sloot and J. Reeve, The CAMAS workbench, Technical Report, CAMAS-TR-2.3.1 ESPRIT III, March 1993.

[6] V.S. Sundaram, PVM a framework for parallel distributed computing, *Concurrency and practice* 2(4) (Dec. 1990) 315–339.

J.F. de Ronde studied Computational Physics at the University of Amsterdam (graduated March 1992). Since September 1992 he has been PhD. student at the Department of Mathematics and Computer Science within the Parallel Scientific Computing and Simulation Group.