Automatic Performance Estimation Of SPMD Programs On MPP

J.F. de Ronde, B. van Halderen, A. de Mes, M. Beemster and P.M.A. Sloot
Parallel Scientific Computing and Simulation Group Department of Computer Science,
University of Amsterdam Kruislaan 403, 1098 SJ Amsterdam, The Netherlands.
email: peterslo@fwi.uva.nl, fax: +31 20 5257490, phone: +31 20 5257463

ABSTRACT

A methodology for the estimation of sequential and SPMD programs performance is presented. Performance metrics concerning the execution of a specific application on a target machine are derived using a machine parameterization and a symbolic application description. Both parameterizations can be obtained automatically using an integrated toolset: the *Performance Estimation Toolkit.* The toolset allows for the investigation of performance behaviour of applications and machines in hypothetical situations. A case study using the alpha-versions of the toolkit shows that the proposed performance model allows for prediction of tendencies of the execution time as well as absolute values.
(keywords: performance estimation, SPMD, tools)

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 suit parallel distributed memory or tightly coupled massively parallel (MPP) machines while it is not clear that the effort spent will pay off eventually. An environment that can predict the performance of SPMD (Single Program Multiple Data) data parallel applications on (MPP) 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 with which data parallel program performance can be predicted by means of parameterization of the distributed memory machine as well as the data parallel application. This technique is based on a static approach. The *Performance Estimation Toolkit* (PET) consists of several tools to obtain these parameterizations and to perform simulations using these formal machine and application descriptions (that is mapping the parameterized application to the parameterized machine).The PET is partially developed within an ESPRIT III project [1]. The structure of this paper is as follows: Section 2 describes the usage of the PET. The development status of the tools involved is described in section 3 . Section 4 presents a case study on a molecular dynamics benchmark. Finally section 5 will conclude on the results obtained so far and briefly shows the path for further development of the methodology and the PET.

2. THE PERFORMANCE ESTIMATION TOOLKIT (PET)

The Performance Estimation Toolkit is being developed with the goal to investigate performance behaviour of SPMD programs on parallel (tightly coupled or distributed memory) systems. In Figure I the synergy of the tools as they have been developed so far is depicted.
The PET consists of the following:
- A database that contains parameters describing machine characteristics that are of importance to execution time or temporal performance. These parameters can be actualized via measurements of elementary times on realistic machines [2], using results obtained with architecture simulation [3], or by handcrafted (hypothetical) machine parameters. The database tool offers the possiblity of adjusting the parameters in an interactive manner. Furthermore the topology of a multi-processor system can be defined.
- A tool that automatically can derive an abstract representation of SPMD code (specifically dedicated to Fortran 77 with message passing constructs) in terms of parameters that describe the characteristics of the code. The representation language is referred to as symbolic application description (SAD).
- A tool that allows investigation of the behaviour and optimization of SPMD programs performance with respect to various possible partitionings of the data-space of the problem under consideration. The development of this tool is in its initial phase.
- A simulator that integrates these tools to obtain an estimation of the performance behaviour of a data parallel application on a parallel architecture.

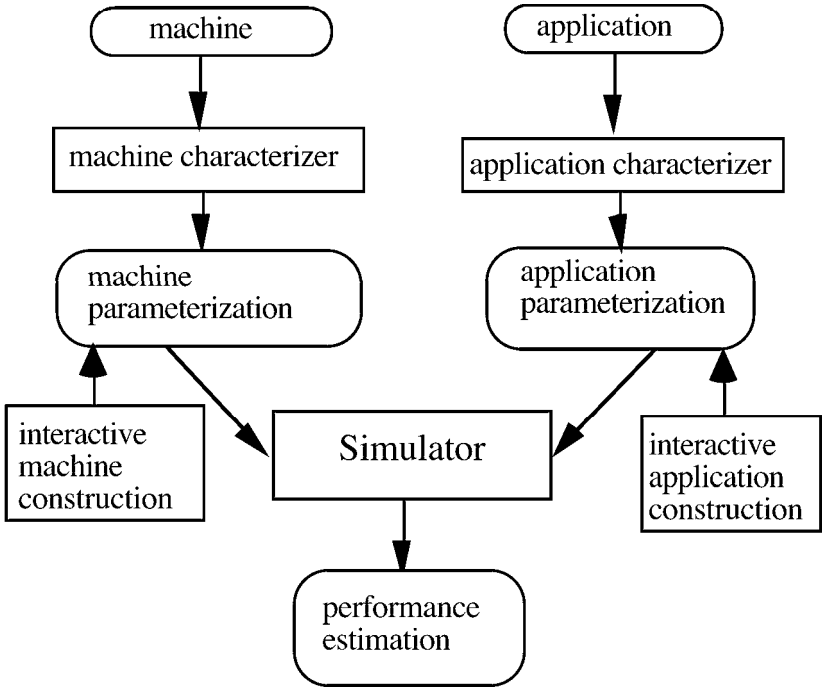


Figure I: synergy of the tools of the PET.

The toolset described above allows for investigation of the temporal performance (see [4]) behaviour of algorithms that can be expressed in terms of data parallel SPMD codes. If an implementation of a specific algorithm is already available, automatic parameterization of the program can be obtained.

The performance behaviour of the parameterized program can be studied under variation of the program parameters as well as machine parameters. The parameter values can be changed interactively.
When only a sequential program is available one can derive a parameterized description and can add parameters that mimick the behaviour of a possible parallelized version of the original code. In this manner the necessity of a full implementation of candidate algorithms for performance testing can be circumvented. In the next section the form of the parameterizations is discussed.

3. PARAMETERIZATIONS

3.1.1 The Machine Database

To obtain reliable performance estimations of an application on a modelled machine, the model must define the machine to a certain level of detail. The model can not be too detailed, because the aim is to estimate the performance using a high level description of the application, still it can not be too shallow because a reasonable estimation of the applications execution time is required.
Many machine characteristics, especially those due to recent development such as multi-level cache and branch prediction strategies, cannot be used effectively in the estimation without resorting to execution traces or low-level simulation of the hardware (see for example [5]). Therefore exact estimates are difficult to obtain in general. However work by Saavedra-Barrera et al. (see [6]) shows that the type of model that is proposed in this paper can provide realistic estimates. When estimates can not be exact at least tendencies of the performance behaviour with respect to variations in the parameter values can be obtained.

The model used for this project distinguishes the following sections:
1. The processor level. Average times are provided for the integer, real, double precision and complex numeric types on the most frequently used operations, such as addition, multiplication, division, power, assignment, compare and intrinsic functions. Times for flow control parameters are also present in the model.
2. The communications level. The hardware topology is specified together with the most often used virtual topologies (where possible). For all topologies latency and throughput are specified per link; when specifying virtual topologies, the timings include routing.
3. The programming model level. The primitives in the PVM, Express and MPI [7-9] models that are relevant for SPMD applications are modelled, e.g. send, receive, multicast and others. The execution behaviour of the procedure calls in these programming models is described in the machine model. Such a description is a composition (i.e. mathematical expression) of other parameters in the machine model.
The machine parameter 'memory size' is considered irrelevant (the machine has enough memory for the application), while memory speed is incorporated into the instruction timings. I/O is left out of the model due to irrelevance and unnecessary complexity.

It should be noted that compiler optimizations are also unpredictable. The model of the machine as well as the model of the application ignore the fact that a compiler might modify code to speedup execution.
The approach that is used to obtain time-averages for the various possible atomic statements is of great importance. The way that these times are approximated is subject of further research.

3.1.2 Status
The machine database has had its alpha-release. Still work has to be done on specification of the parameters that describe communication effort. Further the development of a user-interface as well as an interface to the simulator in order to allow for automatic mapping of the symbolic application description on the parameterized machine is important .

3.2 The symbolic application description
The metric that we use for characterisation of the performance of numerical SPMD applications as occurring in science and industry is execution time. In general these types of applications do consist of reasonably regular algorithms such as finite difference or finite element computations. The regularity of such systems ensures that a description of the program (sequential) will only depend on a handful of indeterminable control flow induced jumps. The so-called temporal performance can be derived from the original program in terms of machine specific parameters and parameters describing the control flow in the system by hand [see for example [10]). The tedious task of deriving such a description is met in our toolkit in an automatical manner.

We describe SPMD programs by the following functional hierarchy:
1) Statement block level
2) Control flow level
3) Data locality level

3.2.1 The statement block level
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 :

`a = a + b * c` : $Time-complexity = t_{assign} + t_{multiplication} + t_{addition}$

3.2.2 Control flow level
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. Branch directions are not statically determinable. In numerical applications fortunately the amount of constructs like `if..then..else` or `loop` that introduce nondeterminism into code is small compared to amount of calculations. The algorithms used in numerical applications can depend in their length of execution on for example the problem size in a clear manner. These indeterminable parameters are treated in a 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 P_1, P_2..,.P_N (where we assume the number of branching directions = N) whereas in `loop` constructs the number of unknown

iterations is presented by a variable that behaves according to a (user defined) function. Basically the variables described above can be stored in three classes:

1) Directly determinable: for example loop counts that directly depend on the problem size.
2) Indirectly determinable: for example through some simple backtracing of data dependencies the dependence on directly determinable variables can be derived.
3) Indeterminable (except through explicit execution): for example stop conditions. Such parameters are modelled using a stochastic approach. We thus can estimate the execution time of a sequential high level language such as Fortran 77 by means of the following symbolic formula:

$$SAD = \sum_{i=1}^{N} \prod_{m_i}^{M_i} \prod_{k_i}^{K_i} P_{k_i} X_{m_i} S[Block(i)]$$

Where N is the number of isolated statement blocks (containing no control flow characteristics whatsoever), P_{k_i} describes the k_i-th nested branch probability of the total of K_i branches in which *S[Block(i)]* is nested. Analogously X_{m_i} describes the loop-count of the m_i-th nested loop of a total of M_i loops in which *S[Block(i)]* is nested. *S[Block(i)]* is the time complexity of a statement block with label i .

3.2.3. Data locality level

The data 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 can change dynamically it is necessary to model this by means of a stochastic description. Development of the formalism in which this level can be expressed is part of future work. It will be done in conjunction with the mapping tool (see section 2).

3.2.4 Status of the F_2SAD compiler

The F_2SAD compiler for the automatic abstraction of symbolic application descriptions of Fortran 77 code has reached an alpha-version. Future developments of the tool are concerning the development of a user-interface, an interface to the simulator (map the abstract application on the abstract machine) and addition of features that allow for translation of sad level 3 (=communication)parameters.

4. A CASE STUDY : SEQUENTIAL MOLECULAR DYNAMICS

We have chosen the molecular dynamics (MD) benchmark from the GENESIS Benchmarks [2] as subject for a case study for first validation experiments of the alpha versions of the F_2SAD compiler and the machine database. The choice of this code is motivated by the fact that molecular dynamics is a field of interest within our group.

4.1 The md1 GENESIS Benchmark

The benchmark concerns a molecular dynamics simulation of a Lennard-Jones fluid with periodical boundary conditions and calculations of system properties in terms of reduced units (see for example [11]). The program contains the MD specific kernel + pre and post-processing of the benchmark timing measurements and contains in total approximately 1200 lines of Fortran 77 code. We are primarily interested in estimating the performance of the MD kernel (the most time consuming part).

4. 2 Results:

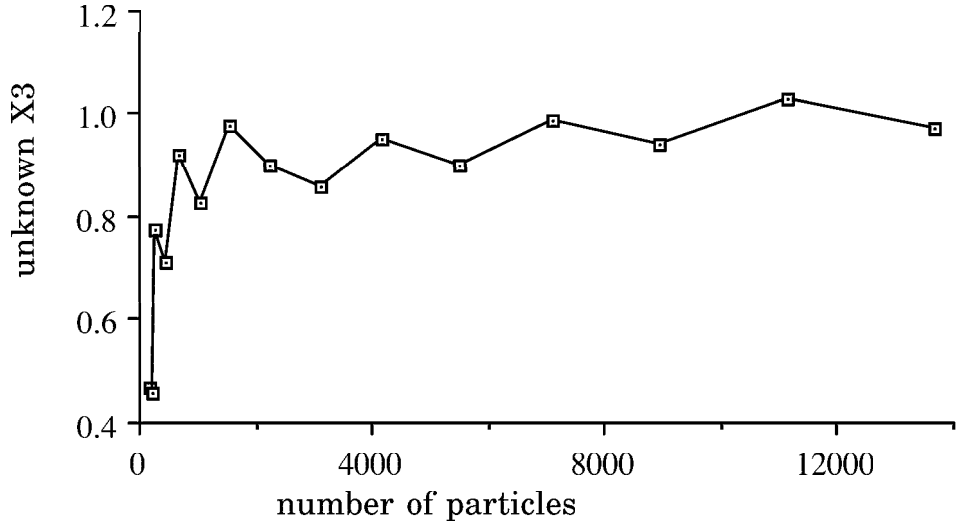
The most time consuming part within the benchmark can easily be identified. It consists of a "DO-loop" in which in every iteration the equations of motion of a system of "Lennard-Jones" particles are solved using the Verlet algorithm. In the SAD-formula of this program kernel 16 statically indeterminable parameters occur (9 loop counts and 7 probabilities). The rest of the parameters are simply determinable from the input set: problem size (=number of particles) etc... The unknown parameters have not been modelled stochastically yet but their behaviour as function of the problem size has been determined by means of line profiling. This approach was taken to be able to validate the models proposed above.

```
      DO 20 K = 0,MZ-1
        DO 21 I= MX-1,MX*MY-1,MX
          J=HEAD(I+K*MX*MY)
22        IE (J.NE.0) THEN
          NEDGE = NEDGE-1
          RX(NEDGE) = RX(J)-1.0D0
          RY(NEDGE) = RY(J)
          RZ(NEDGE) = RZ(J)
          ICELL = 1+IDINT((RY(NEDGE)*SFY+0.5D0)*CELLIY)*MX
     &          +IDINT((RZ(NEDGE)*SFZ+0.5D0)*CELLIZ)*MX*MY

           LIST(NEDGE)=HEAD(ICELL)
           HEAD(ICELL) = NEDGE

          J = LIST(J)      loopcount of DO20..20CONTINUE is referred to as X1
          GOTO 22
          ENDIF            loopcount of DO21..21CONTINUE  is referred as X2
21      CONTINUE           loopcount of 22IF(J.NE.0)..GOTO22 is referred as X3
20    CONTINUE
```

Figure II: kernel fragment of MD code

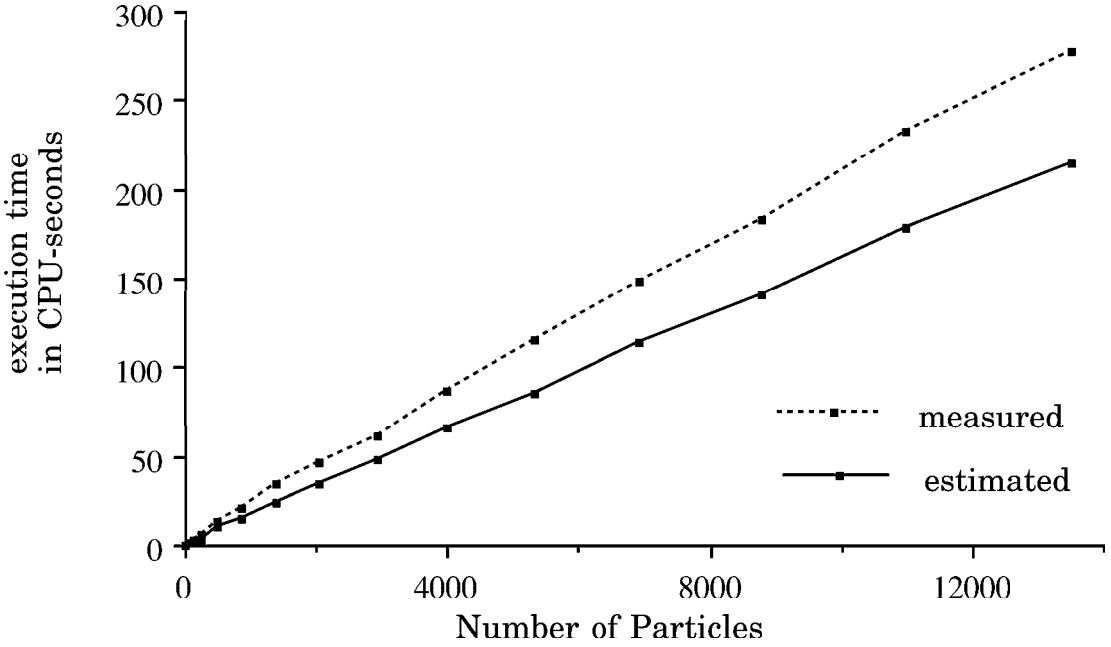
Figure II shows an example of a program fragment in this kernel. In this specific example a loop body is nested within 3 loops of which the outermost two loop counts are determinable statically (in advance of execution). The parameters determining the size of these loops (X1 and X2) depend in a simple manner on input parameters MX and MZ. The values for MX and MZ are determinable from a small set of input parameters. For problem sizes between 0 and 15000 particles the unknown parameter X3 shows a dependence on the problem size as depicted in Figure III. For stochastic simulation purposes this figure inferres that for large problem sizes the value of X3 is approximately equal to 1. For all the unknown parameters occurring in the SAD formula the problem size dependence has been determined in the same manner.

Figure III: Unknown X3 vs. number of particles



We have specifically investigated the execution time behaviour of the program on a SUN-SPARC-2 workstation. The measured as well as the predicted CPU-time consumption of the program kernel as a function of the problem size are shown in Figure IV. The actual execution time is generally about 20 % larger than the estimated time.

Figure IV: estimated and real execution times vs. number of particles



Clearly the tendencies shown by the program execution and the time-complexity formula are identical. Not only the overall tendency (linear dependence on the problem size) but also the presence of structure is predicted correctly. This reinforces the confidence in the modelling power of the

proposed abstractions. To get better agreement with real time measures obviously the timing methods have to be more accurate. Refinement of the timing modules is part of future work.

Future developments
The next phase in the development of the toolset will be dealing with tests of the machine model for parallel architectures. Furthermore the symbolic application description will be expanded within the F_2SAD compiler for SPMD codes. The data locality level will be developed in conjunction with the mapping tool.

5. CONCLUSIONS
The performance characterisation method presented here shows that estimation of performance trends of sequential numerical programs is feasible. This allows for speculation on the suitability of candidate algorithms for sequential systems. It allows for performance estimation of codes of which the time-complexity formula cannot be derived by hand (just because the codes are too large). Therefore it can be very helpful in migration of large sequential codes to parallel systems. The effect that parameter changes have on the performance behaviour can automatically be investigated. The development of a third level of functionality within the symbolic application description and the direct applicability of the machine parameterization to this level will be of crucial importance to the suitability of the PET to data parallel SPMD codes. This will be part of future developments.

Acknowledgements:
Part of this research is funded by the Commission of European Communities within the Esprit Framework under project number: NB 6756. We gratefully acknowledge the contribution of Alistair Dunlop from the University of Southampton who has supplied us with a set of benchmarking routines for atomic Fortran 77 statements.

REFERENCES
1. P.M.A. Sloot and J. Reeve. 1993. "The CAMAS Workbench". Technical Report: CAMAS-TR-2.1.1.2 ESPRIT III. (March).
2. A.J.G. Hey. The GENESIS distributed memory benchmarks. Parallel Computing, 17(10-4).
3.H. Muller. Simulating computer architectures, thesis. February 1993. ISBN 90-800769-4-5.
4.R. Hockney. A framework for benchmark performance analysis. Supercomputer March 1992.
5. K. Hwang. Advanced computer architecture, 1993, ISBN 0-07-113342-9.
6. R.H. Saavedra-Barrera et al. Machine characterization based on an abstract High Level Language Machine. IEEE Transactions on computers, vol. 38 no.12. December 1989.
7. R. Lusky and B. Knighten.1993. "Minutes of the Message Passing Interface Forum" Dallas, Texas. May 12 -14 1993.
8. Parasoft 1992. "Express User Guide version 3.2"
9.V.S. Sundaram. 1990. "PVM a framework for parallel distributed computing " Concurrency and practice, vol.2(4) (December): 315-339
10.A.G. Hoekstra, P.M.A. Sloot et al. Time complexity of a Parallel Conjugate Gradient Solver for Light scattering simulations. Technical Report CS-92-06. June 1992. Department of Computer Systems, University of Amsterdam.
11.M.P. Allen and D.J. Tildesley, Computer simulation of liquids, Clarendon Press. Oxford 1987.