SAD progress report

van Halderen, A.W.; de Ronde, J.F.; Beemster, M.; Sloot, P.M.A.

Publication date
1994

Commission of the European Communities

****

ESPRIT III

PROJECT NB 6756

****

CAMAS

COMPUTER AIDED MIGRATION OF APPLICATIONS SYSTEM

****

CAMAS-TR-2.2.4.4
SAD progress report

****

Date: October 1994 — Review 4.0

ACE - Univ. of Amsterdam - ESI SA - ESI GmbH - FEGS - PARSYTEC - Univ. of Southampton

Authors: Berry A.W. van Halderen
Jan de Ronde
Marcel Beemster
P.M.A Sloot

October, 1994
University of Amsterdam,
Faculty of Mathematics and Computer Science
Parallel Scientific Computing and Simulation group
Netherlands

1 Introduction

Within this report the work that is done within the CAMAS subtask 2.2.4 for the period March 15 till September 14 is described.

The work is divided out over the following subsections:

- Refinement and debugging of the f2sad tool;
- Automatic instantiation of parameters using line profiles;
- PAM-Crash core code to SAD (levels 1 and 2) translation;
- definition of functionalities of the SAD3 level;
- SAD level 3 simulation;

The work spent on the items mentioned is discussed in some detail below. Furthermore the status of the SAD work will be explained. Also the work that is planned for the next six monthly period is pointed out.

2 Refinement and Debugging effort

Several deficiencies were revealed in the f2sad tool in larger experiments with Fortran applications. One of the major problems was the amount of memory necessary to process a large Fortran application such as a reduced version of the PAM-Crash code (30 percent of the original). Over 175 Mb was needed to process the Fortran code. This proved to be a major problem since many machines have a limited amount of 64 Mb of virtual memory.

Much effort has gone into limiting the amount of memory used by the program, which also gives a better runtime of the program. The current status is that the program uses about 45 Mb of memory for the PAMcrash core code, of which 30 Mb is actively used. A machine with 20 Mb of real memory or more is therefore advisable for the run time performance of the tool.

The problem of the extensive memory usage has several causes. Firstly there is the problem that for each token in the input a medium sized structure is kept in memory, this in order to be able to trace all parameters in the output back to the position in the input source. Secondly, much data was replicated since it was reused in other parts of the program. The current implementation reuses the data whenever it can. The third major factor is the way in which the memory is allocated. Because the tool builds a huge datastructure, a fast amount of small structures is kept in memory. Standard memory allocation libraries work well but for such a large amount of small structures it makes sense to try to save upon those few bytes per structure by using a different memory allocation scheme.

3 Automatic instantiation of parameters using line profiles

The SAD2 formula is a time complexity formula with two sets of parameters which need to be filled in. First is the set of machine parameters, which is kept abstract to allow an interactive experimentation with different architectures. The second is the set of parameters derived from if-statements and loops in the program. The goal is to provide (possibly probabilistic) functions for these parameters and simulate the program with different settings.

The first step however, needs to be to provide numbers for the formula for example test runs. In this way, we can test the formula for its correctness, completeness and how well it models the execution time of the input program. Of course these numbers can be supplied by hand, but this is a tedious work and requires someone to have detailed knowledge of the program.

An automatic way in which the parameters can be filled in using some easy to generate process in needed. Most compilers can compile programs with additional data tagged to all basic blocks so that when the program is run a profile of the execution is generated. There are two types of profiles; time profiles and line profiles (sometimes also called test coverage). Time profiles are not suitable because they are very inexact (only reliable data for long executing basic blocks) and mostly because they are machine dependent.

Line profiles can provide enough information in some cases. The data we want to retrieve from such a line profile is how many times a condition in an if-statement was evaluated to true or how many times a loop iterated. This can simply be evaluated by dividing the number of times a loop or then part of an if-statement is entered by a statement just before the if-statement or loop. This data is directly available in the line profile.

There are however some cases which need special attention; If the statement just before or in the loop or if-statement is in a different subroutine or function, we may end up with the wrong number because the subroutine or function may also be called from somewhere else in the program.

There is also another problem, having to do with a process earlier in the f2sad tool. The f2sad tool removes go to's by rewriting the program. In this process it is possible that if-statements or loops are created from multiple conditions in the original program. In this case we end up with a condition which is not directly related to a condition in the original program, but rather is a combination of *and*, *or* and *not* functions and conditions in the original program.

These problems have been addressed and were partly resolved. The version of the automatic parameter actualization work well in e.g. the IPS examples in appendix B. However it is not possible to do so for every program yet, for example the PAM-Crash core code.

4 PAM-Crash core code to SAD translation

A core sample which contained about 30 percent of the original PAM-Crash source was selected by ESI for the evaluation of the f2sad tool. Evaluation by ESI of an estimation by f2sad showed the problems with the memory behaviour of f2sad. Though these have now been resolved there remains the problem addressed in the previous section which is still unresolved for the performance evaluation of PAM-Crash.

5 Functionalities of SAD3

SAD levels 1 and 2 are used to describe the functionality of general sequential programs written in an imperative programming language such as Fortran. The f2sad tool can be used to abstract a symbolic description from a Fortran 77 program. This sequential time-complexity description can be used to estimate sequential programs performance.

SAD level 3 introduces performance behaviour due to the presence of communication constructs as appearing in general data-parallel SPMD programs. We consider as basic message types that can be used:

- Send (synchronous or asynchronous and blocking or non-blocking)
- Send to group (asynchronous and blocking or non-blocking)
- Receive (blocking or non-blocking)
- Receive from group (blocking or non-blocking)

5.1 Message patterns

In general an SPMD program will consists of separate communication and calculation phases that alternate. Several typical communication patterns may occur. In principal though we distinguish three basic communication strategies.

- Static Communication patterns
- Quasi Static Communication patterns
- Dynamic Communication patterns

The communication patterns are considered on the level of process ID's that send data to one another or to a group. How these processes are mapped on the physical processor topology is handled by an underlying layer that depends on the machine definition within Parasol I.

5.1.1 Static Communication patterns

The characteristics of a static communication pattern are that the message sizes are constant every time a specific message is send from processor A to B. Also the destinations remain constant throughout the execution of the parallel program.

For example in PAM Crash initially the input mesh is decomposed. Nodal points that are shared by two or more processors remain the same throughout execution. Consequently the message pattern for sending forces or velocities along "boundary" nodal points remains static also.

5.1.2 Quasi Static Communication patterns

For quasi static communication patterns applies that the messages sizes do not have to be constant, but the destinations remain constant throughout the execution of the parallel program.

5.1.3 Dynamic Communication patterns

In a dynamic communication pattern neither the message sizes nor the destinations are constant. This communication pattern is the most complex and unpredictable.

6 SAD level 3 simulation

SAD level 3 will utilize simulation to mimic performance behaviour of SPMD applications. Phases within the SPMD application in which number crunching takes place is simulated using the functionality of SAD levels 1 and 2. The communication phases that occur are mimiced using simulation.

We consider applications that exhibit a certain behaviour on a network of processors. This behaviour can be subdivided into the classes *application* parameters and *machine* parameters. Which processor communicates with which other processor, at which point in time and how much data is transmitted are parameters determined by the application. The application parameters dictate what impact the machine parameters have. Which processors communicate determines the locality of messages in the computernetwork. In this way parameters like congestion are influenced. Send and receive times are related to message sizes (application parameter) in a linear fashion. Also the send and receive times depend on the setup and transmission times (machine parameters). The message size defines the effect of the machine paramters on the total performance behaviour.

So there is a link from the application parameters to the machine parameters, where the application parameters determine at which point in the execution time the machine

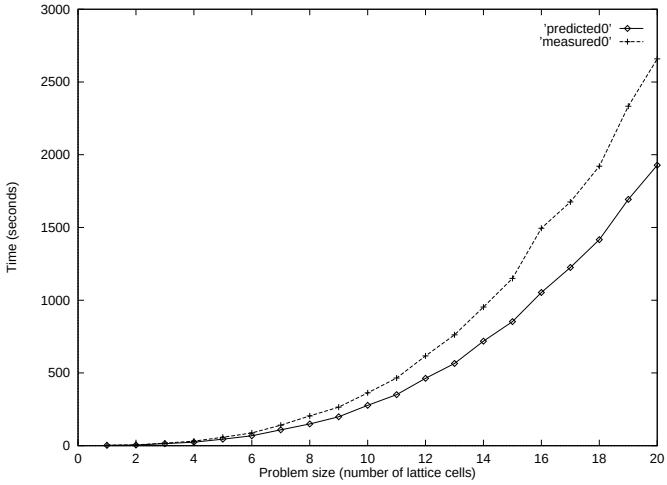
parameters take their effect. For SPMD applications there is no link back. Typical SPMD applications do not adapt for send/receive delays; the communication pattern remains the same. This is not the case for all parallel application types. Many server–client applications have time-out influences and transmission control flow. Note that synchronization cost, apart from the messages that need to be send, is an application parameter.

A simulation on the basis of the behaviour instantiated with the hardware parameters is therefore obvious. For this simulation to be fast enough in an interactive environment it needs to be done on a course grained basis; simulation on network events only and the interleaving computation parts of the program are simulated using the simple formula of the SAD2 level.

A simulation of the processors and the network on the level of events is planned. Because of asynchronous communication it is possible that a processor node is involved in multiple network transactions and can still be performing computation. The simulator must therefore manage a queue of running actions on each processor.

The abstraction level of events is quite high. Computation is simulated as a period of time in which the process cannot issue additional events. Synchronous communication is simulated by invoking a load on the network and querying the machine database for the time the operation will take, given the current total network load. The network load is only taken into account for other queries to the machine database during the time the network transaction is in progress. The difference between synchronous and asynchronous communication is that in the asynchronous communication the process will immediately generate new events, while in synchronous communication the process waits for the current event.

A Results of f2sad/parasol on MD1 (Genesis Benchmark)



problem size	measured time	predicted time
1	3.04	2.55
2	5.97	4.67
3	17.62	14.22
4	30.94	23.74
5	58.11	45.01
6	87.87	68.31
7	141.01	108.85
8	204.74	149.21
9	265.05	198.63
10	363.04	277.28
11	465.32	350.49
12	617.03	463.36
13	762.07	565.03
14	952.90	718.28
15	1149.61	853.07
16	1495.07	1052.88
17	1675.49	1225.46
18	1921.28	1416.26
19	2333.41	1693.02
20	2659.20	1928.34

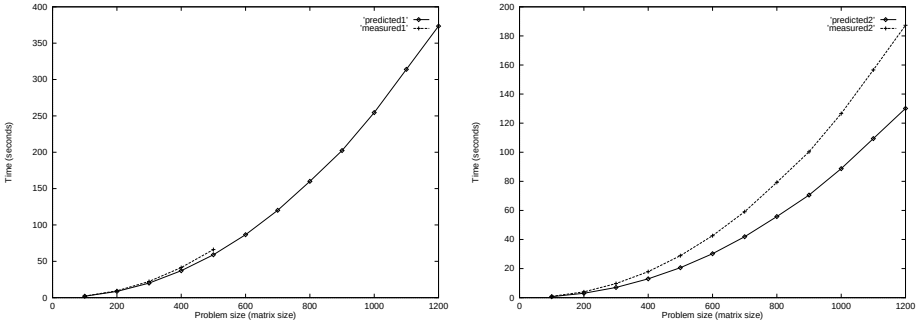
Figure 1: Measured and Predicted execution times for the MD1 benchmark program from the Genesis benchmark suite

The MD1 benchmark has extensively been used to verify the operation of f2sad. In the figure 1 the results of the measured execution time and the predicted value by f2sad is shown. There is a 27 percent difference between the expected and estimated value. This difference can be ascribed to the fact that in the f2sad model the data cache is not modeled; the data cache is considered near perfect. In other experiments we can see that this can give substantial performance loss, even greater than 20 percent.

Modeling the data cache is clearly one of the attention points to be covered.

The trends are however clearly the same, even the irregularity of the real performance behaviour is shown by the prediction. The irregular behaviour is caused by a call to the function `MIN()` which determines a loop variable.

B Results of f2sad/parasol on IPS

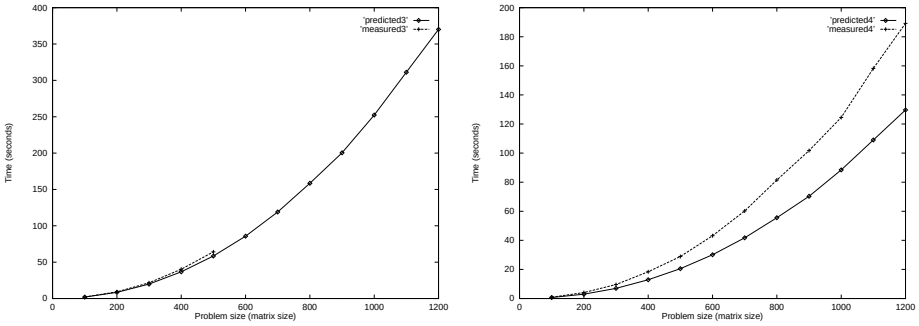


	GCel			Sun		
n	Predicted	Measured	Error	Predicted	Measured	Error
100	2.004	2.175	0.078	0.706	0.900	0.216
200	8.593	9.473	0.093	3.011	4.000	0.247
300	20.047	22.269	0.100	7.011	9.700	0.277
400	37.073	41.424	0.150	12.951	17.900	0.277
500	59.005	66.137	0.178	20.600	28.900	0.287
600	86.602			30.221	42.600	0.291
700	120.164			41.916	59.100	0.291
800	159.990			55.789	79.300	0.297
900	202.303			70.536	100.300	0.297
1000	254.605			88.747	126.600	0.299
1100	313.970			109.413	156.600	0.301
1200	373.465			130.137	187.300	0.305

Figure 2: Predicted and Measured execution times of the IPS program (*grid* version) for both the Parsytec GCel (program was executed on a single transputer) and a Sun Sparc Lx workstation

Within the IPS subtask two different parallel implementations have been realized, one using a grid communication pattern and one using a ring. The time complexity of the calculation kernel of both implementations have been derived using f2sad, for which the results are shown in figures 2 and 3.

In these two experiments with two different implementations (grid and ring) it clearly can be seen that a prediction for a machine with a simpler memory model (the Parsytec GCel builds upon t800's) is far more accurate that a complicated memory model (Sun workstation) at least for the points we were able to measure on the Parsytec GCel. We have identified the deviation in the predictions for the Sun to be due to the effects of caching.



	GCel			Sun		
n	Predicted	Measured	Error	Predicted	Measured	Error
100	1.983	2.122	0.066	0.703	0.900	0.219
200	8.504	9.223	0.078	2.999	4.100	0.269
300	19.844	21.648	0.083	6.983	9.600	0.273
400	36.710	40.236	0.088	12.901	18.400	0.299
500	58.434	64.208	0.090	20.520	28.900	0.290
600	85.777			30.105	43.200	0.303
700	119.038			41.757	60.200	0.306
800	158.517			55.581	81.500	0.318
900	200.436			70.271	101.700	0.309
1000	252.296			88.420	124.500	0.300
1100	311.174			109.016	158.200	0.311
1200	370.134			129.664	189.200	0.315

Figure 3: Predicted and Measured execution times of the IPS program (*ring* version) for both the Parsytec GCel (program was executed on a single transputer) and a Sun Sparc Lx workstation