Dynamically Adapting the Degree of Parallelism with Reflexive Programs

Niels Reimer (reimer@ira.uka.de) [*]
Stefan U. Hänßgen (haenssgen@ira.uka.de)
Walter F. Tichy (tichy@ira.uka.de)

IPD, Fakultät für Informatik, Universität Karlsruhe, Germany

Abstract. In this paper we present a new method for achieving a higher cost–efficiency on parallel computers. We insert routines into a program which detect the amount of computational work without using problem–specific parameters and adapt the number of used CPUs at runtime under given speedup/efficiency constraints. Several user–tunable strategies for selecting the number of processors are presented and compared. The modularity of this approach and its application–independence permit a general use on parallel computers with a scalable degree of parallelism.

1 Introduction

Programs on parallel computers usually use all available processors. This is a waste of resources if the load is not evenly distributed or the amount of work is too small to justify a further partitioning. Our goal is to reduce the costs[1] by adapting the number of used processors dynamically according to the current load. In order to do so, the program observes its parallel routines and controls the degree of parallelism individually at runtime. This leads to an implicit load balancing because the load is automatically distributed evenly with each call of the parallel routines. In this paper we therefore do not deal with load balancing algorithms which are described e.g. in [1]. The search for related work in the area of reflexive programs or adapting the amount of parallel resources showed no exploitable references.

In our work we improved the cost–efficiency of a parallelised molecular dynamics (MD) simulation program, but any application with frequent calls of parallel subroutines can benefit from this method. We define the cost–efficiency as the sequential-to-parallel cost ratio.

2 Molecular Dynamics as an Irregular Problem

The importance of MD in the area of biological, chemical and medical research is increasing. Simulations with larger molecules require a computational performance which necessiates the use of parallel computers. We first introduce the principles of MD briefly and show that it is an example of an irregular problem.

[*] This research was performed at the EMBL (European Molecular Biology Laboratory) Heidelberg in co-operation with the University of Karlsruhe.

[1] The cost is the sum of all used CPU seconds over all used processors.

MD simulations calculate the interactions of particles (molecules/atoms) in order to derive geometrical and structural properties of molecules. For simulation purpose, the timescale is split into regular steps. At each timestep, the interactions of the particles are calculated and used to determine the position of each particle in the next timestep. To be physically accurate, ideally the interaction of each pair of particles has to be considered. Since the forces decay at least with the square of the particles' distance, one can speed up calculation by ignoring the interactions to particles beyond a certain cutoff radius from a particle. However, the relevant particle pairs then have to be managed in addition to the force calculations. Most MD–programs perform the following cycle of steps:

1. **Generate pairlist** with all relevant pairs of particles.
2. **Calculate forces** according to the pairlists and the particle data.
3. **Update particle data** by applying the forces to the particles.

Parallelisation approaches of MD–programs usually associate particle data to processors. The movement of the particles then leads to unpredictable irregular communication patterns among the processors, hence parallel MD is an irregular problem. The corresponding pairlist routine shows an increase of work load with the square of the number of particles. The work load of the force calculation routines increases only linearily with the number of particles because the maximum number of particles inside a cutoff radius sphere is limited. Further information about MD can be found in [3].

To allow the use of more specialised force calculation routines, the MD–program ARGOS [8] distinguishes between two classes of particles:

1. **Solvent particles** which are all equal (generally water molecules).
2. **Solute particles** which are normally the atoms of the examined chemical compounds, e.g., proteins.

As a result, there are three pairlist and three force calculation routines, each with its individual amount of work. For the pairlist routines this depends on the number of particles, which is constant within each simulation. The work load for the force calculation depends on the density of particles, which can vary to a large extent during simulation. Therefore, the most cost–efficient number of processors to use for each routine is different and variable.

Different approaches of the parallelisation of a MD–program are described in the work of Hanxleden [2]. As part of our work, we parallelised the sequential MD–program "ARGOS" [7], [8] on a SGI Power Challenge, a shared memory machine with 16 R8000 processors [6].

3 Dynamically Adapting the Degree of Parallelism

Traditional load balancing methods redistribute the amount of work among all processors of the parallel computer at certain time intervals or on demand in order to reduce the number of idle processors. This leads to a higher cost–efficiency and a shorter runtime, provided that there is sufficient work for each processor. However, with powerful processors suitable for coarse–grained parallelisation this is often not the case since larger pieces of work cannot be distributed evenly. It is then advantageous to adapt the number of used processors to the program's resource consumption to offer each processor a suitable amount of work. The unused processors of this parallel machine (or of a cluster of workstations) are not wasted since they are available to all other users. The cost function therefore only takes the CPU–seconds of the used processors into account.

If the parallelised program contains sequential parts and several parallel routines, each of these routines is likely to require a different number of processors to reach an appropriate load level. Furthermore, this number can change during runtime. All of these problems occur in the parallelised version of ARGOS. Most computation time is consumed by the pairlist generation and the force calculation, hence it is sufficient to parallelise only these particular routines.

The idea is to estimate the load produced by each parallel routine and adapt the number of assigned processors accordingly. Thus, the program observes its own behaviour, which is why we call it reflexive. The load determination should be problem– and machine–independent. The convincing approach is to measure the time spent inside the parallel part. The requirement of adapting each parallel routine individually can be satisfied using a separate adaptation for each routine. This also leads to more modular code. Each parallel routine is therefore encaspulated in a preroutine which sequentially executes the following tasks:

1. Set the number of processors for the parallel routine
2. Start the timing
3. Invoke the parallel routine
4. Stop the timing
5. Calculate data for the setting of the number at the next invocation

The determination of the number of processors at the next invocation is based upon calculations of the timings, and is done according to a strategy described later. Figure 1 represents the encapsulation and the structure of a preroutine.

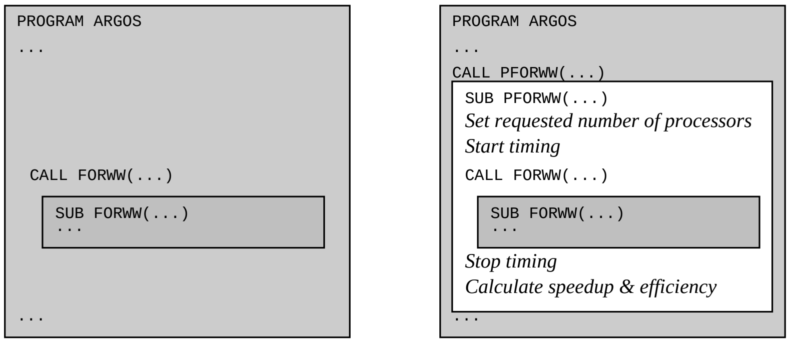


Fig. 1. Structure of the program with and without preroutine. The force calculation routine FORWW is executed in parallel.

With this approach, we gain a dynamic adaptation at runtime and a reflexive behaviour of the program without using problem–specific data. This implies the possibility to combine this approach with any parallel application.

4 Adaptation Strategies

In this paper we will consider three strategies in detail[2]. The *speedup–driven incremental search strategy (SISS)* works in the following way: The initial setting is one processor to determine the "sequential" execution time of the controlled routine. On return from the parallel routine call, the timing is taken and speedup and efficiency are calculated. With every subsequent invocation, the number of processors is increased by one if the measured efficiency rate is above and not too close to a fixed threshold (e.g. the rate is 5% above the threshold EFFTHR = 50%). If it is inside this region, the number remains fixed to minimise the number of threshold violations. If it is below the threshold, the number is decreased by one. The next timing results in a new efficiency rate and so on. After a certain number of invocations, the strategy calibrates itself by repeating the one-processor run to keep the "sequential" execution time up-to-date for the next calculations. Once the optimum is reached, the number of processors oscillates or remains temporarily fixed, which guarantees that the average efficiency is close to the threshold. An oscillation itself causes no additional costs since the number has to be set anyway at each invocation of the parallel routine.

An incremental search runs the risk of finding only a local mimimum. This problem was not encountered in the parallel version of ARGOS, but may occur with other applications.

To avoid this, a global search strategy can be used which splits the adaptation process into three phases: 1) a global search over all possible processor numbers is performed, 2) the best setting with the highest speedup is chosen under the restriction that the efficiency has to be above the threshold value. This feature gives the strategy its name: *speedup–driven global search strategy (SGSS)*. 3) this setting is kept fixed for a certain number of invocations. After this, the strategy starts again with the first phase (calibration) to gather the data.

A variation of this strategy is derived by changing the optimisation goal. This last strategy chooses the setting with the highest efficiency under the constraint that it must guarantee a speedup of at least a certain threshold value. This strategy is therefore called *efficiency–driven global search strategy (EGSS)*.

The indicated threshold values of the strategies are user-adjustable parameters. Modularity also allows the use of different parameters and strategies for each parallel routine to accomodate special requirements, e.g., routines with a stable work load can afford longer durations of the third phase.

Global searches for the optimum processor number are not prohibitive since the Power Challenge only has 16 processors. That means the first 16 invocations of the parallel routine are used to collect the timing data for each setting. On parallel computers with significantly more processors other scanning approaches will be necessary.

5 Results

Simulations of scenarios in which the pairlist lengths change, especially all denaturation simulations, challenge the adaptation algorithms with a dynamically

[2] For a closer study of more strategies we refer the reader to [5].

changing work load. Figure 2 shows the evolution of the pairlist lengths obtained from a denaturation simulation of a large protein (*myoglobin*) in water.

To reduce the comparison of the strategies to the highlights, we here present only the results for the force calculation of the solvent–solvent interactions because their work load changes most dramatically. The adaptation of routines with a more stable work load are equally accurate for all strategies. Figures 3 and 4 show the numbers of processors used for the force calculation of the solvent–solvent interactions using the SISS and the SGSS.

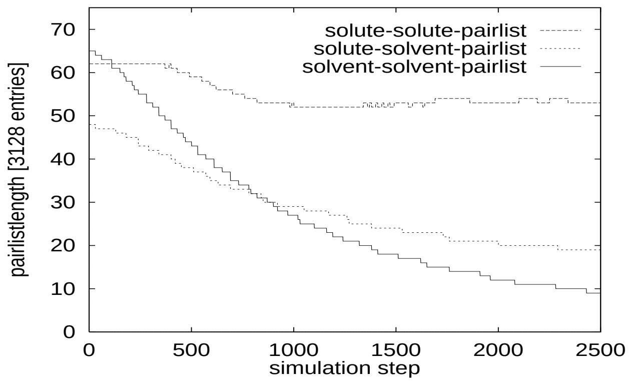
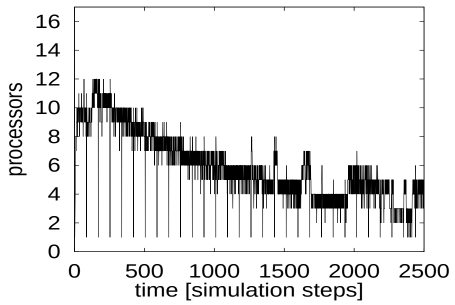


Fig. 2. Evolution of the three pairlist lengths during the progress of a denaturation simulation of myoglobin in water.



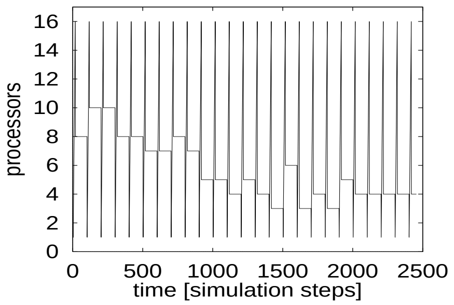


Fig. 3. Assigned processors for the force calculation of the solvent–solvent interactions using SISS

Fig. 4. Assigned processors for the force calculation of the solvent–solvent interactions using SGSS

The requested numbers of processors follow ideally the same trend as the pairlist lengths. The comparision of the curves in figure 3 and 4 shows that the results of the incremental search correspond to those of the global search, which demonstrates that local minima have no effect here.

Table 1 shows the runtimes and the costs obtained from the denaturation simulation to compare the different strategies with the commonly used alternative (*16 P*) which is to use always the maximum available number of processors (here 16).

These numbers demonstrate the potential of self–adapting reflexive programs to reduce the costs while maintaining a reasonable speedup.

Table 1. Runtimes and costs of the different strategies and the 16–processor alternative

	SISS	SGSS	EGSS	16 P
Runtime [sec]	3831	3436	4220	2886
Cost [CPUsec]	20817	23038	18626	40508

6 Conclusion and Future Work

Reflexive programs that dynamically adapt their degree of parallelism show that a problem–independent self–control mechanism can be used to achieve a higher cost–efficiency. The mechanism presented considers runtimes and thus automatically takes both problem size variations and machine–specific influences into account. This is crucial because traditional load balancing algorithms disregard these aspects. Furthermore, our approach is not restricted to MD–programs but can be used with any parallelised application that allows timings of the parallel executed parts.

Still, there are many open questions related to this work. What advantages and disadvantages would the use of problem–specific data imply? How can a compiler for parallel machines automatically benefit from this method? Will (heterogenous) workstation clusters of a PVM environment [4] behave with a similar "linear" and smooth scaling like the SGI Power Challenge, or will this require other strategies? These questions are a matter of further research.

7 Acknowledgements

We thank Dr. T. P. Straatsma for use of the ARGOS program and Prof. Dr. W. F. Tichy for support at the University of Karlsruhe. This work was performed in collaboration with the group of Dr. R. C. Wade at the European Molecular Biology Laboratory (EMBL) at Heidelberg and supported in part by the EU ASLI Supercomputing Resource for Molecular Biology (Contract ERBCHGECT940062).

References

1. I. Foster, *Designing and Building Parallel Programs*, Addison–Wesley, 1995, ISBN: 0-201-57594-9, http://www.hensa.ac.uk/parallel/books/addison–wesley/index.html
2. R. v. Hanxleden, T. W. Clark, J. A. McCammon, L. R. Scott, *Parallelization Strategies for a Molecular Dynamics Program*, Intel Technology Focus Conf. Proc., 1992
3. J. A. McCammon und S. C. Harvey, *Dynamics of proteins and nucleic acids*, Cambridge University Press 1987, ISBN: 0-521-35654-0
4. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. S. Sunderam, *A Users' Guide and Tutorial for Network Parallel Computing*, MIT Press, November 1994, ISBN: 0-262-57108-0, http://www.hensa.ac.uk/parallel/books/mit/pvm
5. N. Reimer, *Dynamische Einstellung des Parallelitätsgrades mit reflexiven Programmen*, `reimer@ira.uka.de`, University of Karlsruhe, January 1996
6. Silicon Graphics Inc., *Power Challenge Technical Report*, SGI 2011 Northern Shoreline Boulevard, Mountain View, CA 94039-7311, 1994
7. T. P. Straatsma, *ARGOS Reference Manual*, `tp_straatsma@pnl.gov`, 1994
8. T. P. Straatsma, J. A. McCammon, *ARGOS, a vectorized general molecular dynamics program*, Journal of Computational Chemistry II(8): 943-951, 1990