CONF-970477--2

Performance Prediction for Complex Parallel Applications

Jürgen Brehm, University of Hannover
Institut für Rechnerstrukturen und Betriebssysteme,
Lange Laube 3, 30159 Hannover, Germany
brehm@irb.uni-hannover.de

Patrick H. Worley, Oak Ridge National Laboratory
Mathematical Sciences Section
P. O. Box 2008, Oak Ridge, TN 37831-6367
worleyph@ornl.gov

Abstract:

Today's massively parallel machines are typically message-passing systems consisting of hundreds or thousands of processors. Implementing parallel applications efficiently in this environment is a challenging task, and poor parallel design decisions can be expensive to correct. Tools and techniques that allow the fast and accurate evaluation of different parallelization strategies would significantly improve the productivity of application developers and increase throughput on parallel architectures.

This paper investigates one of the major issues in building tools to compare parallelization strategies: determining what type of performance models of the application code and of the computer system are sufficient for a fast and accurate comparison of different strategies. The paper is built around a case study employing the Performance Prediction Tool (PerPreT) to predict performance of the Parallel Spectral Transform Shallow Water Model code (PSTSWM) on the Intel Paragon.

1. Introduction

Advances in microprocessor technology and interconnection networks have made it possible to construct parallel systems with a large number of processors (e.g., Cray Research T3D, IBM SP2, Intel Paragon, workstation networks running PVM). Unfortunately, the application programs developed for conventional sequential systems or for pipelined supercomputers do not automatically run efficiently on these systems. There are few tools to support the development of parallel programs, and the performance of parallel programs is strongly dependent on the parallel programming skills of the application developer.

Before writing a program, the developer must identify a parallelization strategy. In most cases there are many options for distributing the data and tasks onto the processors. These options often have widely varying performance characteristics that are functions of numerous system and program parameters, and it can be difficult to predict *a priori* which options are best. Accurate prediction of the performance trade-offs of alternative strategies and of how the performance will change as program parameters change would greatly benefit program developers.

Several approaches for the modeling of parallel systems have been presented that use Markov models or Petri nets [8], [11], [12]. Unfortunately, it is difficult to apply these approaches to massively parallel systems or complex parallel applications:

- The graphical representation required by these approaches is very complex for systems with hundreds or thousands of processors.
- The parallel application description required is very detailed.
- The resulting systems of equations defining the models are large and expensive to solve.

Applications for massively parallel systems typically use the single program multiple data (SPMD) programming model and are loosely synchronous [3]. For such programs, simpler modeling techniques utilizing algebraic abstractions of the application and computer system can often be used without a significant loss of accuracy [2]. These techniques make it feasible to model architectures with thousands of processors, and the resulting models can be evaluated quickly. Recent research utilizing algebraic performance models includes [4], [9], and [10]. These papers focus on tools or methodologies, many of them language or system specific, that automatically generate performance models from source code and user input. We are primarily interested in investigating the accuracy of algebraic performance models. We want to identify what types of models can be used when modeling full application codes in the context of comparing parallelization strategies. In earlier

work we found that the different phases of a parallel code place both implementation and performance constraints on each other, and that evaluation of kernels in isolation can be misleading, especially in a prototyping environment.

In this paper we show that a reasonably accurate prediction of performance measures is possible without requiring detailed application and system characterizations. We describe a case study employing algebraic models to predict the performance of the Parallel Spectral Transform Shallow Water Model code (PSTSWM) on the Intel Paragon, using these models to determine which parallel algorithm options are optimal for a given problem size and number of processors. We concentrate on the feasibility of such an approach for comparing parallelization strategies, and do not address directly how to generate accurate models before the application code has been written.

This research was possible only because of the prior existence of a number of tools: PSTSWM, PICL, and PerPreT. PSTSWM is a convenient testbed for such studies. PICL (Portable Instrumented Communication Library) was used to collect the performance data needed to construct and to validate the performance models [6]. PerPreT (Performance Prediction Tool) was used to define and evaluate the performance models [2]. All three tools are available via the World Wide Web from the following locations:

http://www.epm.ornl.gov/chammp/pstswm
http://www.epm.ornl.gov/picl
http://www.irb.uni-hannover.de/~brehm/publications

The remainder of this paper is organized as follows. Section 2 is a description of how to use the performance prediction tool PerPreT. Section 3 is a description of the PSTSWM code and of the different parallelization strategies. Section 4 is a description of the parametrized PerPreT formulae for PSTSWM. Section 5 is a description of the modeling experiments and an analysis of the results.

2. PerPreT

The performance prediction tool PerPreT uses high level descriptions of computation and communication of a parallel application and a high level system description to predict performance measures such as execution time, communication time, and speedup. The system description is derived by system parameters for computation (sustained MFLOP per seconds rate) and communication (setup time, link bandwidth, topology). Since many parallel scientific codes are SPMD programs, the current PerPreT implementation focuses on this programming model. An SPMD application is reduced to formulae for computation (number of arithmetic statements) and communication (calls to the communication library). The problem size for an application and the number of processors used to execute the SPMD program are free parameters.

The system and application descriptions are kept independent of each other. Thus, applications are modeled on different systems without the need of defining new application descriptions and vice versa. For modeling complex codes such as PSTSWM, PerPreT supports splitting the code into different computation phases according to their performance behavior. If extra operations for parallel computing are necessary (e.g., copy operations to prepare for communication), such extra phases can also be modeled with their performance characteristics. More details on PerPreT are described in [2].

3. PSTSWM

PSTSWM is a message-passing parallel program that solves the nonlinear shallow water equations on a rotating sphere using the spectral transform method. PSTSWM was developed to evaluate different parallelization strategies for global atmospheric circulation models [5].

PSTSWM advances the solution fields in a sequence of timesteps. During each timestep, the state variables of the problem are transformed between the physical domain, a tensor product longitude-latitude-vertical grid, and the spectral domain. Transforming from physical coordinates to spectral coordinates involves performing a real fast Fourier transform (FFT) for each line of constant latitude, followed by a numerical integration over latitude (approximating the Legendre transform (LT)) to obtain the spectral coefficients [7]. The parallel algorithms in PSTSWM are based on decompositions of the physical and spectral computational domains over a logical processor mesh of size **P=PX** x **PY**. Parallel efficiency is determined primarily by the efficiency of the parallel algorithms used for the FFT and LT transforms and by any load imbalances caused by the choice of domain decomposition.

Two classes of parallel algorithms are available for each transform: distributed algorithms, using a fixed data decomposition and computing results where they are assigned, and transpose algorithms, remapping the domains to allow the transforms to be calculated sequentially. We restrict ourselves to one transpose algorithm (T) for both FFT and LT, one distributed FFT algorithm (D), and two distributed LT algorithms (R and H). These generate six parallel algorithms (parallel FFT/parallel LT): DH, DR, DT, TH, TR, TT. For more detail, see [13].

4. Modelling PSTSWM

PerPreT expects separate formulae for the computation and communication. For PSTSWM, both computation and communication were further decomposed into phase models, representing distinct code fragments, each with their own rates. The phase models were derived from the source

Phase	Model	Rate
	physical domain computation	1/a, 1/b
1	12· NLLON_P · NLLAT_P · NLVER_P	4.8
	forward FFT	
2	$\lceil$(PX-1)/PX$\rceil$·32 · NLLAT_P · NLVER_P· (a+b·NLLON_P)	4.5, 23.1
3	$\lceil$(PX-1)/PX$\rceil$· 32 · NLLAT_P · NLVER_F· (a+b·NLLON_F)	17.7, 21.6
5	20· NLLAT_F · NLVER_F · NLLON_F · (a+b·log$_2$(NLLON_F/4))	3.8, 24.0
6	64· NLLAT_F · NLVER_F· (a+b·NLLON_F/4)	4.0, 15.2
7	144· NLLAT_F · NLVER_F· (a+b·NLLON_F/4)	10.4, 19.8
	forward LT	
9	(PY-1) · 6 · NLVER_S · NCSP_S/PY	4.4
10	61· NLVER_S · NLMM_S · NLLAT_S	10.0
11	(14· NLLAT_S-1) · NCSP_S · NLVER_S	15.1
	spectral domain computation	
12	13· NLSP_S · NLVER_S	11.5
	inverse LT	
13	17· NCSP_S · NLVER_S	7.0
14	(14· NCSP_S + 10 · NLMM_S)· NLLAT_S · NLVER_S	12.8
17	40· NLLAT_F · NLVER_F· (a + b·(NLLON_F/2 - NLMM_S))	22.1, 36.8
	inverse FFT	
18	70· NLLAT_F · NLVER_F· (a+b·NLLON_F/4)	8.8, 20.4
19	40· NLLAT_F · NLVER_F· (a+b·NLLON_F/2)	2.8, 18.6
20	(25/2)· NLLAT_F · NLVER_F · NLLON_F· (a+b·log$_2$(NLLON_F/4))	3.8, 24.0
21	$\lceil$(PX-1)/PX$\rceil$· 20 · NLLAT_F · NLVER_F· NLLON_F	10.2
22	$\lceil$(PX-1)/PX$\rceil$· 20 · NLLAT_F · NLLON_P· (a·PX + b·NLVER_P)	15.2, 18.6

Table 1. Computational models and MFLOP/s or MByte/s rates for algorithm TH

code. Rates were specified with either one or two parameters and were determined empirically from a series of 8-processor experiments.

As an example of the form, the computation phase models for algorithm TH are given in Table 1, where the parameters are functions of the problem size, number of processors, and domain decomposition. For full descriptions of these and the other models, see [1].

5. Experiments

The performance models are meant to be simple enough to be generated by the application developer, yet accurate enough to be used when scaling problem and machine parameters and when comparing alternative parallel algorithms. The approach taken here has been to construct the application model from a set of phase models. In this section we use the models to examine the following performance questions:

1. What is the best parallel algorithm to use for a given number of processors?
2. How long will the application take to complete a run?

Two problem sizes were investigated, denoted by T42 and T85. Only T42 results are presented here. The T85 results are described in [1].

For the two performance questions, we discuss `P=8,16,32,64,128,256,512`. The optimal parallel algorithms are determined over all algorithms and processor mesh aspect ratios. The estimation of runtimes is discussed in terms of the optimal parallel algorithms. Finally, we reexamine the models, evaluating the effectiveness and importance of the phase model approach in being able to answer the performance questions.

5.1. Optimal parallel algorithm

Determining the optimal parallel algorithm experimentally requires developing, tuning, and evaluating multiple parallel implementations. This is very time consuming, and there is much to be gained from using performance models to predict the optimal parallel algorithm. The relative accuracy of the execution time prediction is important here, not the absolute accuracy. Table 2 indicates the true and predicted optimal parallel algorithm for different numbers of processors and the percentage loss from using the model-identified algorithm. The optimal aspect ratio was found for each parallel algorithm before being compared with the other parallel algorithms. The model results use the model-determined optimal aspect ratios. The empirical results use the experimentally-determined optimal aspect ratios.

	T42				
Processors	model optimum		experimental optimum		%diff. in runtime
8	DR	1x8	DR	1x8	-
16	DT	1x16	DT	1x16	-
32	TR	8x4	TR	8x4	-
64	TR	16x4	TR	16x4	-
128	TH	16x8	TR	16x8	1.1
256	TH	16x16	TT	16x16	3.7
512	TH	16x32	TH	16x32	--

Table 2. Error in choosing opt. algorithm from model results instead of experimentally.

The performance models correctly identify the optimal algorithm and aspect ratio in five out of seven cases. The errors in misidentifying the optimal algorithm were acceptable. The performance sensitivity of choosing the wrong algorithm (but with an optimum aspect ratio) is not as extreme as when choosing the aspect ratio, but worst case errors range as high as 85%. Note that when considering a larger sampling of interesting problem sizes, all of the parallel algorithms are optimal in some cases. It is not possible to eliminate any of the parallel algorithms *a priori.*

5.2. Runtime predictions

When allocating resources, it is important to know how long a parallel job will take to run on a given number of processors. For example, runtime information is often required when submitting batch requests. This type of prediction requires a certain degree of absolute accuracy, but the degree needed is not great. (However, accurate predictions of runtime can be extremely important in real-time environments). Table 3 indicates how accurately the models predict the runtime for the model-determined "optimal" parallel algorithms (to pick particular examples). Sources for the error in the predictions are discussed in [1]. With possibly one exception, the accuracy of these predictions is adequate for the determination of resource requirements. Note that similar accuracies hold for predicted speedup and parallel efficiency.

		T42	
Processors	algorithm	predicted runtime	%error in prediction
8	DR 1x8	79.8	- 1.6
16	DT 1x16	40.9	- 6.6
32	TR 8x4	23.0	2.2
64	TR 16x4	12.2	1.7
128	TH 16x8	6.7	- 5.4
256	TH 16x16	4.0	- 11.1
512	TH 16x32	2.6	- 27.8

Table 3. Error in predicting runtimes (seconds)

5.3. Model accuracy requirements

The previous results indicate that the accuracy of our phase model approach is adequate for algorithm tuning and comparison for this case study. We next discuss whether a simpler model might also suffice.

There are numerous ways to simplify the current model. Here we consider only a few obvious alternatives. First, we choose the optimal algorithm on the basis of arithmetic complexity alone, ignoring copy phases, communication costs, and phase-dependent rates. (Including copy and communication complexity would require some sort of rate estimation to weight the different components of the model.)

Table 4 indicates the true and predicted optimal parallel algorithms using this simplified model, and the percentage loss from using the model-identified algorithm. These predictions are not as good as those from using a phase model. Depending on the application, the size of these errors may or may not be acceptable. But, since the error in the prediction is not known in practice, the wide and unpredictable variation in the error is worrisome.

		T42	
Processors	model optimum	experimental optimum	%diff. in runtime
8	DT 1x8	DR 1x8	6.6
16	DT 1x16	DT 1x16	-
32	DT 2x16	TR 8x4	17.3
64	DT 4x16	TR 16x4	22.3
128	TT 16x8	TR 16x8	2.7
256	TT 16x16	TT 16x16	-
512	TR 16x32	TTH 16x32	45.1

Table 4. Error in choosing optimal algorithm from complexity analysis instead of experimentally.

We can not predict runtimes from the complexity analysis alone. The next model we consider uses the sustained computation rate for an 8-processor run for a given parallel algorithm to weight the corresponding arithmetic complexity model. Table 5 indicates how accurately this model predicts the runtime for the above model-determined "optimal" parallel algorithms. For this type of model to be accurate requires that either copy and communication costs are negligible or they scale similarly with the computation costs, and that the rates are insensitive to scaling. It is clear from Table 5 that these conditions do not hold for PSTSWM.

		T42	
Processors	algorithm	predicted runtime	%error in prediction
16	DT 1x16	41.2	- 6.3
32	DT 2x16	20.8	- 21.1
64	DT 4x16	10.6	- 27.4
128	TT 16x8	5.5	- 23.0
256	TT 16x16	3.0	- 32.0
512	TR 16x32	1.6	- 56.1

Table 5. Error in predicting runtime (seconds) using complexity based model.

Our final simplified model includes terms for computation, copy, and communication costs, but does not take into account phase-specific rates. Instead we use average copy and computation rates determined from the 8-processor runs. Table 6 indicates how accurately this type of single-phase model predicts the runtime for the phase model "optimal" parallel algorithms (to allow direct comparison with the phase model results).

With the exception of predictions for T42 for large numbers of processors, the single-phase model is as accurate a predictor of runtime as is the (multiple-) phase model. So the question arises whether a phase model is required as long as the copy, computation, and communication costs are included in the model. A phase model does not appear to be required for accurate performance prediction for PSTSWM.

Processors	algorithm	T42 predicted runtime	%error in prediction
8	DR 1x8	86.8	7.0
16	DT 1x16	43.9	0.2
32	TR 8x4	23.3	3.7
64	TR 16x4	12.1	0.9
128	TH 16x8	6.6	- 7.2
256	TH 16x16	3.8	- 15.5
512	TH 16x32	2.4	- 32.4

Table 6. Error in predicting runtime (seconds) using single phase model

However, we found constructing the phase model to be necessary. The error prone aspect of the phase model approach was in the generation of the phase model expressions. These same expressions are needed in a single-phase model or in a complexity analysis. The additional step of calculating rates and validating the individual phase models also validates the expressions. Modeling phases can also identify performance "problems", for example, code that is overly sensitive to aspect ratio due to compiler peculiarities. Using average rates and a single-phase model removes the necessity of detailed profiling to determine individual phase model rates, but makes it more difficult to validate the model.

6. Conclusions

This case study demonstrates that relatively simple algebraic models can be used to construct scalable performance models for use in algorithm tuning and comparison. These models can be difficult to generate and validate, but the phase model approach makes it feasible to do so. In addition, constructing and modifying models and generating predictions were easy using PerPreT.

A phase model approach was useful in generating a performance model, but it may not be necessary when "porting" the model to a new platform. As described earlier, single rates for computation, copy, and communication phases may be sufficient when using the model for predictions. In future work, we will examine this issue by repeating our evaluation studies on the IBM SP2 and on the Cray Research T3D or T3E. The SP2 will be a particularly interesting platform; communication costs are relatively high, and a simple communication model may not be adequate.

7. Acknowledgments

This research was supported by the U.S. Department of Energy under Contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Inc. and by the Alexander von Humboldt foundation. The Intel XP/S 150 MP Paragon operated by the Center for Computational Science at ORNL is funded by the Department of Energy's Mathematical, Information and Computational Sciences Division of the Office of Computational and Technology Research.

8. References

[1] J. Brehm, P. H. Worley, and M. Madhukar, *Performance Modeling for SPMD Message-Passing Programs*, Tech. Report ORNL/TM-13254, Oak Ridge National Laboratory, Oak Ridge, TN, June 1996

[2] J. Brehm, L. Dowdy, M. Madhukar, and E. Smirni, *PerPreT - a performance prediction tool*, in Quantitative Evaluation of Computing and. Communication Systems, Lecture Notes in Computer Science 977, Springer, Heidelberg, 1995.

[3] M. Calzarossa and G. Serazzi, *Workload charaterization - a survey*, Proceedings of the IEEE, 81 (1993), pp. 1136-1150.

[4] T. Fahringer, *Estimating and optimizing performance for parallel programs*, IEEE Computer, 28 (1995), pp. 47-56.

[5] I. T. Foster, and P. H. Worley, *Parallel algorithms for the spectral transform method*, Tech. Report ORNL/TM-12507, Oak Ridge National Laboratory, Oak Ridge, TN, May 1994.

[6] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, PICL: a portable instrumented communication library, *C reference manual*, Tech. Report ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, TN, July 1990.

[7] J. J. Hack and R. Jakob, *Description of a global shallow water model based on the spectral transform method*, NCAR Tech Note NCAR/TN-343+STR, National Center for Atmospheric Research, Boulder, CO, February 1992.

[8] P. Heidelberger and K. S. Trivedi, *Analytic queuing models for programs with internal concurrency*, IEEE Trans. Comput., c-32 (1983), pp. 73-82.

[9] M. Parashar and S. Hariri, *Compile time performance prediction of HPF/Fortran 90D*, IEEE Parallel and Distributed Technology, 4 (1996), pp. 57-73.

[10] S. R. Sarukkai, P. Mehra, and R. J. Block, *Automated scalability analysis of message-passing parallel programs*, IEEE Parallel and Distributed Technology, 3 (1995), pp. 21-32.

[11] A. Thomasian and P. F. Bay, *Analytic queuing network models for parallel processing of task systems*, IEEE Trans. Comput., c-35 (1986), pp. 1045-1054.

[12] H. Wabnig and G. Haring, *PAPS - the parallel program performance prediction toolset*, in 7th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, 1994, pp. 284-304.

[13] P. H. Worley and B. Toonen, *A user's guide to PSTSWM*, Tech. Report ORNL/TM-12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.