Transactions on Information and Communications Technologies vol 9, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

Parallelization of software for coastal hydraulic simulations for distributed memory parallel computers using FORGE 90

Z.W. Song, D. Roose, C.S. Yu, J. Berlamont
Laboratory of Hydraulics and Department of Computer Science, Katholieke Universiteit te Leuven, de Croylaan 2, B-3001 Heverlee, Belgium

Abstract

Due to the increasing availability of powerful distributed memory parallel computers, the parallelization of existing sequential software is a very important issue. Since this is often difficult and time-consuming, the usage of software tools for analysis and (semi-)automatic parallelization may be of great help. In this study, the usage of the FORGE 90 software tool is illustrated by its application to two software packages in the field of coastal hydraulics. FORGE 90 consists of two parts : '*Baseline FORGE 90*' provides many facilities to analyze sequential Fortran programs, and the '*FORGE 90 parallelizer*' is an interactive tool which can be used to create a parallel version of a Fortran program targeted to execute on a distributed memory machine. The former function of the package has been used to analyze the water quality model - DIVAST. The latter function of FORGE 90 has been used to parallelize the hydrodynamic model mu-CSM.

1 Introduction

Parallel computing will only be accepted by a broad application community, if parallel programming can be made nearly as easy as sequential programming. With the existing programming models we are still far from that goal [1]. Two paradigms dominate the programming of parallel computers: the shared memory model and the message-passing model. Programming in the shared memory paradigm can be facilitated and made machine independent by coordination languages [2] or by using parallelizing compilers. Programming in the message-passing model is made machine independent and is simplified by

appropriate parallel programming libraries. Examples are PVM (Parallel Virtual Machine) and MPI (Message Passing Interface). However, the programmer still has to take care of domain partitioning, communication and synchronization. The ultimate solution for the parallel programming problem will be compilers that automatically translate a (sequential) code into a optimized parallel version. Then, the user can program the application in a conventional style, without having to be concerned with parallel computing aspects such as data partitioning, data distribution, communication and synchronization. Research in the domain of parallelizing compilers currently goes into two directions. In the first approach, the programmer guides the compiler in parallelizing the program by means of directives. The best-known example of a language for directive-driven parallel programming is *High Performance Fortran* (HPF). In the second approach the compiler parallelizes a program automatically. An example of such a compiler is FORGE 90, developed by Applied Parallel Research (APR), Placerville, CA.

Very interesting and practical questions are: (a) to which extend the present parallel software tools, such as FORGE 90, can help the programmer to do the porting job? (b) which parallel performance can be achieved?

In this paper, we first briefly describe the capabilities of the FORGE 90 software tool. We report on the use of '*Baseline FORGE 90*' for the analysis of a sequential FORTRAN code, namely the water quality model DIVAST, developed at the Univ. of Bradford, U.K. [5]. Further, we study the (semi-) automatic parallelization of some sequential codes using FORGE 90. Several numerical kernels (Jacobi, Gauss-Seidel and ADI type) used in the solution of the Two Dimensional Shallow Water Equations have been parallelized using FORGE 90 and tested on an iPSC/860 parallel computer. A Fortran code developed at K.U.Leuven, Belgium [9], for the simulation of 2D water flow has been parallelized using FORGE 90. Timing results for the tidal flow simulation of the Northwest European Continental Shelf Seas on an iPSC/860 are presented.

2 Introduction to FORGE 90

Many simulation codes have been developed, maintained and renewed by different persons over a long time period. Furthermore, many of these codes were written by engineers who are not well-trained in software development. This leads to a situation where a program's control flow and data flow may no longer be easily comprehensible.

To port and to maintain a Fortran program, a global view of its symbol usage and its control structure must be available.

'*Baseline FORGE 90*' is a comprehensive set of tools that provides the required high-level approach to these problems. It consists of a Fortran syntax parser, database generator, program maintaining facility, and database viewing tools that enable to examine and manipulate an entire Fortran program as a simple entity. The analysis capabilities of FORGE 90 enable to understand

program control flows and usage of variables and data. Runtime statistics are provided by an instrumentation facility [3].

The '*FORGE 90 parallelizer*' is an interactive tool that can be used to create a version of a Fortran program targeted to execute on a distributed memory machine. Higher-level analysis and restructuring modules for vectorization and parallelization are added to the baseline system. A pre-processor and a run-time library that executes on the parallel system allow the interactive parallelizer to work essentially independent of the target system. Specific system dependencies are handled by the pre-processor and the run-time library. FORGE 90's role is to advise the user regarding the parallelizability of do-loops, and to display the inter-processor communications required when loops are parallelized. The actual parallelization of Fortran code occurs when the user chooses a set of loops to be distributed over the processors, and a data partitioning scheme for the arrays enclosed in those loops. The '*FORGE 90 parallelizer*' does the rest, rewriting the loops and inserting array partitioning and data communication directives to be handled by the pre-processor. The FORGE 90 distributed memory parallelizer uses the Single Program - Multiple Data (SPMD) model. The primary concerns for this SPMD model are the choice of loops to be distributed and the partitioning of data over processors [4].

3 Analysis of the DIVAST software for water quality simulation using *'Baseline FORGE 90'*

DIVAST is a numerical model designed for predicting the water elevation and depth averaged velocity components in the horizontal plane and up to eight user specified water quality constituents and sediment transport fluxes. DIVAST is developed by R.A. Falconer [5] at the University of Bradford, UK. The governing differential equations are solved using a finite difference discretization and a time-integration scheme based on the Alternating Direction Implicit (ADI) formulation. The model has been used extensively by industrial engineering companies for commercial applications and by over 30 universities world-wide for research purposes.

The parallelization of the ADI scheme is quite difficult due to the alternating data dependencies in x- and y-direction respectively. Several approaches to parallelise ADI-schemes have been developed during the last years (see Song et. al. [6]). The most time consuming part of the porting work is the understanding of the control flow and data flow of the Fortran code. The parallelization is based on a partitioning of the domain into several sub-domains. The coordination of the sub-domains is done by exchanging information between the sub-domains. Depending on the partitioning strategy, the variables involved in the communication must be identified. To do this, the global and local variables in each routine must be separated. For the global variables - which are passed between various subroutines - it has to be clear which variables are used, set or used and set in a particular routine. The global

array variables in the overlap regions must be updated before they are used. The global variables which are set in a routine must be transferred to the processors holding neighbouring sub-domains to re-fresh the overlap regions. In the DIVAST model, also some scalar global variables are set in some routines. For this type of variables, a broadcast is necessary instead of communication between neighbouring sub-domains.

Implementation of the communications described above (communication between neighbours or broadcasting) is not difficult, but to determine the proper communication for all variables is very time consuming. In this respect, FORGE 90 provides more information and support than classical compilers.

Using FORGE 90, it is easy to display the global input and output variables (parameters or arguments and COMMON block variables) for each routine. For each routine, a table is generated, from which it can be easily determined which variables are updated in a particular routine. In case a scalar variable is updated, it is very likely that a broadcast of that variable from one of the processors is necessary. In case an array variable is updated and this variable is used in some routine afterwards, it is probably necessary to introduce a communication step in between.

The COMMON Block Grid function in FORGE 90 lists all the COMMON blocks and summarizes the usage of variables in COMMON blocks in each routine. This function gives a general picture of the COMMON variables in the sequential code.

FORGE 90 also displays the inter-dependencies between subroutines (or program blocks) in a table, which shows, for each subroutine, which other routines are called by that particular routine.

The analysis function of FORGE 90 can be summarized as follows: trace variables, display constants, query the database, list all routines, show the COMMON grid, perform the data flow analysis and view timing information.

The analysis capabilities of FORGE 90 enable us to understand, to explore and to scrutinize complicated Fortran application code within a short time period. Our experiences with FORGE 90 baseline to analyze the water quality model DIVAST are very positive. With '*Baseline FORGE 90*', we can efficiently examine the program using a tool set designed for easy query of the database information. A clear picture of the DIVAST code was available within a week.

4 Parallelization of numerical kernels using '*FORGE 90 parallelizer*'

In coastal hydraulics, hydrodynamic models are based on the Shallow Water Equations (SWEs). Commonly used time integration schemes for the SWEs range from explicit, semi-implicit to implicit. The numerical kernels involved in these schemes are an explicit (or Jacobi type) update, an ADI scheme and iterative linear system solvers of Gauss-Seidel type. To test the capability of the '*FORGE 90 parallelizer*' for distributed memory parallel computers, these kernel

problems for the solution of the 2D SWEs have been parallelized using FORGE 90 and executed on an Intel iPSC/860 machine.

An explicit time-integration scheme (or Jacobi-type scheme) can be parallelized easily.

The semi-implicit Alternating Direction Implicit (ADI) method is a popular time-integration scheme. The data dependencies in the ADI scheme occurs alternatingly in x- and y-direction, respectively. As mentioned before, the primary aims of the FORGE 90 parallelizer are the distribution of do-loops and the data partitioning. Thus, the only reasonable way to parallelize the ADI method using FORGE 90 is to introduce a complete data transposition operation in between the two half-steps in order to avoid the distribution of the coupled data over several processors in one of the half-steps.

When an implicit time-integration scheme is used, the resulting linear systems are normally solved by iterative methods. Gauss-Seidel (or SOR) iteration is a simple and, in case of SWEs, a relatively efficient method. However, the lexicographic Gauss-Seidel (or SOR) method can not be parallelized. To explore parallelization, a colouring scheme has to be used (e.g. Red-Black Gauss-Seidel).

We have compared the timing results and the parallel performance of the hand parallelized version and the version parallelized using FORGE 90 for the kernel problems mentioned above. The results are listed in tables 1, 2 and 3. The parallel efficiencies are always computed using the original sequential code as reference.

Table 1: Jacobi type scheme (timings in seconds and parallel efficiency)

# Proc.	1	2	4
Hand coded	17.96/100%	9.13/98%	4.73/95%
FORGE	18.07/99%	9.14/98%	4.78/94%

Table 2: Gauss-Seidel type scheme (timing in seconds and parallel efficiency)

# Proc.	1	2	4
Hand coded	10.36/100%	5.35/97%	2.87/90%
FORGE	10.56/98%	5.43/95%	2.99/87%

Table 3: ADI type scheme (timing in seconds and parallel efficiency)

# Proc.	1	2	4
Hand coded	45.69/100	27.4/83%	15.2/75%
FORGE	68.5/67%	36.4/63%	18.8/61%

By using the '*FORGE 90 parallelizer*', porting a sequential Fortran code into a machine-independent parallel code becomes straightforward. FORGE parallelizes a code by distributing the execution of a do-loop over the processors. This is essentially equivalent to a one-dimensional partitioning of the work load. The choice of loops to be distributed and the data partitioning are the two major concerns for the programmer, while all the machine-dependent issues, communication and synchronization between processors are handled by the run-time library. The performance results of the code parallelized using FORGE 90 are nearly the same as that of a hand parallelized code for simple kernel problems (tables 1 and 2). However, the ADI kernel (table 3) poses some problems. The execution time of the parallel code generated by FORGE is much higher than that of the code parallelized by hand. This is especially true when only 1 or 2 processors are used. We do not have an explanation for this. When 4 processors are used, the differences in execution time are acceptable. Some of the inefficiencies introduced by FORGE are due to additional communication when FORGE cannot decide at compile-time about data dependencies. In these cases, additional communication is inserted to ensure correct results.

5 Parallelization of mu-CSM hydrodynamic model using FORGE 90

mu-CSM is a numerical model for simulating tidal flows in the Northwest European Continental Shelf Seas. mu-CSM has been jointly developed by the Laboratory of Hydraulics, K.U.Leuven and the Management Unit of the Mathematical Models of the North Sea and the Scheldt Estuary, Belgium [7,8]. The model is constructed based on a finite difference discretization and a modified ADI time integration scheme [9]. The model covers an area from 12^0W to 13^0E and 48^0N to 63^0N. The mesh is defined by a spherical grid with a grid size of 2.5 minutes in the latitude direction and 5 minutes in the longitude direction, which results in a resolution of about 4km x 4km in the centre of the model area. The total number of grid points is 102,076 while 44,302 grid points are 'wet' points for which the calculation must be performed.

The performance results obtained with the code parallelized by hand and the version parallelized using FORGE 90 are listed in table 4.

Table 4: Timing in seconds and parallel efficiency for the mu-CSM model.

# Proc.	1	2	4
Hand coded	1017/100%	575/88%	312/81%
FORGE (original code)	2927/35%	3223/16%	3682/7%
FORGE (modified code)	1690/60%	962/53%	637/40%

When the original mu-CSM code is parallelized using the *'FORGE 90 parallelizer'*, the performance results for the Northwest European Continental Shelf Sea application are unacceptable (table 4, second row). This can be explained as follows. Due to the traditional way of programming in hydrodynamic modelling, the velocities in x- and y-direction are calculated according to the orientation of the velocity. The one-dimensional partitioning strategy in FORGE 90 leads to a situation where the inner loops are distributed. This situation generates many short messages for communication and this results in a substantial degradation in performance. We have modified the mu-CSM code such that the inner loop distribution disappeared. The execution time decreased significantly but is still much higher than the timing for the hand coded version (see table 4, third row).

The parallel efficiency of the modified mu-CSM code generateded by FORGE 90, is much lower than the efficiency of the hand coded parallel code also for the following reasons. The computational domain of the Northwest European Continental Shelf Sea application has a irregular shape but the information is stored in matrices, i.e. regular structures. The domain partitioning done in FORGE 90 is based only on the regular data structures that are ***allocated***, so an unbalanced work load distribution occurs, because large parts of the allocated data structures are not used. This is an important factor that influences the performance results. Further, the mu-CSM code is a complicated operational model. Due to this complexity, FORGE 90 introduces more communications between processors than in the hand parallelized code, in order to ensure correct results.

6 Conclusions

FORGE 90 provides the problem-solving capabilities needed when one is trying to understand a complicated Fortran code in its current environment, and when porting it to a parallel architecture. A significant time saving has been experienced in analysing the water quality model DIVAST.

The *'FORGE 90 parallelizer'* is capable of parallelizing simple kernel problems efficiently. For application codes, such as the mu-CSM model, a detailed analysis of the usage of the variables in the routines and some modifications to the original code are necessary to be able to obtain a good parallel performance. Applications with irregular data, e.g. the Northwest European Continental Shelf Sea application, can hardly be parallelized well by using the *'FORGE 90 parallelizer'*. This is especially true if regular data structures (matrices) are used to store the (irregular) data. In such case, a partitioning of the allocated data structures does not necessarily lead to a well-balanced partitioning of the actual data. It is important to point out that, in our experiences, the codes generated by FORGE 90 always produce correct results.

ACKNOWLEDGEMENT

We are grateful to Uwe Block from Genias GmbH for his kind cooperation on the usage and the evaluation of the FORGE 90 package. We acknowledge also the help of Applied Parallel Research (APR). Thanks also to Liliane De Roose for her help concerning the use of the iPSC parallel computer.

REFERENCES

1. Giloi, W.K. (1994) Programming Modelling and Tools for Massively Parallel Computers, In L. Dekker, W. Smit and J.C. Zuidervaart (Eds), *Massively Parallel Processing Applications and Development*, pp. 3.14, Elsevier Sciences B.V, Amsterdam 1994.
2. Gelernter, D. Generative (1985) Communication in Linda, *ACM Trans. on Programming Languages and Systems*, 1985, 7(1), 80-112.
3. Applied Parallel Research, (1992) *FORGE 90 Version 8.0 Baseline System,* Sep. 1992.
4. Applied Parallel Research, (1993) *FORGE 90 Version 8.7 Distributed Memory Parallelizer User's Guide*, April, 1993.
5. Falconer, R.A. (1986) A Two-Dimensional Mathematical Model Study of the Nitrate Levels in an Inland Natural Basin, pp. 325-344, Pro. Int. Conf. on BHRA, *Fluid Engineering*, Paper J1, June 1986.
6. Song, Z.W., C.S. Yu, D. Roose and J. Berlamont, (1993) Solving the 2D shallow water equations by explicit and ADI methods on distributed memory parallel computers, (eds Brebbia, C.A. and Power, H.), pp. 239-252, *Applications of Supercomputers in Engineering III*, Computational Mechanics Publications, Southampton.
7. Yu, C.S., A. Vermunicht, M. Fettweis, and J. Berlamont (1989) *Numerical simulation of long waves on the north-west European continental shelf. Part 1 : data collection and model test.* Technical report to the Ministry of Public Health and Environment, Belgium, Ref. BH/88/28.
8. Yu, C.S., A. Vermunicht, M. Rosso, M. Fettweis and J. Berlamont (1990) *Numerical simulation of long waves on the north-west European continental shelf. Part 2 : Model setup and calibration.* Technical report to the Ministry of Public Health and Environment, Belgium, Ref. BH/88/28.
9. Yu, C.S. (1993) *Modelling Shelf Sea Dynamics*, Ph.D Thesis, Faculty of Applied Science, K.U.Leuven, Belgium.