Parallelizing Flood Models with MPI: Approaches and Experiences[1]

Viet D. Tran and Ladislav Hluchy

Institute of Informatics, Slovak Academy of Sciences
Dubravska cesta 9, 842 37 Bratislava, Slovakia
viet.ui@savba.sk

Abstract. Parallelizing large sequential programs is known as a challenging problem. This paper focuses on problems encountered during parallelization process of different flood models and on the approaches used for solving them. The approaches are focused on reducing development time, which can help programmers make a parallel version of existing sequential programs within a short time.

1 Introduction

Over the past few years, floods have caused widespread damages throughout the world. Most of the continents were heavily threatened. Therefore, modeling and simulation of floods in order to forecast and to make necessary prevention is very important. As Linux clusters are widely used as low-cost high performance platforms, it is important to make the parallel versions of the flood models running on Linux clusters. That limits the possibility of using OpenMP or parallel compilers for parallelization. Therefore, programmers have to rely on MPI or other message-passing libraries for developing the parallel version of the flood models.

This paper focuses on the problems encountered during parallelizing flood models using MPI and solutions for them. In Section 2, the flood models are introduced. The problems encountered during parallelization and their solutions are discussed in Section 3. Section 4 gives the results of the parallelization and Section 5 concludes the paper.

2 Numerical Flood Models

At the beginning of ANFAS project [4], many surface-water flow models were studied in order to find a suitable high-performance model for pilot sites at Vah river in Slovakia and Loire river in France. The result of the study showed that many models exist only in sequential forms. Two models were chosen for the pilot site; one is FESWMS [3] which is based on finite element approach and is distributed with

[1] This work is supported by EU 5FP CROSSGRID IST-2001-32243 RTD and the Slovak Scientific Grant Agency within Research Project No. 2/3132/23

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3036, pp. 425–428, 2004.

commercial package SMS [9] by EMS-I. The second model is DaveF, a new model based on time-explicit, cell-centered, Godunov-type, finite volume scheme.

Although both models are used for modeling water flow, they are based on completely different numerical approaches. Detailed descriptions of the numerical approaches of the models can be found in [5]. This paper focuses on problem encountered during its parallelization and solutions for the problems. Therefore, the following descriptions of computational approaches are purely from the view of parallel programming.

3 Problems Encountered during Parallelization with MPI

Understanding Algorithms
Although the mathematical approaches (finite elements, finite volumes) of the models may be well-known, there are many hydraulic details in the algorithms such as different boundary conditions, wetting/drying, raining/infiltration and different tricks to stabilize the solutions. Such details complicate the programs considerably and are not easy to understand for the programming experts who parallelize the source.

However, in our parallelization approaches (see later in data and code duplications), the programmers do not have to understand all the details in the algorithms. From the view of parallelization, every finite-element model consists of three steps: generating the matrix, solving the matrix and updating solutions. The programmers do not have to learn what governing differential equations are used in the models or Galerkin method works, as the implementation already exists in the source code and they are not going to change them. Similarly, the finite-volume algorithm can be simplified as follows: in every time step, each cell updates its values from its current values and the values of its neighbors. That frees the programmers from learning all details in hydraulics and allows them to parallelize applications they are not familiar with.

Understanding the Source Codes
The source code of FESWMS has about 65 thousand lines, i.e. about one thousand pages; DaveF is nearly the same. Reading and understanding the source codes (especially when programmers do not understand all details in algorithms) for identifying the critical part may take a long time. Therefore, profiling tools (e.g. gprof in Linux) are extremely useful for parallelizing sequential programs. By using profiling tools, programmers can easily identify the computation-intensive parts in the source code (computation kernel), see the call graphs and analyze the performance of the program. Programmers then can concentrate on studying the computation kernel needed to parallelize/optimize, and consider the rests of the source code as “black boxes”. Discussion with the original authors of the models is also useful for understanding the most important data structures (e.g. global arrays of nodes/cells/elements).

Writing the Parallel Code
After analyzing the algorithm and the source code, the programmers can start to write parallel versions of the models. Paralleling with MPI for Linux clusters adds some more problems. It may be argued whether writing a parallel program from scratch

with MPI on distributed-memory architectures such as Linux clusters is easier or more difficult than with OpenMP on share-memory systems such as supercomputers [6]. However, to parallelize existing sequential programs it is much easier to use OpenMP, because OpenMP does not change program and data structure.

In our approach, data and codes, which are not interesting in parallelization, are duplicated to reduce the development time. The data and code duplication greatly reduce the amount of codes that need to be modified during parallelization. That also allows programmers to ignore the implementation details in the parts of the codes that are duplicated. The programmers have to understand only the very basic computation scheme of the algorithms used in the models, and study/modify only few routines in the computation kernel of the models during parallelization.

4 Experimental Results

Experiments have been carried out on Linux cluster at the Institute of Informatics (II-SAS) in Slovakia. The Linux cluster at II-SAS consists of 16 computational nodes, each with a Pentium IV 1800 MHz processor and 256 MB RAM. All of the nodes are connected by an Ethernet 100Mb/s switch. Input data for the experiments are taken from Vah river in Slovakia and Loire river in France.

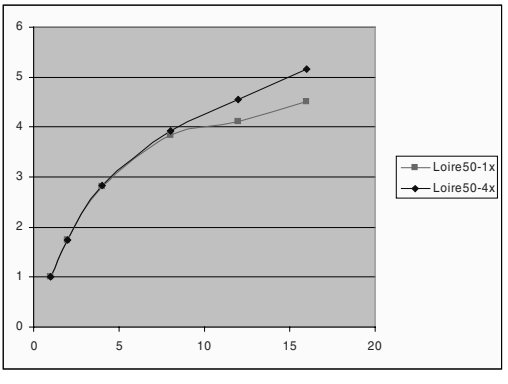

Fig. 1. Speedup of DaveF on II SAS cluster

In parallel version of FESWMS, only about 50 lines of code are modified from the 65000 lines in original sequential version, and 150 new lines of code are created for the new parallel iterative matrix solver from PETSC library, which replaces the frontal solver from the original sequential version. The speedup of FESWMS is difficult to describe in a table or graph. There are several iterative solvers and preconditioners, each of them has also several additional parameters. According to our experiments, the combination of BiCGStab method and ILU preconditioner is the quickest one (about 10x faster than the original frontal method), but GMRES/ILU is the most stable combination in the sequential version. Generally the speedup of FESWMS is about 5-7 on 16 nodes.

The parallel version of DaveF has less than modified 100 lines from its 45000 lines of code in the original sequential version and it is developed in 3 days. That clearly proves the advantages of our approach: to develop a parallel version in a very short time. Fig.1 shows the speedup of DaveF on II-SAS with two different input data from Loire river, the one being four times larger than the other one. It is easy to see that the speedup is increased with the size of input data, especially for larger number of processors. The reason is the fine granularity of DaveF; the more processors are used the larger is the granularity performance effect.

5 Conclusion and Future Work

This paper has presented an approach to parallelizing sequential flood models. The approach allows programmers to produce parallel versions within very short time. The approach is proved with two different flood models that are used in the ANFAS project.

At the moment, both models have been ported to Grid environment in CrossGrid project [7] and are running in CrossGrid testbed [8]. The details of Grid-aware Flood Virtual Orgranization, where DaveF is used, are described in a separate paper [2].

References

1. L. Hluchy, V. D. Tran, J. Astalos, M. Dobrucky, G. T. Nguyen, D. Froehlich: Parallel Flood Modeling Systems. International Conference on Computational Science ICCS'2002, pp. 543-551.
2. L. Hluchy, V. D. Tran, O. Habala, J. Astalos, B. Simo, D. Froehlich: Problem Solving Environment for Flood Forecasting. Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting 2002, pp. 105-113.
3. FESWMS - Finite Element Surface Water Modeling. http://www.bossintl.com/html/feswms.html
4. ANFAS Data Fusion for Flood Analysis and Decision Support. http://www.ercim.org/anfas/
5. D. Froehlich: IMPACT Project Field Tests 1 and 2: "Blind" Simulation by DaveF. 2002.
6. OpenMP, MPI and HPF: Comparing The Three: Which Programming Model Is Best? The Portland Group, Inc. http://www.pgroup.com/SLC2000/omp_hpf_mpi_files/frame.htm.
7. EU 5FP project CROSSGRID. http://www.crossgrid.org/
8. Marco, R.: Detailed Planning for Testbed Setup. The CrossGrid Project, 2002. http://grid.ifca.unican.es/crossgrid/wp4/deliverables/CG-4-D4.1-001-PLAN.pdf
9. Surface-water Modeling System. http://www.ems-i.com/SMS/sms.html.