Mathematical Modelling and Analysis 2005. Pages 171–177
Proceedings of the 10th International Conference MMA2005&CMAM2, Trakai
© 2005 Technika ISBN 9986-05-924-0

APPLICATION OF PARALLEL ARRAYS FOR SEMIAUTOMATIC PARALLELIZATION OF FLOW IN POROUS MEDIA PROBLEM SOLVER

A. JAKUŠEV, V. STARIKOVIČIUS and R. ČIEGIS

Vilnius Gediminas Technical University

Saulėtekio al. 11, LT-10223 Vilnius, Lithuania

E-mail: `alexj@fm.vtu.lt; vs@sc.vtu.lt; rc@fm.vtu.lt`

Abstract. Parallel Arrays library *ParSol* is the C/C++ library for semiautomatic parallelization of data parallel algorithms. This library offers sequential and parallel arrays to be used in C/C++ algorithm implementations. Any program that uses sequential *ParSol* arrays and is written in accordance with some simple rules, may be parallelized in similar way as Fortran 90 program may be parallelized by using HPF. *ParSol* uses MPI for interprocess communication.

In this article, issues of application of parallel arrays for parallelization of `MfsolverC++` is discussed. This solver simulates two-phase immiscible flow in porous media using mathematical model with global pressure formulation.

Key words: porous media, numerical algorithms, parallel agorithms, solvers

1. Introduction to Porous Media Problems

Porous medium consists of solid phase and void spaces. The void spaces may be filled with various gaseous and fluid phases. In order to derive mathematical models for fluid flow, porous medium must fulfill the following requirements:

1. The dimensions of the void space must be small enough so that the fluid flow is controlled by adhesive and cohesive forces (multiphase systems);
2. The dimensions of the void space must be large compared to the mean free path length of the fluid molecules;
3. The void spaces of the porous media are interconnected.

The flow in porous media is described by the following equations [5]:

$$\frac{\partial \left(\Phi \rho_\alpha S_\alpha\right)}{\partial t} + \nabla \cdot \{\rho_\alpha \mathbf{u}_\alpha\} = \rho_\alpha q_\alpha \ , \tag{1.1}$$

$$\mathbf{u}_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\nabla p_\alpha - \rho_\alpha \mathbf{g}\right) \ , \tag{1.2}$$

$$p_{c\beta\alpha}\left(\mathbf{x},t\right) = p_\beta\left(\mathbf{x},t\right) - p_\alpha\left(\mathbf{x},t\right), \ \beta \neq \alpha \ \ , \tag{1.3}$$

$$\sum_\alpha S_\alpha = 1\,, \tag{1.4}$$

where equation (1.1) is the mass conservation law for every phase α, equation (1.2) is the Darcy law for every phase α, equation (1.3) describes the capillary pressure for all phase pairs (α, β).

To solve this system of equations, *global pressure formulation* approach is used, since equations in this model are less coupled and entering quantities are smoother [6].

2. Solver for Computation of Flows in Porous Media

We use our software tool `MfsolverC++` for computation of multiphase flows in porous media [6].

We mention some similar projects. A general PDE software tool *Diffpack* is an object oriented development framework for the solution PDE [8]. The toolbox UG is a framework for unstructured grid computations. A number of applications of this tool for computations of complex fluid flows in porous media are described in [1, 5].

Initially `MfsolverC++` was created as a sequential application [6]. Its key features are the following:

- Written in C++, enabling portability of the code;
- Written using OOP, thus giving clear and maintainable code;
- Its architecture is similar to *DiffPack* package [8].

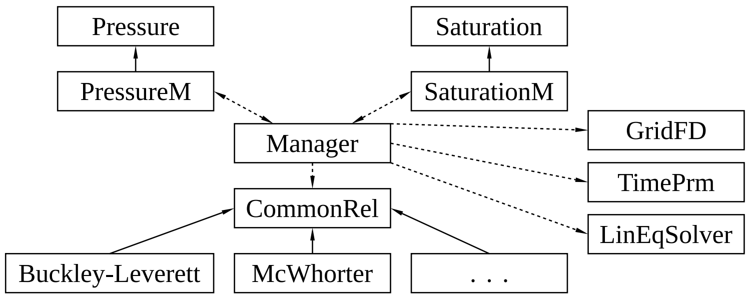

Figure 1. `MfsolverC++` class diagram.

The diagram of `MfsolverC++` classes is presented in Figure 1. It consists of the following main classes:

Pressure equation solvers. Here Pressure class and its descendants are implemented. They contain various pressure equation solution methods.

Saturation equation solvers. Here Saturation class and its descendants are implemented. They contain various saturation equation solution methods.

System properties. Common set of relations (constitutive relationships, model definitions, etc.) are collected in class CommonRel.

Manager. Manager class acts as the solver class for the whole PDEs system. This class contains two way pointers to the subclasses for solving the pressure and saturation equations.

3. Parallel Arrays Library *ParSol*

The initial goal of creating *ParSol* parallel array library was to provide a tool for parallelization of `MfsolverC++`. However, *ParSol* is designed with intention to be used in much wider range of applications.

ParSol has the following key features:

1. It is written in C++, using template mechanism and OOP.
2. It uses MPI [4] for underlying communications.
3. It operates similar to HPF [7].
   C++ becomes most popular programming language for developing many applications in computational science and modelling, thus we need similar tools as were developed for Fortran codes.
   Parallelization of data parallel algorithms is done in few simple steps. The difference from HPF is that the user must specify explicitly the stencil of the grid used in the algorithm. Such information is required to implement additional data communication part of parallel algorithm. The other requirement is that all computations should not depend on the order in which array points are processed (for example the Jacobi iterative method satisfies this requirement, but the Seidel iterative method can not be parallelized with *ParSol*.)

The *ParSol* arrays have many useful features, e.g. operations with arrays, possibility to compute various norms of vectors, scalar products and etc. Thus it is recommended to use array operations provided by ParSol wherever it is possible. It frees programmer from implementing simple tasks, allowing to concentrate on problem solving, and makes the code cleaner and in many case more efficient.

Features 1 and 2, if exploited properly, allow to use *ParSol* on wide range of computer platforms and compilers. Currently, *ParSol* was tested in the following cases:

- MS-Windows OS, MS Visual C++ 6.0, MPICH MPI implementation;

- Linux OS, gcc 3.3.2 and higher versions, LAM MPI implementation (see, `http://vilkas.vtu.lt`);
- IBM SP4 supercomputer (AIX), Visual Age C++ compiler, IBM MPI implementation (see, `http://www.cineca.it`).

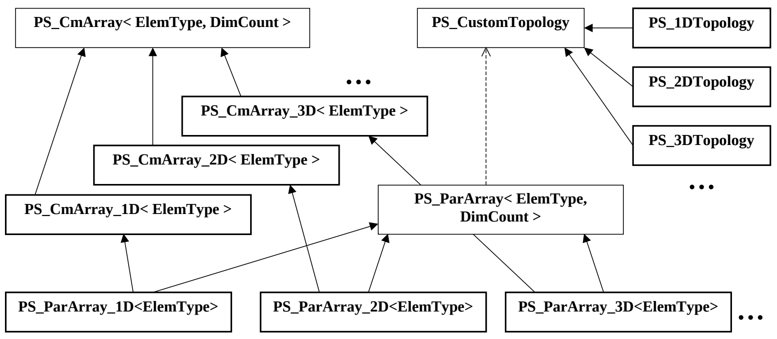
PS_CmArray< ElemType, DimCount >
PS_CustomTopology
PS_1DTopology
PS_2DTopology
PS_3DTopology
PS_CmArray_3D< ElemType >
PS_CmArray_2D< ElemType >
PS_CmArray_1D< ElemType >
PS_ParArray< ElemType, DimCount >
PS_ParArray_1D<ElemType>
PS_ParArray_2D<ElemType>
PS_ParArray_3D<ElemType>
...

Figure 2. ParSol library class diagram.

ParSol class diagram is shown on Figure 2. The main elements of the library are the following:

Sequential array classes. These are the classes to be used instead of native `C/C++` arrays. No MPI or other libraries, except *ParSol* itself, is necessary to use them. Comparing to native `C/C++` arrays, *ParSol* sequential arrays have a number of advantages for implementation of mathematical algorithms, such as virtual indexes, built-in array operations, automated management of dynamically allocated memory.
The main functionality resides in template class `PS_CmArray`. However, general functionality requires interface complexity. So children are derived for special cases (e.g., 1D, 2D, 3D arrays), that provide intuitive and user-friendly interface. It is recommended for end-user to use those classes whenever possible.

Parallelization and parallel array classes. If parallel arrays are used in place of sequential ones, it is natural to make them the descendants of appropriate sequential arrays, adding parallelization code to the sequential array functionality. We note that parallelization is similar for different kinds of arrays. So parallelization code is localized in class `PS_ParArray`, and is used in parallel array classes by multiple inheritance.

Topology classes. The purpose of these classes is to ensure that all processes are in proper order for parallel array functionality. In HPF, this functionality is performed by special directives. All the general code resides in `PS_CustomTopology` class. As with sequential array classes, there are also descendants for some special cases (`PS_`{1,2,3}`DTopology`), which provide end user with more friendly interface.

Stencil classes. A stencil is determined depending on what computational scheme is used. This information should be provided by the user of *ParSol* tool. Based on stencil, a different amount of information needs to be exchanged among neighbours processors. Hence, stencil information is required for parallel arrays to operate properly.

To use *ParSol*, a programmer must develop his/her sequential application in C++, only *ParSol* arrays must be used to store computational data. Parallelization of such sequential program takes the following steps:

1. Replace includes of sequential headers with parallel ones (e.g. the header file `PS_CommonArray.h` should be replaced by `PS_ParallelArray.h`).
2. Replace sequential classes with their parallel versions in variable declaration part of the code.
3. Add MPI initialization code, i.e. one line at the beginning of the program.
4. Add topology initialization code (in its simplest case, it is sufficient to add one line at the beginning of the program).
5. Specify when processors – array neighbours should exchange data.
6. MPI library should be linked during building process.

4. Results of Computational Experiments

The parallelization of `MfsolverC+` is still in developing stage. Till now we have tested *ParSol* on a set of benchmarks designed to measure performance of several components and parts crucial for efficient performance of parallel version of `MfsolverC+`.

A nonlinear 2D diffusion problem was approximated by the explicit Euler scheme. A detailed description of the problem is given in [3]. In Table 1, the results of testing *ParSol* on PC cluster "Vilkas" of Vilnius Gediminas technical university are presented. PC clusters are known for their high communication latency thus the efficiency of parallel algorithms can be reduced for small computational problems when communication costs are relatively very high.

The parallelization of the same algorithm is much more effective on SP4 computer where communication costs are smaller. The results are presented in Table 2.

In the second test we solved by the Conjugate Gradient method a system of linear equations which was obtained after the discretization of the three dimensional Poisson problem by the finite–volume method. Table 3 presents experimental speedup $S_p(n)$ and efficiency $E_p(n)$ values for solving problems of different size using the diagonal preconditioner. Computations were performed on PC cluster "Vilkas".

Scalability analysis of parallel CG method and more computational results with different preconditioners are presented in [2].

Table 1. The speedup and efficiency for explicit Euler algorithm on PC cluster.

p	$S_p(160)$	$E_p(160)$	$S_p(240)$	$E_p(240)$	$S_p(320)$	$E_p(320)$
2	1.56	0.780	1.76	0.880	1.87	0.934
4	2.36	0.590	3.00	0.750	3.45	0.862
6	2.78	0.463	3.93	0.655	4.77	0.795
8	2.95	0.369	4.69	0.585	5.88	0.735
9	3.16	0.351	5.04	0.560	6.28	0.698
11	3.33	0.303	5.50	0.500	7.09	0.644
12	3.35	0.279	5.64	0.470	7.47	0.623
15	3.39	0.226	6.38	0.425	8.56	0.571

Table 2. The speedup and efficiency for the explicit Euler algorithm on SP4.

p	$S_p(80)$	$E_p(80)$	$S_p(160)$	$E_p(160)$	$S_p(320)$	$E_p(320)$
2	1.975	0.988	1.984	0.992	2.004	1.002
3	2.794	0.931	2.950	0.985	2.970	0.990
4	3.741	0.935	3.928	0.982	3.986	0.996
6	5.168	0.861	5.463	0.910	5.916	0.986
8	6.766	0.846	7.293	0.911	7.831	0.979
9	6.784	0.754	7.604	0.845	8.467	0.941
12	8.701	0.725	10.19	0.849	11.216	0.934
16	10.84	0.677	12.75	0.797	15.041	0.940
24	14.18	0.591	18.24	0.760	21.961	0.915

5. Conclusions

Application of *ParSol* tool for parallelization of `MfsolverC++` still requires some restructuring of solver's code. However we have shown that essential

Table 3. The speedup and efficiency of the CG algorithm on PC cluster.

p	Iterations	Size	T_p	S_p	E_p
1	188	100	24.10		
2	188	100	13.22	1.82	0.911
4	188	100	6.65	3.63	0.906
8	188	100	4.03	5.97	0.747
1	350	200	366.54		
2	350	200	185.51	1.98	0.988
4	350	200	94.99	3.86	0.965
8	350	200	51.58	7.11	0.888
4	453	300	407.57		
8	453	300	215.60		

parts of the solver, i.e. solvers for saturation and pressure equations (which are convection – diffusion and elliptic equations) can be successfully parallelized by using *ParSol* tool.

References

[1] P. Bastian, K. Birken, S. Lang, K. Johannsen, N. Neuss, H. Rentz-Reichert and C. Wieners. UG: A flexible software toolbox for solving PDE. *Computing and Visualization in Science*, **1**, 27 – 40, 1997.

[2] R. Ciegis. Analysis of parallel preconditioned conjugate gradient algorithms. *Informatica*, **15**(2), 155 – 172, 2005.

[3] R. Ciegis, A. Jakušev, A. Krylovas and O. Suboč. Parallel algorithms for solution of nonlinear diffusion problems in image smoothing. *Math. Modelling and Analysis*, **10**(2), 155 – 172, 2005.

[4] W. Gropp, E. Lusk and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface.* The MIT Press, Cambridge, Massachusetts, London, 1995.

[5] R. Helmig. *Multiphase Flow and Transport Processes in the Subsurface – A Contribution to the Modelling of Hydrosystems.* Springer – Verlag, 1997.

[6] A. Jakušev and V. Starikovičius. Multiphase flow problem solver and its application for multidimensional problems. *Lithuanian Mathematical Journal*, **44**, 634–638, 2004. (in Lithuanian)

[7] C. Koelbel, D. Loveman, R. Schreiber, G. Steele and M. Zosel. *The High Performance Fortran handbook.* MIT Press, USA, 1994.

[8] H.P. Langtangen. *Computational Partial Differential Equations. Numerical Methods and Diffpack Programming.* Springer, Berlin, 2002.