

# Parallelization of Linear Algebra Algorithms Using ParSol Library of Mathematical Objects

Alexander Jakušev, Raimondas Čiegis, Inga Laukaitytė, and Vyacheslav Trofimov

**Abstract** The linear algebra problems are an important part of many algorithms, such as numerical solution of PDE systems. In fact, up to 80% or even more of computing time in this kind of algorithms is spent for linear algebra tasks. The parallelization of such solvers is the key for parallelization of many advanced algorithms. The mathematical objects library ParSol not only implements some important linear algebra objects in C++, but also allows for semiautomatic parallelization of data parallel and linear algebra algorithms, similar to High Performance Fortran (HPF). ParSol library is applied to implement the finite difference scheme used to solve numerically a system of PDEs describing a nonlinear interaction of two counter-propagating laser waves. Results of computational experiments are presented.

## 1 Introduction

The numerical solution of PDEs and systems of PDEs is the basis of mathematical modeling and simulation of various important industrial, technological, and engineering problems. The requirements for the size of the discrete problem and speed of its solution are growing every day, and parallelization of PDE solvers is one of the ways to meet those requirements. All the most popular PDE solvers today have parallel versions available. However, new methods and tools to solve PDEs and systems of PDEs appear constantly, and parallelization of these solvers becomes a very important task. Such a goal becomes even more urgent as multicore computers and clusters of such computers are used practically in all companies and universities.

---

Alexander Jakušev · Raimondas Čiegis · Inga Laukaitytė  
Vilnius Gediminas Technical University, Saulėtekio al. 11, LT-10223, Vilnius, Lithuania  
e-mail: {alexj · rc · Inga.Laukaityte}@fm.vgtu.lt

Vyacheslav Trofimov  
M. V. Lomonosov Moscow State University, Vorob'evy gory, 119992, Russia  
e-mail: vatro@cs.msu.su

Let us consider parallelization details of such well-known PDE solvers as Clawpack [13], Diffpack [12], DUNE [4], OpenFOAM [16], PETSc [1], and UG [3]. All of them share common features:

1. Parallelization is implemented on discretization and linear algebra layers, where parallel versions of linear algebra data structures and algorithms are provided. Data parallel parallelization model is used in all cases.
2. Using the PDE solvers, the user can produce parallel version of the code semi-automatically, however, it is required to specify how data is divided among processors and to define a stencil used in the discretization step.
3. The parallelization code is usually tightly connected with the rest of the solver, making it practically impossible to reuse the code for the parallelization of another PDE solver. However, widespread low-level parallelization libraries, especially MPI [17], are used for actual interprocess communications.

All these features, except for the tight connection to the rest of the code, are important and are used in our tool. However, the tight integration with the solvers prevents the reuse of parallelization code. For quick and efficient parallelization of various PDE solvers, a specific library should be used.

Many parallelization libraries and standards exist; they may be grouped by parallelization model they are using. The following models are the most popular ones:

*Multithreading.* This model is best suited for the Symmetric Multiprocessing (SMP) type of computers. OpenMP standard is a good representative of this model. However, this model is not suited for computer clusters and distributed parallel memory computers.

*Message passing.* This model is best suited for systems with distributed memory, such as computer clusters. MPI [17] and PVM [8] standards are good representatives of this model. However, this is a low-level model and it is difficult to use it efficiently.

*Data parallel.* This model is ideal if we want to achieve semi-automatic parallelization quickly. HPF [10] and FreePOOMA are good representatives of this model. However, in its pure form this model is pretty restricted in functionality.

*Global memory.* This parallel programming model may be used for parallelization of almost any kind of algorithms. PETSc [2] and Global Arrays [14] libraries are good representatives of this model. However, the tools implementing this model are difficult to optimize, which may produce inefficient code.

The conclusions above allow us to say that modern parallelization tools are still not perfect for parallelization of PDE solvers. In this chapter, the new parallelization library is described. Its main goal is parallelization of PDE solvers.

Our chapter is organized as follows. In Sect. 2, we describe main principles and details of the implementation of the library. The finite difference scheme for solution of one problem of nonlinear optics is described in Sect. 3. This algorithm is implemented by using ParSol tool. The obtained parallel algorithm is tested on the cluster of PCs in Vilnius Gediminas Technical University (VGTU). Results of computational experiments are presented. Some final conclusions are made in Sect. 4.

## 2 The Principles and Implementation Details of ParSol Library

The ParSol library was designed to overcome some problems of the models mentioned above. It employs the elements of both *data parallel* and *global memory* parallel programming models. From the *data parallel* model, the library takes parallelizable objects and operations with them. For linear algebra, these objects are arrays, vectors, matrices. It is important to note that BLAS level 2 compatible operations are implemented in sequential and parallel modes. The same functionality was previously implemented in HPF and PETSc tools.

In order to overcome the lack of features of the *data parallel* model, ParSol library utilizes some elements of *global memory* model. Like in *global memory* model, arrays and vectors in the library have global address space. However, there are strict limits for the interprocess communications. Any given process can communicate with only a specified set of the other processes, and to exchange with them elements defined by some a priori stencil. Such a situation is typical for many solvers of PDEs. It is well-known that for various difference schemes, only certain neighbour elements have to be accessed. The position of those neighbor elements is described by *stencil* of the grid.

Rather than trying to guess the shape of stencil from the program code, as HPF does, the library asks the programmer to specify the stencil, providing convenient means for that. While requiring some extra work from the programmer, it allows one to specify the exact necessary stencil to reduce the amount of transferred data to the minimum.

### 2.1 Main Classes of ParSol

Next we list the main classes of the ParSol library:

*Arrays* are the basic data structure, which may be automatically parallelized.

*Vectors* are arrays with additional mathematical functionality of linear algebra.

Vectors are more convenient for numerical algorithms, whereas arrays are good for parallelization of algorithms where the data type of arrays does not support linear algebra operations.

*Stencils* are used to provide information about which neighbor elements are needed during computations.

*Matrices* are used in matrix vector multiplications and other operations that arise during solution of linear systems.

*Array elements* are internally stored in 1D array; however, the user is provided with multidimensional element indexes. The index transformations are optimized thus the given interface is very efficient. Also, arrays implement dynamically calculated boundaries, which are adjusted automatically in parallel versions.

*Cyclic arrays* are implemented in a specific way, using additional shadow elements where the data is copied from the opposite side of the array. This increases

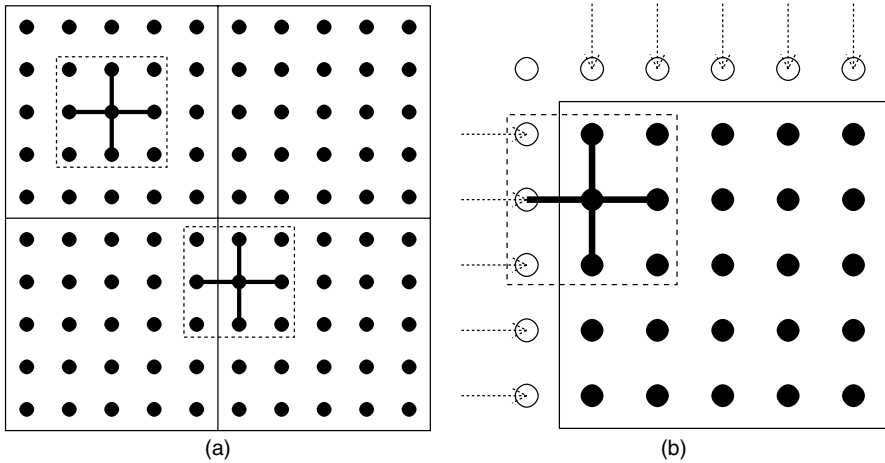
array footprint slightly and requires user to specify data exchange directives even in a sequential version; however, the element access does not suffer, as no additional calculations are necessary.

*ParSol arrays* are implemented as template classes, so it is possible to have arrays of various data types, such as integers, floats or complex numbers.

*Vectors* provide additional functionality on top of the arrays, such as global operations, multiplication by a constant, and calculation of various norms. All these operations make a basis of many popular numerical algorithms used to solve PDEs and systems of such equations.

The library provides the user with both dense and sparse matrices. The dense matrices are stored in 2D array, whereas sparse matrices are stored in the CSR (Compressed Sparse Row) format. We note that the same format is used in PETSc library [2, p. 57]. For sparse matrices, it is possible to estimate a priori the number of preallocated elements if the stencil of the vector is known. The matrix dimensions are defined at the creation stage, using a vector given in the constructor of the matrix (thus matrix inherits a lot of its properties that the matrix will be used in conjunction with).

The parallelization scheme is shown in Fig. 1. The arrays (and thus vectors) are divided among processors depending on the given topology of processors. Similar to the case of stencils, the library provides an interface for setting the topology through special *Topology* classes and the stencil. When data exchange starts, neighbour elements are copied to the shadow area, where they must be read-only. Several methods of data exchange are implemented in ParSol: all-at-once, pair-by-pair, and pair-by-pair-ordered.



**Fig. 1** Transition from sequential to parallel array: (a) the sequential array and its stencil, (b) part of the parallel array for one processor.

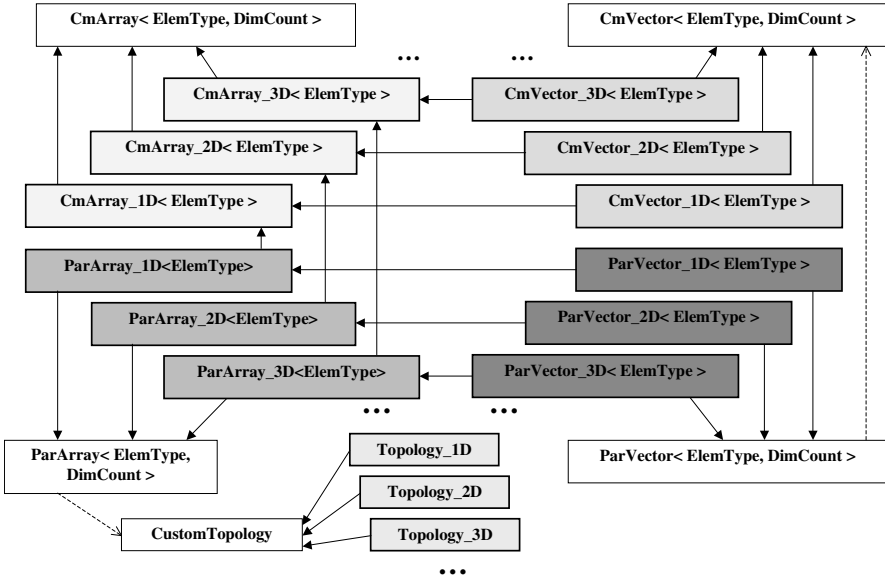


Fig. 2 Class diagram of ParSol arrays and vectors.

## 2.2 Implementation of ParSol

The library has been implemented in C++ [18], using such C++ features as OOP, operator overloading, and template metaprogramming [9]. This implementation is similar to such numerical libraries as Boost++ and FreePOOMA. Only standard C++ features are used, resulting in high portability of the library. As the library uses some advanced C++ features, modern C++ compilers are required.

MPI-1.1 standard is used for the parallel version of the library. The parallel versions of the classes are implemented as children of sequential analogical classes, ensuring that the code should not be changed much during parallelization. The size of the library code is  $\approx 20000$  lines, with  $\approx 10000$  more lines for regression tests. ParSol consists of more than 70 various classes, a subset of which is shown in Fig. 2. The library is available on the Internet at <http://techmat.vgtu.lt/~alexj/ParSol/>.

## 3 Parallel Algorithm for Simulation of Counter-propagating Laser Beams

In the domain (see Fig. 3)

$$D = \{0 \leq z \leq L_z, \quad 0 \leq x_k \leq L_x, \quad k = 1, 2\}$$



**Fig. 3** Scheme of the interaction of counter-propagating laser pulses.

dimensionless equations and boundary conditions describing the interaction of two counter-propagating laser beams are given by [15]:

$$\frac{\partial A^+}{\partial t} + \frac{\partial A^+}{\partial z} + i \sum_{k=1}^2 D_k \frac{\partial^2 A^+}{\partial x_k^2} + i\gamma(0.5|A^+|^2 + |A^-|^2)A^+ = 0,$$

$$\frac{\partial A^-}{\partial t} - \frac{\partial A^-}{\partial z} + i \sum_{k=1}^2 D_k \frac{\partial^2 A^-}{\partial x_k^2} + i\gamma(0.5|A^-|^2 + |A^+|^2)A^- = 0,$$

$$A^+(z=0, x_1, x_2, t) = A_0(t) \exp\left(-\sum_{k=1}^2 \frac{(x_k - x_{ck})^{m_k}}{r_{pk}}\right),$$

$$A^-(z=L_z, x_1, x_2, t) = A^+(z=L_z, x_1, x_2, t)R_0 \\ \times \left(1 - \exp\left(-\sum_{k=1}^2 \frac{(x_k - x_{mk})^{q_k}}{R_{ak}}\right)\right) \exp\left(i \sum_{k=1}^2 \frac{(x_k - x_{mk})^2}{R_{mk}}\right).$$

Here  $A^\pm$  are complex amplitudes of counter-propagating pulses,  $\gamma$  characterizes the nonlinear interaction of laser pulses,  $x_{ck}$  are coordinates of the beam center,  $r_{pk}$  are radius of input beam on the transverse coordinates, and  $A_0(t)$  is a temporal dependence of input laser pulses. In the boundary conditions,  $R_0$  is the reflection coefficient of the mirror,  $R_{ak}$  are the radius of the hole along the transverse coordinates,  $x_{mk}$  are coordinates of the hole center, and  $R_{mk}$  characterize curvature of the mirror.

At the initial time moment, the amplitudes of laser pulses are equal to zero

$$A^\pm(z, x_1, x_2, 0) = 0, \quad (z, x_1, x_2) \in D.$$

Boundary conditions along transverse coordinates are equal to zero.

We note that in previous papers, the mathematical model was restricted to 1D transverse coordinate case.

### 3.1 Invariants of the Solution

It is well-known that the solution of the given system satisfies some important invariants [19]. Here we will consider only one of such invariants. Multiplying differential

equations by  $(A^\pm)^*$  and integrating over

$$Q(z, t, h) = [z - h, z] \times [0, L_x] \times [0, L_x] \times [t - h, t],$$

we prove that the full energy of each laser pulse is conserved during propagation along the characteristic directions  $z \pm t$ :

$$\|A^+(z, t)\| = \|A^+(z - h, t - h)\|, \quad \|A^-(z - h, t)\| = \|A^-(z, t - h)\|, \quad (1)$$

where the  $L_2$  norm is defined as

$$\|A(z, t)\|^2 = \int_0^{L_x} \int_0^{L_x} |A|^2 dx_1 dx_2.$$

This invariant describes a very important feature of the solution and therefore it is important to guarantee that the discrete analogs are satisfied for the numerical solution. In many cases, this helps to prove the existence and convergence of the discrete solution.

### 3.2 Finite Difference Scheme

In the domain  $[0, T] \times D$ , we introduce a uniform grid  $\Omega = \Omega_t \times \Omega_z \times \Omega_x$ , where

$$\Omega_t = \{t^n = nh_t, \quad n = 0, \dots, N\}, \quad \Omega_z = \{z_j = jh_z, \quad j = 0, \dots, J\},$$

$$\Omega_x = \{(x_{1l}, x_{2m}), \quad x_{km} = mh_x, \quad k = 1, 2, \quad m = 0, \dots, M\}.$$

In order to approximate the transport part of the differential equations by using the finite differences along the characteristics  $z \pm t$ , we take  $h_t = h_z$ .

Let us denote discrete functions, defined on the grid  $\Omega$  by

$$E_{j,lm}^{\pm,n} = E^\pm(z_j, x_{1l}, x_{2m}, t^n).$$

We also will use the following operators:  $\beta(E, W) = \gamma(0.5|E|^2 + |W|^2)$ ,

$$\begin{aligned} \bar{E}^+ &= \frac{E_j^{+,n} + E_{j-1}^{+,n-1}}{2}, \quad \bar{E}^- = \frac{E_{j-1}^{-,n} + E_j^{-,n-1}}{2}, \\ \mathcal{D}E_{j,kl} &= D_1 \frac{E_{j,l+1,m} - 2E_{j,lm} + E_{j,l-1,m}}{h_x^2} + D_2 \frac{E_{j,l,m+1} - 2E_{j,lm} + E_{j,l,m-1}}{h_x^2}. \end{aligned}$$

Then the system of differential equations is approximated by the following finite difference scheme

$$\frac{E_j^{+,n} - E_{j-1}^{+,n-1}}{h_t} + i\mathcal{D}\bar{E}_j^+ + i\beta(\bar{E}_j^+, \bar{E}_j^-)\bar{E}_j^+ = 0, \quad (2)$$

$$\frac{E_{j-1}^{-,n} - E_j^{+,n-1}}{h_t} + i\mathcal{D}\bar{E}_j^- + i\beta(\bar{E}_j^-, \bar{E}_j^+)\bar{E}_j^- = 0$$

with corresponding boundary and initial conditions.

We will prove that this scheme is conservative, i.e., the discrete invariants are satisfied for its solution. Let us define the scalar product and  $L_2$  norm of the discrete functions as

$$(U, V) = \sum_{l=1}^{M-1} \sum_{m=1}^{M-1} U_{lm} V_{lm}^* h_x^2, \quad \|U\|^2 = (U, U).$$

Taking a scalar product of equations (2) by  $\bar{E}^+$  and  $\bar{E}^-$  respectively and considering the real parts of the equations, we get that the discrete analogs of the invariants (1) are satisfied

$$\|E_j^{+,n}\| = \|E_{j-1}^{+,n-1}\|, \quad \|E_{j-1}^{-,n}\| = \|E_j^{-,n-1}\|, \quad j = 1, \dots, J.$$

Finite difference scheme is nonlinear, thus at each time level  $t^n$  and for each  $j = 1, \dots, J$  the simple linearization algorithm is applied:

$$\frac{E_j^{+,n,s+1} - E_{j-1}^{+,n-1}}{h_t} + i\mathcal{D}\bar{E}_j^{+,s+1} + i\beta(\bar{E}_j^{+,s}, \bar{E}_j^{-,s})\bar{E}_j^{+,s} = 0, \quad (3)$$

$$\frac{E_{j-1}^{-,n,s+1} - E_j^{+,n-1}}{h_t} + i\mathcal{D}\bar{E}_j^{-,s+1} + i\beta(\bar{E}_j^{-,s}, \bar{E}_j^{+,s})\bar{E}_j^{-,s} = 0,$$

where the following notation is used

$$\bar{E}^{+,s} = \frac{E_j^{+,n,s} + E_{j-1}^{+,n-1}}{2}, \quad \bar{E}^{-,s} = \frac{E_{j-1}^{-,n,s} + E_j^{+,n-1}}{2}.$$

The iterations are continued until the convergence criterion

$$\max_{lm} |E_{lm}^{\pm,s+1} - E_{lm}^{\pm,s}| < \varepsilon_1 \max_{lm} |E_{lm}^{\pm,s}| + \varepsilon_2, \quad \varepsilon_1, \varepsilon_2 > 0$$

is valid. It is important to note that iterative scheme also satisfies the discrete invariant

$$\|E_j^{+,n,s}\| = \|E_{j-1}^{+,n-1}\|, \quad \|E_{j-1}^{-,n,s}\| = \|E_j^{-,n-1}\|, \quad s \geq 1.$$

For each iteration defined by (3) two systems of linear equations must be solved to find vectors  $E_j^{+,n,s+1}$  and  $E_{j-1}^{-,n,s+1}$ . This is done by using the 2D FFT algorithm.



### 3.3 Parallel Algorithm

The finite difference scheme, which is constructed in the previous section, uses the structured grid, and the complexity of computations at each node of the grid is approximately the same (it depends on the number of iterations used to solve a nonlinear discrete problem for each  $z_j$ ). The parallelization of such algorithms can be done by using domain decomposition (DD) paradigm [11], and ParSol is exactly targeted for such algorithms.

In this chapter, we apply the 1D block domain decomposition algorithm, decomposing the grid only in  $z$  direction. Such a strategy enables us to use a sequential version of the FFT algorithm for solution of the 2D linear systems with respect to  $(x, y)$  coordinates.

This parallel algorithm is generated semiautomatically by ParSol. The parallel vectors, which are used to store discrete PDE solutions, are created by specifying three main attributes:

- the dimension of the parallel vector is 3D;
- the topology of processors is 1D and only  $z$  coordinate is distributed;
- the 1D grid stencil is defined by the points  $(z_{j-1}, z_j, z_{j+1})$ .

Thus in order to implement the computational algorithm, the  $k$ th processor ( $k = 0, 1, \dots, p$ ) defines its subgrid as well as its ghost points  $\Omega(k)$ , where

$$\begin{aligned}\Omega(k) &= \{(z_j, x_{1l}, x_{2m}), z_j \in \Omega_z(k), (x_{1l}, x_{2m}) \in \Omega_x\}, \\ \Omega_z(k) &= \{z_j : \tilde{j}_L(k) \leq j \leq \tilde{j}_R(k)\}, \\ \tilde{j}_L(k) &= \max(j_L(k) - 1, 0), \quad \tilde{j}_R(k) = \min(j_R(k) + 1, J).\end{aligned}$$

At each time step  $t^n$  and for each  $j = 1, 2, \dots, J$ , the processors must exchange some information for ghost points values. As the computations move along the characteristics  $z \pm t$ , only a half of the full data on ghost points is required to be exchanged. Thus the  $k$ th processor

- sends to  $(k+1)$ th processor vector  $E_{j_R, \cdot}^{+,n}$  and receives from it vector  $E_{\tilde{j}_R, \cdot}^{-,n}$ ,
- sends to  $(k-1)$ th processor vector  $E_{j_L, \cdot}^{-,n}$  and receives from it vector  $E_{\tilde{j}_L, \cdot}^{+,n}$ .

Obviously, if  $k = 0$  or  $k = (p-1)$ , then a part of communications is not done. In ParSol, such an optimized communication algorithm is obtained by defining temporal reduced stencils for vectors  $E^+$  and  $E^-$ ; they contain ghost points only in the required directions but not in both.

Next we present a simple scalability analysis of the developed parallel algorithm. The complexity of the serial algorithm for one time step is given by  $W = KJ(M+1)^2(\gamma_1 + \gamma_2 \log M)$ , where  $\gamma_1(M+1)^2$  estimates the CPU time required to compute all coefficients of the finite-difference scheme,  $\gamma_2(M+1)^2 \log M$  defines the costs of FFT algorithm, and  $K$  is the averaged number of iterations done at one time step.

Let us assume that  $p$  processors are used. Then the computational complexity of parallel algorithm depends on the size of the largest local grid part given to one processor. It is equal to

$$T_{p,\text{comp}} = \max_{0 \leq k < p} K(k) (\lceil (J+1)/p \rceil + 1) (M+1)^2 (\gamma_1 + \gamma_2 \log M),$$

where  $\lceil x \rceil$  denotes a smallest integer number larger than or equal to  $x$ . This formula includes costs of extra computations involving ghost points.

Data communication time is given by  $T_{p,\text{comm}} = 2(\alpha + \beta(M+1)^2)$ ; here  $\alpha$  is the message startup time and  $\beta$  is the time required to send one element of data. We assume that communication between neighbour processors is done in parallel. Thus the total complexity of the parallel algorithm is given by

$$T_p = \max_{0 \leq k < p} K(k) (\lceil (J+1)/p \rceil + 1) (M+1)^2 (\gamma_1 + \gamma_2 \log M) + 2(\alpha + \beta(M+1)^2).$$

### 3.4 Results of Computational Experiments

The parallel code was tested on the cluster of PCs at Vilnius Gediminas Technical University. It consists of Pentium 4 processors (3.2GHz, level 1 cache 16kB, level 2 cache 1MB) interconnected via Gigabit Smart Switch (<http://vilkas.vgtu.lt>). Obtained performance results are presented in Table 1. Here for each number of processors  $p$ , the coefficients of the algorithmic speed up  $S_p = T_1/T_p$  and efficiency  $E_p = S_p/p$  are presented. The size of the discrete problem is  $M = 31$  and  $J = 125$ .

More applications of the developed parallelization tool ParSol are described in [5, 6, 7].

## 4 Conclusions

The new tool of parallel linear algebra objects ParSol employs elements of both data parallel and global memory parallel programming models. In the current version the main objects, i.e., arrays, vectors and matrices, are targeted for structured grids. The interface for 1D, 2D, and 3D linear algebra objects is done. The algorithm

**Table 1** Results of computational experiments on Vilkas cluster.

	$M$	$J$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 12$
$S_p$	31	125	1.0	1.87	3.46	5.86	7.83
$E_p$	31	125	1.0	0.94	0.87	0.73	0.65

implemented by ParSol objects is parallelized semi-automatically, by specifying the stencil of the grid, topology of processors, and data communication points in the algorithm.

ParSol is applied to parallelize one numerical algorithm, which is developed to simulate the nonlinear interaction of two counter-propagating nonlinear optical beams. It is shown how the specific features of the algorithm can be taken into account to minimize the amount of data communicated between processors. The results of numerical experiments show a good efficiency of the obtained parallel numerical algorithm.

**Acknowledgments** R. Čiegis and I. Laukaitytė were supported by the Lithuanian State Science and Studies Foundation within the project on B-03/2007 “Global optimization of complex systems using high performance computing and GRID technologies”.

## References

1. Akcelik, V., Biros, G., Ghattas, O., Hill, J. et al.: Frontiers of parallel computing. In: M. Heroux, P. Raghaven, H. Simon (eds.) *Parallel Algorithms for PDE-Constrained Optimization*. SIAM, Philadelphia (2006)
2. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., Curfman McInnes, L., Smith, B.F., Zhang, H.: PETSc user manual. ANL-95/11 - Revision 2.1.5. Argonne National Laboratory (2004)
3. Bastian, P., Birken, K., Johannsen, K., Lang, S. et al.: A parallel software-platform for solving problems of partial differential equations using unstructured grids and adaptive multigrid methods. In: W. Jage, E. Krause (eds.) *High Performance Computing in Science and Engineering*, pp. 326–339. Springer, New York (1999)
4. Blatt, M., Bastian, P.: The iterative solver template library. In: B. Kågström, E. Elmroth, J. Dongarra, J. Wasniewski (eds.) *Applied Parallel Computing: State of the Art in Scientific Computing, Lecture Notes in Scientific Computing*, vol. 4699, pp. 666–675. Springer, Berlin Heidelberg New York (2007)
5. Čiegis, Raim, Čiegis, Rem., Jakušev, A., Šaltenienė, G.: Parallel variational iterative algorithms for solution of linear systems. *Mathematical Modelling and Analysis* **12**(1), 1–16 (2007)
6. Čiegis, R., Jakušev, A., Krylovas, A., Suboč, O.: Parallel algorithms for solution of nonlinear diffusion problems in image smoothing. *Mathematical Modelling and Analysis* **10**(2), 155–172 (2005)
7. Čiegis, R., Jakušev, A., Starikovičius, V.: Parallel tool for solution of multiphase flow problems. In: R. Wyrzykowski, J. Dongarra, N. Meyer, J. Wasniewski (eds.) *Sixth International conference on Parallel Processing and Applied Mathematics*. Poznan, Poland, September 10–14, 2005, *Lecture Notes in Computer Science*, vol. 3911, pp. 312–319. Springer, Berlin Heidelberg New York (2006)
8. Geist, A., Beguelin, A., Dongarra, J. et al.: PVM: Parallel Virtual Machine. A User’s Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA (1994)
9. Jakušev, A.: Application of template metaprogramming technologies to improve the efficiency of parallel arrays. *Mathematical Modelling and Analysis* **12**(1), 71–79 (2007)
10. Koelbel, C.H., Loveman, D.B., Schreiber, R.S., Steele, G.L., Zosel, M.E.: *The High Performance Fortran Handbook*. The MIT Press, Cambridge, MA (1994)
11. Kumar, V., Grama, A., Gupta, A., Karypis, G.: *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City (1994)

12. Langtangen, H.P.: Computational Partial Differential Equations — Numerical Methods and Diffpack Programming, *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, New York (1999)
13. Le Veque, R.: Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, Cambridge, UK (2002)
14. Nieplocha, J., Palmer, B., Tipparaju, V., Krishnan, M., Trease, H., Apra, E.: Advances, applications and performance of the Global Arrays shared memory programming toolkit. *International Journal of High Performance Computing Applications* **20**(2), 203–231 (2006)
15. Nikitenko, K.Y., Trofimov, V.A.: Optical bistability based on nonlinear oblique reflection of light beams from a screen with an aperture on its axis. *Quantum Electronics* **29**(2), 147–150 (1999)
16. OpenFOAM: The Open Source CFD Toolbox. URL <http://www.opencfd.co.uk/openfoam/>
17. Snir, M., Otto, S., Hus-Lederman, S., Walker, D., Dongarra, J.: MPI. The Complete Reference. The MIT Press, Cambridge, Reading, MA (1995)
18. Stroustrup, B.: The C++ Programming Language. Addison-Wesley, MA (1997)
19. Tereshin, E.B., Trofimov, V.A.: Conservative finite difference scheme for the problem of propagation of a femtosecond pulse in a photonic crystal with combined nonlinearity. *Comput. Math. and Mathematical Physics* **46**(12), 2154–2165 (2006)

Parallel Scientific Computing and Optimization

Advances and Applications

Ciegis, R.; Henty, D.; Kågström, B.; Žilinskas, J. (Eds.)

2009, XXIV, 274 p., Hardcover

ISBN: 978-0-387-09706-0