Mathematical Modelling and Analysis
Volume 16 Number 2, June 2011, 326–341
Doi:10.3846/13926292.2011.582591

www.tandf.co.uk/journals/TMMA
Publisher: Taylor&Francis and VGTU
Online ISSN: 1648-3510
Print ISSN: 1392-6292

A Parallel Solver for the Design of Oil Filters

Vadimas Starikovičius[a], Raimondas Čiegis[a] and Oleg Iliev[b]

[a] *Vilnius Gediminas Technical University*
Saulėtekio al. 11, LT-10223 Vilnius, Lithuania
[b] *Fraunhofer ITWM*
Fraunhofer-Platz 1, D-67663 Kaiserslautern, Germany
E-mail(*corresp.*): vadimas.starikovicius@vgtu.lt
E-mail: raimondas.ciegis@vgtu.lt
E-mail: oleg.iliev@itwm.fraunhofer.de

Received March 10, 2011; revised April 15, 2011; published online May 1, 2011

Abstract. Nowadays, it is widely recognized that computer simulation plays a crucial role in designing oil filters used in the automotive industry. However, even a single direct simulation of the flow usually requires significant computational resources. Thus, it is obvious that solution of optimization problems is only feasible using parallel computers and algorithms.

In this paper, we present a general master-slave parallel template, which was specially designed for the easy integration of direct parallel solvers into a parallel optimization tool. We show how an already existing direct solver for the 3D simulation of flow through the oil filter is integrated into our template to obtain a parallel optimization solver. Some capabilities and performance of this solver are demonstrated by solving geometry optimization problem of a filter element.

Keywords: parallel algorithms, Navier–Stokes–Brinkmann model, oil filters.

AMS Subject Classification: 76S05; 65Y05; 68W10.

1 Introduction

The performance of oil filters used in automotive engines and other areas can be significantly improved using computer simulation as essential component of the design process. Computer simulations provide engineers with the detailed information about the flow field through the filter. Moreover, numerical simulations, when they can be performed, allow a significant reduction in time and costs for the design of new filter elements.

In this work, we build upon our previous studies on the mathematical modeling and numerical simulation of flow through oil filters [8, 13, 17, 21, 22]. Mathematical models, efficient numerical schemes and parallel algorithms were successfully developed in close collaboration with the oil filter manufacturing

company, namely IBS Filtran. However, even a single direct simulation of the flow often requires significant computational resources. Generally, 30–36 simulations with the different flow rates and viscosities have to be performed for each geometry to evaluate the performance of the filter at different flow conditions. Thus, any kind of geometry optimization (design of the filter housing or/and filtering medium) is a computationally challenging problem, which can only be tackled using parallel computers and algorithms.

The aim of this paper is to present a parallel solver for the optimization of oil filters. It is built upon existing direct solver, which is also parallel [22]. For this purpose, we have developed a general master-slave parallel template, which was specially designed for the easy integration of direct parallel solvers with complicated input and output.

Obviously, in our days the ideas of simulation-driven design are getting a lot of attention from the software developers and manufacturers themselves. Commercial software products from such companies as Altair ProductDesign, FRIENDSHIP SYSTEMS are employed by the world-known manufacturers of aircrafts, cars, ships, etc. All such software tools are based on integration of geometry design with simulation through some kind of optimization. In general, the goal of any global optimization algorithm is to find the best possible element in a search space according to a given objective function [16]. The usage of global optimization algorithms for solution of industrial problems requires to take into account three important features. First, in order to compute one value of the objective function usually we should solve a discrete problem (e.g. a system of 3D discrete nonlinear PDEs), which requires big CPU time costs. Second, there is no possibility to get information on the properties of the objective function, such as its gradient or at least a reasonable estimation of the Lipschitz constant. Third, we are not concentrated on finding the exact global minimum, it is sufficient to get a solution which improves the known engineering approximation. Thus in engineering applications, most popular optimization algorithms are based on specific classes of algorithms, such as the branch and bound (BB) strategy with simple heuristics for the definition of bounds (see, papers on implementations of parallel BB templates [3, 4]), genetic, ant colony type algorithms [11, 19] or the simulated annealing method [23]. If the dimension of a search space is small, then the Nelder–Mead simplex method can be used.

In this work, we build upon our previously developed master-slave (MS) templates for the solution of optimization problems. Application of one of those in civil engineering is described in [2]. The other two level parallel MS template was successfully used to implement optimization algorithms for simulation and optimization of electrical cables in automotive industry [10]. One of the main tasks there is to determine optimal conductor cross-sections in the bundles of electric cables in order to minimize the total weight of cables. A simple heuristic algorithm based on the greedy type search method was successfully used as an optimization technique. More information on these tools is given on the web page of GridGlobOpt project at `http://www.gridglobopt.vgtu.lt/`.

The main focus of this paper is not on the optimization step but on the efficiency of two level parallel algorithms for the parallel optimization tools,

when the master-slave and domain decomposition paradigms are combined.

The rest of the paper is organized as follows. In Section 2, we describe the direct solver for the 3D simulation of flow through the oil filter. Mathematical model, numerical schemes and parallel algorithms are briefly discussed with the references to our previous works. The general master-slave parallel template is presented in Section 3. We show how the direct solver is integrated into our new template to obtain a parallel optimization solver. We investigate the efficiency of two level parallel algorithm on the modern parallel architecture. In Section 4, some capabilities of the obtained solver are demonstrated by solving geometry optimization problem of a filter element. Finally, some conclusions are given in Section 5.

2 Direct Problem Solver

In this section we present a formulation of the mathematical model describing flow through the oil filter. The approximation of the differential model is done by using the finite volume method, and a parallel algorithm is constructed by using the domain decomposition method (for details, see [8, 22]).

2.1 Mathematical model

In the case of liquid filtration, the flow in the filter is usually laminar and incompressible. We use the Navier–Stokes–Brinkman system of equations [17, 21] to describe the coupled flow in 3D domain Ω, consisting of fluid and porous subdomains, i.e. $\Omega = \Omega_p \cup \Omega_f$:

$$\underbrace{\rho\frac{\partial \vec{u}}{\partial t} - \nabla\cdot(\tilde{\mu}\nabla\vec{u}) + (\rho\vec{u}\cdot\nabla)\vec{u}}_{\text{Navier–Stokes}} + \overbrace{\tilde{\mu}\tilde{\mathbf{K}}^{-1}\vec{u} + \underbrace{\nabla p = \vec{\tilde{f}}}}^{\text{Darcy}}, \tag{2.1}$$

$$\nabla\cdot\vec{u} = 0,$$

where the tilde quantities are defined using fictitious region method:

$$\tilde{\mu} = \begin{cases} \mu & \text{in } \Omega_f, \\ \mu_{eff} & \text{in } \Omega_p, \end{cases} \qquad \vec{\tilde{f}} = \begin{cases} \vec{f}_{NS} & \text{in } \Omega_f, \\ \vec{f}_D & \text{in } \Omega_p, \end{cases} \qquad \tilde{\mathbf{K}}^{-1} = \begin{cases} 0 & \text{in } \Omega_f, \\ \mathbf{K}^{-1} & \text{in } \Omega_p. \end{cases}$$

Here $\vec{u}$, p stand for velocity and pressure respectively, ρ, μ and $\mathbf{K}$ denote the density, viscosity, and the permeability tensor of the porous medium, respectively.

The Navier–Stokes equations (see [14]) describe the flow in pure fluid regions. The Brinkman equations [5] are used as an extension to the Darcy model for the flow in highly porous media (note, that porosity of the nonwoven filtering media, which is of our primary interest, is often more than 0.9). It was shown that the Navier–Stokes–Brinkman system can be used to describe the coupled flow without the explicit interface conditions [1].

2.2 Discrete approximation

The governing equations (2.1) are discretized by the Finite Volume Method (see [14]). A collocated arrangement of the unknowns $\vec{u}$ and p is used, i.e. the unknowns are assigned to the centres of control volumes. The Chorin method [7] for the Navier–Stokes equations, along with a proper modification for Navier–Stokes–Brinkman case [8], is used as a projection method decoupling momentum and continuity equations.

The following notations are introduced: the operators corresponding to discretized convective terms and diffusive terms in the momentum equations are denoted by $\mathrm{C}(\vec{u})\vec{u}$ and by $D\vec{u}$, respectively. G is the discretization of the gradient, and G^T is the discretization of the divergence operator. Finally, $\mathrm{B}\vec{u}$ denotes the operator corresponding to the Darcy term $\mu\tilde{\mathbf{K}}^{-1}\vec{u}$ in the momentum equations. Below, we use the superscript n to denote the values at the old time level, and $(n+1)$, or no superscript to denote the values at the new time level. Notation τ stands for the time step, $\tau = t^{n+1} - t^n$. Then the following fractional time step discretization scheme is defined [8, 22]. First, momentum equations are solved with respect to the velocities using the old value of the pressure gradient, thus obtaining a prediction for the velocity:

$$\begin{gathered}\big(\rho\vec{u}^{n+\frac{1}{2}} - \rho\vec{u}^{n}\big) + \tau\big(C(\vec{u}^n) - D + B\big)\vec{u}^{n+\frac{1}{2}} = \tau G\, p^n,\\ \big(\rho\vec{u^{n+1}} - \rho\vec{u}^{n+\frac{1}{2}}\big) + \tau\big(B\vec{u}^{n+1} - B\vec{u}^{n+\frac{1}{2}}\big) = \tau\big(Gp^{n+1} - Gp^n\big),\\ G^T\rho\vec{u^{n+1}} = 0.\end{gathered} \tag{2.2}$$

The pressure correction equation then is solved. It takes into account the specifics of the flow in the porous media (see a detailed discussion in [8, 9])

$$G^T\Big(I + \frac{\tau}{\rho}B\Big)^{-1}\tau Gq = -G^T\rho\vec{u}^{n+\frac{1}{2}}, \tag{2.3}$$

here $q = p^{n+1} - p^n$ is the pressure correction, I is the 3×3 identity matrix. At the last step, the pressure is updated, $p^{n+1} = p^n + q$, and the new velocity is calculated:

$$\rho\vec{u}^{n+1} = \rho\vec{u}^{n+\frac{1}{2}} + \Big(I + \frac{\tau}{\rho}B\Big)^{-1}\tau Gq. \tag{2.4}$$

2.3 Parallel algorithm

On the basis of this sequential numerical algorithm, the parallel algorithm is developed using data (domain) decomposition method [20]. We note that the load balancing problem should be solved during the implementation of this step. First, it is aimed to guarantee that each processor has about the same number of elements, since this number defines the computational complexity for all parts of the discrete algorithm (2.2)–(2.4).

Due to the stencil of discretization, the computational domains of different processors are overlapping. The information belonging to the overlapped regions should be exchanged among processors. This is done by so called ghost-cells approach. The data exchange between the processes is implemented using

the MPI library [15]. The time of data exchanges is contributing to the additional costs of the parallel algorithm. Thus, a second goal of defining the optimal data mapping is to minimize the overlapping regions. In our parallel direct solver, we are using the multilevel partitioning method from METIS software library [18] to partition the discrete mesh between the parallel processes.

Next, we present a summary on the complexity analysis of the parallel algorithm. The matrixes and right-hand side vectors are assembled element by element. This is done locally by each processor. All ghost values of the vectors belonging to overlapping regions are exchanged among processors. We can estimate the costs of data exchange operation in the worst case as $W_{exch} = \alpha_e + \beta_e m$, where m is the number of items sent between two processors, α is the message startup time and β is the time required to send one element of data. The time required to calculate all coefficients of the discrete problem is given by $W_{p,coeff} = c_1 n/p$, here n is the number of elements in the grid.

In the parallel algorithm, we use BiCGSTAB algorithm as a linear solver. A block version of the Gauss–Seidel preconditioner is implemented, when each processor computes B^{-1} by using only a local part of matrix $\mathbf{A}$. The complexity of all vector *saxpy* operations calculated during one iteration is $W_{p,saxpy} = c_2 n/p$. The complexity of two matrix-vector multiplications during one iteration is estimated by $W_{p,mv} = c_3 n/p + 2(\alpha_e + \beta_e m)$. Computation of all inner products and norms during one iteration require $W_{p,dot} = c_4 n/p + 5R(p)(\alpha_r + \beta_r)$ operations. The computation of the preconditioner B and application of this preconditioner is done locally by each processor without any communication operation, the complexity of this step is given by $W_{p,D} = (c_5 + c_6)n/p$. Summing up all the estimates, the theoretical model of the complexity of the parallel algorithm is obtained

$$W_p = K\Big((c_1 + c_5)\frac{n}{p} + c_7(\alpha_e + \beta_e m(p))\Big) + N\Big((c_2 + c_6 + c_{dot})\frac{n}{p} + c_8 R(p)(\alpha_r + \beta_r) + c_9(\alpha_e + \beta_e m(p))\Big), \quad (2.5)$$

where K is the number of steps in the outer loop of algorithm (2.2)–(2.4), and N is a total number of BiCGSTAB iterations. The experimental efficiency analysis of the proposed parallel direct solver was done in [8, 22]. The obtained computational results are in agreement with the theoretical scalability models.

Remark 1. The scalability analysis of the parallel algorithm (2.2)–(2.4) is done in the case when we have p computing processes and all of them are used to solve one fixed job in parallel as fast as possible. Optimization algorithms define a set of jobs, therefore a more complicated resource distribution problem should be solved: all processes are divided into groups and the main goal is to minimize the time required to solve all tasks in the set of jobs. For branch and bound and quasi-gradient optimization algorithms, the order among tasks is defined by a dependence graph, thus the resource distribution problem starts to be even more complex. Some theoretical and experimental results on solution of this problem will be presented in the following sections.

3 Parallel Optimization Solver

In this work, on the basis of existing direct solver we build a parallel solver for the optimization of oil filters. For this purpose, we have developed a general master-slave (MS) parallel template, which is specially designed for the easy integration of direct parallel solvers with complicated input and output parts of the solver. Master-slave parallelization paradigm is well known and widely used [20]. Master process reads the problem input, generates and distributes jobs to the slave processes. Slave processes receive jobs from the master, solve them and return back the obtained results. Finally, master process receives the results from the slaves and generates a new set of jobs if necessary.

Let us now formulate the special features of our master-slave template:

- Jobs are solved by the groups of slaves using a parallel direct problem solver. Master forms the groups of slaves at the beginning of solution process and directly communicates later only with the masters of the groups.
- Input parameters and results of the job are exchanged between master and according group of slaves using input and output files.
- Template is built as a hierarchy of C++ classes. Basic class implements the basic functionality of master-slave algorithm and specifies the pure virtual functions, which need to be implemented in descendant classes to obtain problem-specific solvers.

Basic class of our template is shown in Figure 1.

```
class BasicSolver{
  ...
  public:
    ...
    void  CreateGroupsOfSlaves();
    void  SolveProblem();
    ...
    virtual int  TakeNewJob(int& JobId) = 0;
    virtual int  SolveOneJob(int JobId) = 0;
    virtual int  AssimilateResults(int JobId) = 0;
}
```

Figure 1. Basic class of parallel master-slave template.

The main basic method `SolveProblem`() implements the general loop of master-slave work-flow using the pure virtual functions. Specifically, `TakeNewJob`(int& JobId) is called by the master process to create a new job, i.e. if the pool of jobs is not empty, master creates a directory with the according input files for direct problem solver. `SolveOneJob`(int JobId) is called by the group of slaves, which has received a job from master, to solve it by using direct solver and to save the results in output files. These results are further processed and accumulated by master in `AssimilateResults`(int JobId).

The usage of technology based on input and output files is not so efficient as a direct message passing between processes, however, the performance overhead is negligible for coarse grained jobs (typical for industrial applications). In turn, such an approach significantly simplifies the integration of direct problem solvers with complicated input and output data structures into our template.

Such a template type implementation of the MS tool enables users to integrate into the parallel MS optimization solver two specific problem oriented components: a parallel solver of the direct problem (black-box approach) and different optimization algorithms for searching the optimal solution of a given technological or industrial problem. Optimization algorithms can be integrated into this tool in a robust and efficient way through specific versions of the `TakeNewJob` method. Then a pool of jobs is updated dynamically according to the logic of optimization algorithms (e.g., genetic or the Nelder–Mead simplex method).

Let us now consider a problem of scalability of parallel optimization solver. We return to the question, how to divide processes into the groups to minimize the optimization problem solution time? Clearly, the answer depends on many factors including parallel hardware architecture and granularity of the jobs. Obviously, the groups should be big enough for a single job to fit into the memory available to one group.

Let us now consider a more interesting case when the jobs are small enough for several jobs to fit at the same time into a single node with multicore processor, which is currently a predominant parallel architecture. We have performed scalability tests on two types of nodes with four cores processors each. First node has Intel®Core™ processor i7-860 @ 2.80 GHz and 4 GB DDR3-1600 RAM. Second node has Intel®Core™2 Quad processor Q6600 @ 2.4 GHz and 4 GB DDR2-800 RAM. We have defined 3 test problems of different sizes. "Small" test solves direct problems with 45120 control volumes, "medium" test with 360960 volumes, "big" test with 1443840 volumes. Solving direct problems the number of BiCGSTAB iterations was fixed to 100, 200 and 400 accordingly, and the number of time steps fixed to 30. This was done to ensure that the same amount of computations is performed with different number of processes in spite of the differences in the convergence rate due to parallel preconditioner and roundoff errors.

In Table 1, we present the performance results of our parallel solver obtained on one i7 node. We show the solution time T_p and speedup S_p solving 15 small, 15 medium and 9 big problems with different number of groups. One of 4 available processes is reserved for the master. So, for one node we can define 1 group of 3 slaves, 2 groups of 1 and 2 slaves, or 3 groups made from 1 slave.

Analyzing obtained results, we see that for all configurations performance of the solver is degrading with increasing problems size. This is a known issue of memory bus saturation in the multicore systems. A more interesting is the fact, that the use of larger groups to solve direct problems in parallel is more efficient only for relatively small problems. For bigger problems it is better to use groups made from one slave and to solve several problems at the same time. Such a scenario is using 3 times more memory than 1 group of 3 slaves, however, it is obviously better utilizing the caching systems of multicore processor. 3 groups

Table 1. Performance results of parallel optimization solver: time T_p and speedup S_p for 1 node with i7 processor (4 cores).

		small(15)	medium(15)	big(9)
	T_1	477.9	6970	27950
1 group	T_4	201.8	3778	17036
	S_4	2.37	1.85	1.64
2 groups	T_4	213.8	3731	16539
	S_4	2.24	1.87	1.69
3 groups	T_4	220.1	3594	15880
	S_4	2.17	1.94	1.76

can solve their jobs independently, i.e. asynchronously, when 3 slaves in one group need to communicate (send and receive data on ghost values of vectors belonging to overlapping regions) and synchronize (global reduce operations) with each other during computations (see the complexity analysis (2.5) of the parallel algorithm).

In Table 2, we show the performance results obtained on our second testing node with the Quad processor. We see qualitatively the same picture, although, the actual numbers are somewhat different due to the differences in hardware architecture and characteristics.

Table 2. Performance results of parallel optimization solver: time T_p and speedup S_p for 1 node with Quad processor (4 cores).

		small(15)	medium(15)	big(9)
	T_1	791.1	11805	50604
1 group	T_4	305.5	7727	36386
	S_4	2.59	1.53	1.39
2 groups	T_4	356.2	7739	35060
	S_4	2.22	1.53	1.44
3 groups	T_4	386.8	7290	32930
	S_4	2.06	1.62	1.54

In Table 3, we present the performance results obtained on two i7 nodes. Now we can define 1 group of 7 slaves, 2 groups of 3 and 4 slaves, 4 groups made from 1, 2, 2, 2 slaves accordingly, and 7 groups made from 1 slave. We have selected the number of jobs (14) to ensure the workload balance between the groups of different sizes. For example, it was assumed that in configuration with 2 groups the first group of 3 slaves will solve 6 jobs, when the second group of 4 slaves will solve 8 jobs. However, ideal 100% workload balance is practically achieved only for the configuration with 1 group of 7 slaves. Despite that configuration with one group is the least efficient among all configurations, due

to relatively slow inter-node communication (Gigabit Ethernet). The results of all other tests are more or less affected by the imbalance in workload, when some groups finished their work earlier then the others, became idle and do not contributed to the overall speedup.

Table 3. Performance results of parallel optimization solver: time T_p and speedup S_p for 2 nodes with i7 processors (2x4 cores).

		small(14)	medium(14)	big(14)
	T_1	448.7	6251	43524
1 group	T_8	120.2	1856	14542
	S_8	3.73	3.37	2.99
2 groups	T_8	84.9	1768	13258
	S_8	5.29	3.54	3.28
4 groups	T_8	99.0	1869	14085
	S_8	4.53	3.35	3.09
7 groups	T_8	106.1	1822	13476
	S_8	4.23	3.43	3.23

To understand the obtained results, one needs to construct the Gantt timing charts [20] and to determine the times, required to solve a single problem by the groups of different sizes. For the case of 7 groups consisting of one slave (i.e. sequential jobs), we have to distinguish two different nodes: 3 groups solving their jobs on the master node at the same time and 4 groups on the other node. Let us denote by $T_{p,1}$ the time used to solve one sequential job when p such jobs are solved on one node at the same time.

Table 4. Performance results solving 1 job on one node with i7 processor.

Problem	T_1	$T_{3,1}$	$S_{3,1}$	$T_{4,1}$	$S_{4,1}$	$T_{3,3}$	$S_{3,3}$	$T_{4,4}$	$S_{4,4}$
small	31.86	44.0	2.17	53.1	2.40	13.45	2.37	10.5	3.03
medium	464.67	718.8	1.94	911.1	2.04	251.9	1.85	234.4	1.98
big	3106	5293	1.76	6738	1.84	1893	1.64	1785	1.74

In Table 4, we show the $T_{3,1}$ times, which can be obtained from Table 1 dividing the respective CPU times by 5, 5 and 3. Then after construction of Gantt charts for the results with 7 groups in Table 3, we see that 3 slaves on the master node solve 2 jobs each in 88.0, 1437.6 and 10586 [s]. For the remaining time they are idle, waiting until the 4 groups on the second node will finish their second job. So, we can obtain the $T_{4,1}$ times, which are also shown in Table 4. These times were confirmed by the separate tests, what shows that the master-slave overhead (times for sending, receiving and waiting for the tasks) is negligible for our problem sizes.

To compare the $T_{3,1}$ and $T_{4,1}$ results, we use the speedup coefficient $S_{p,1}$,

which in notations of Table 4 is defined as

$$S_{p,1} = pT_1/T_{p,1},$$

because actually p problems are solved at the same time (in parallel) in these cases. Analyzing obtained results, we see that for all three test problems $S_{4,1} > S_{3,1}$, thus it is recommended to use all cores in computations if the limit of the memory of one node is not reached. However, due to the memory bus saturation, obtained performance gain and efficiency of the core usage is decreasing as the problem size increases.

To analyze in a similar way the results for 2 groups in Table 3, we need the time $T_{3,3}$ to solve one job in the group of 3 slaves on the master node and the time $T_{4,4}$ to solve one job in the group of 4 slaves on the other node with 4 available cores. The $T_{3,3}$ times can again be obtained from the Table 1 dividing the respective times by 15, 15 and 9 (see Table 4). Constructing Gantt charts, we see that for the small problem the group of 3 slaves solves 6 problems and finishes first in 80.7 [s]. Hence the total time - 84.9 [s] is the time, when the group of 4 slaves ends the solution of all 8 problems. For medium and big problems we see that the group of 3 slaves solves 7 problems and finishes last. Using the separate $T_{4,4}$ timings presented in Table 4, we get the finishing times of the group of 4: 1641 (medium) and 12495 (big) seconds. So, the idle times for the configuration with 2 groups are significantly smaller than for the configuration with 7 groups.

Similar analysis can be done for the other configurations of groups. Clearly, for the relatively small number of problems the influence of work balance on parallel performance is very significant (it is sufficient to consider the case of 8 problems and 7 groups). However, what kind of performance one can expect to obtain for big numbers of problems, when the influence of the work balance is small? Next, we propose a theoretical model for the computation of asymptotic (ideal) speedup. Let us consider the case with 2 groups of 3 and 4 processes. Let $n = n_1 + n_2$ be the total number of solved jobs, where n_1 is the number of jobs solved on the master node and n_2 on the other node. The ideal work balance is achieved, when $n_1 T_{3,3} = n_2 T_{4,4}$. Under this condition, we can obtain the following expression for the asymptotic speedup coefficient:

$$S_{8,2gr} = \frac{nT_1}{n_2 T_{4,4}} = \frac{(n_2 T_{4,4}/T_{3,3} + n_2)T_1}{n_2 T_{4,4}} = \frac{(T_{4,4}/T_{3,3} + 1)T_1}{T_{4,4}}. \tag{3.1}$$

Similarly, we can get the asymptotic speedup coefficient for the configuration with 7 groups:

$$S_{8,7gr} = \frac{nT_1}{n_2 T_{4,1}} = \frac{(3n_2 T_{4,1}/T_{3,1} + 4n_2)T_1}{n_2 T_{4,1}} = \frac{(3T_{4,1}/T_{3,1} + 4)T_1}{T_{4,1}}. \tag{3.2}$$

Note that these formulas can be extended for the bigger number of nodes. For example, for m nodes with the m groups, we get:

$$S_{4m,m\,gr} = \frac{(T_{4,4}/T_{3,3} + (m-1))T_1}{T_{4,4}}.$$

Using the performance benchmarks from Table 4, we can compute the asymptotic speedup coefficients (3.1) and (3.2) for two i7 nodes. The results are presented in Table 5. Comparing now configurations with 2 and 7 groups, we see a similar tendency as for 1 node in Table 1. Bigger groups are better only for the small problem. Increasing the problem size, the groups made from one slave become more efficient. However, in practice it could be a good compromise (or at least the starting configuration) to take the groups made from all cores on the node to avoid the possible work balance problems.

Table 5. Asymptotic (ideal) speedup coefficients using 2 nodes with i7 processors.

	small	medium	big
$S_{8,2gr}$	5.40	3.83	3.38
$S_{8,7gr}$	4.57	3.98	3.60

4 Optimization Problem

The example of optimization problem is chosen to serve the needs of the design of new generation of filters [13] as shown in Figure 2. It should be noted that oil filters are often equipped with a so called bypass option. The bypass prevents the breaks in the porous media due to the high pressure at cold regimes - in this case the oil is redirected to a pipe which bypasses the filtering medium. Obviously, the bypassed oil is not filtered in this situation.

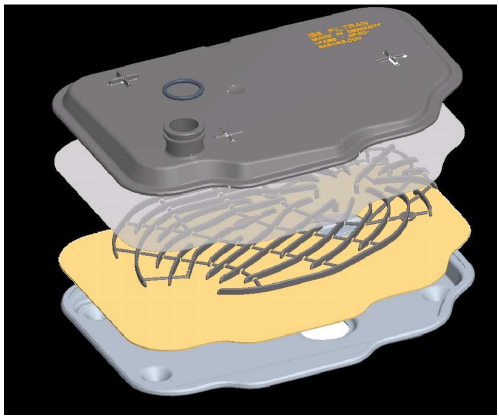

Figure 2. A real industrial filter with the multiple porous media layers.

Alternatively, the new league of filters are designed without the bypass option, but with an additional fine perforated filter layer that allows the oil to flow through the holes at cold temperatures. Additionally, a coarse filter layer in the form of a solid mesh is added to filter out the large particles. With the support of the numerical simulations, the size of the holes and the distance

between the two porous layers should be designed such that most of the oil flows through the porous part of the first layer at high temperature, and not through the holes.

It is common understanding that the porous layers are the primary cause of the pressure drop across the filter. This is true only when the filter element is designed in a way that there is enough space between the inlet(bottom) and the porous layer, and between the porous layer and the outlet(top). Therefore, we study this filter in a simplified geometry, namely a simple channel geometry as shown in Figure 3.

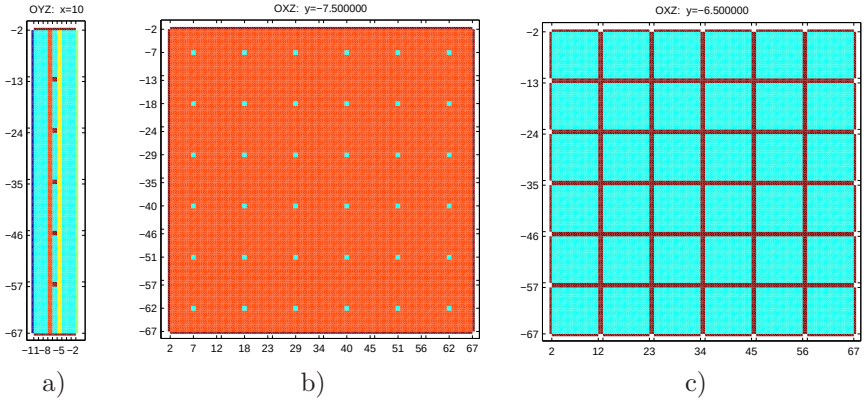

Figure 3. A simplified channel filter with the multiple porous layers. Cross sectional views of the complicated structure of the multiple porous layers.

The considered filter element is a parallelepiped with two filtering porous layers and a supporting solid mesh between them, as shown in Figure 3 (a). Additionally, the first porous layer ($-8 \leqslant y \leqslant -7$) has a set of holes, as shown in Figure 3 (b).

The considered geometry has up to 9 parameters, which can be modified: the distance from the inlet to the first porous layer (Lin), the distance from the second porous layer to the outlet ($Lout$), the thickness of the first and second porous layers ($hP1$ and $hP2$), the thickness of the solid mesh in X direction (hMX) and Y–Z directions ($hMYZ$), the size of the hole (h), the size of the open cell in the solid mesh ($hCell$), the number of holes (and cells). Obviously, the number of parameters and the size of the problems need to be kept to minimum.

Reduction of the size of discrete problems solved in one job can be done by using different methods. A general and robust technique is to approximate PDEs on coarse meshes which are still sufficient to get a correct trend in the values of the objective function. At the second stage of the optimization process, the accuracy of the predicted optimal solution on a coarse mesh can be corrected on the fine mesh. Such multiscale strategy can reduce essentially computational costs of the full optimization cycle. It was used in [10] to optimize electrical cables.

Table 6. Results of numerical experiments for the filter element with two porous layers. h is the size of the hole, $hP2$ is the thickness of the second porous layer, hMX is the thickness of the solid mesh in X direction (i.e. the distance between porous layers), dP is the average pressure drop between the inlet and outlet, Fr is the flow rate ratio through the hole of the first porous layer.

h [mm]	$hP2$ [mm]	hMX [mm]	dP [mbar]	Fr [%]
0.5	0.5	0.5	982.394	33.1547
0.5	0.5	1	970.497	33.9901
0.5	0.5	1.5	969.362	34.0744
0.5	1	0.5	1021.26	33.0996
0.5	1	1	1008.49	33.9869
0.5	1	1.5	1007.29	34.0740
0.5	1.5	0.5	1058.43	33.0956
0.5	1.5	1	1045.53	33.9794
0.5	1.5	1.5	1044.10	34.0894
0.75	0.5	0.5	560.531	62.9467
0.75	0.5	1	520.700	65.7343
0.75	0.5	1.5	516.815	66.0124
0.75	1	0.5	601.242	62.7783
0.75	1	1	558.838	65.7235
0.75	1	1.5	554.785	66.0104
0.75	1.5	0.5	639.290	62.7173
0.75	1.5	1	595.889	65.7204
0.75	1.5	1.5	591.781	66.0095
1	0.5	0.5	334.173	79.0471
1	0.5	1	275.933	83.1414
1	0.5	1.5	268.899	83.6375
1	1	0.5	376.246	78.7858
1	1	1	314.156	83.1257
1	1	1.5	306.884	83.6345
1	1.5	0.5	414.743	78.6966
1	1.5	1	351.238	83.1242
1	1.5	1.5	343.872	83.6373
1.25	0.5	0.5	229.606	86.5193
1.25	0.5	1	167.113	90.9288
1.25	0.5	1.5	158.500	91.5381
1.25	1	0.5	272.251	86.2217
1.25	1	1	205.373	90.9128
1.25	1	1.5	196.488	91.5345
1.25	1.5	0.5	310.911	86.1232
1.25	1.5	1	242.487	90.9060
1.25	1.5	1.5	233.493	91.5330
1.5	0.5	0.5	178.111	90.2249
1.5	0.5	1	117.296	94.5145
1.5	0.5	1.5	108.474	95.1376
1.5	1	0.5	220.909	89.9182
1.5	1	1	155.568	94.4952
1.5	1	1.5	146.461	95.1336
1.5	1.5	0.5	259.570	89.8204
1.5	1.5	1	192.674	94.4905
1.5	1.5	1.5	183.454	95.1345

The second approach of reduction of model order is based on Proper Orthogonal Decomposition (POD). This method can also be applied in a systematic manner and it was used for studies of lithium-ion battery simulations [6], turbulent flow [12], image processing. The main idea of of POD is to find a special basis for a modal decomposition of the snapshots (solutions computed on the fine mesh) and to project the residual of governing equations to the subspace generated by a small number of selected most energetic modes. A standard eigenproblem yields eigenvalues and eigenvectors (POD modes), that form a complete, orthonormal set.

Solving the direct problems, we compute two target values: average pressure drop between the inlet and outlet (dP) and flow rate ratio through the holes of the first porous layer (Fr). The governing equations (2.1) were solved with the following parameters: inflow velocity $U_{in} = 25.3812$ mm/s (Reynolds number $Re = 1$), $\rho = 8.526 \cdot 10^{-7}$ kg/mm^3, viscosity $\mu = 2.164 \cdot 10^{-4}$ kg/(mm s), isotropic permeability of the first porous layer, $K = 4.2 \cdot 10^{-5}$ mm^2, and isotropic permeability of the second porous layer, $K = 7.5 \cdot 10^{-4}$ mm^2.

Numerical experiments showed that our target quantities (dP and Fr) are independent on the number of holes and cells in the solid mesh, when symmetry boundary conditions are applied on the side walls. Therefore, further we are considering a filtering element consisting only of one single hole and cell. Obviously, this reduces the size of our problem 36 times comparing to the one shown in Figure 3. The problem was further reduced by changing Lin to 2 and $Lout$ to 1 mm without noticeable difference to our target quantities.

In Table 6, we show the results of numerical experiments for the described geometry of the filtering element. Target quantities were computed for the changing size of the hole h, the thickness of the solid mesh in X direction hMX (i.e. the distance between porous layers) and the thickness of the second porous layer $hP2$. The other geometry parameters were fixed at $hCell = 10$, $hP1 = 1$, $hMYZ = 1$ mm. Computations were performed on VGTU PC cluster VILKAS. Solution of the single problem with 0.0625 mm grid takes around one hour for a group of 16 processes.

Obtained results qualitatively are not surprising. Obviously, when the hole size is decreasing, the flow rate ratio through the hole(s) is decreasing and the pressure drop is increasing and vice versa. The advantage of numerical simulation is the fact that it allows to obtain quantitative dependence, which can be of great value for the engineers (similar to Pareto front in multi-criteria optimization).

5 Conclusions

In this work, we have presented a parallel solver for the optimization of oil filters. We have combined two parallelization paradigms, namely domain decomposition for the direct problem solver and master-slave for optimization. Such an approach gives a noticeable performance gain on modern computer clusters with multicore and SMP processors. It should be noticed that chosen structure of the solver is very suitable for the grids built from the computer clusters.

We have also developed a general master-slave parallel template. It was specially designed for easy integration of direct parallel solvers with complicated input and output data structures. Then integration step does not require from the template user any additional parallel MPI programing.

References

[1] Ph. Angot. Analysis of singular perturbations on the Brinkman problem for fictitious domain models of viscous flows. *Math. Meth. Appl. Sci.*, **22**(16):1395–1412, 1999. Doi:10.1002/(SICI)1099-1476(19991110)22:16¡1395::AID-MMA84¿3.0.CO;2-3.

[2] M. Baravykaitė, R. Belevičius and R. Čiegis. One application of the parallelization tool of Master – Slave algorithms. *Informatica*, **13**(4):393–404, 2002.

[3] M. Baravykaitė and R. Čiegis. An implementation of a parallel generalized branch and bound template. *Math. Model. Anal.*, **12**(3):277–289, 2007. Doi:10.3846/1392-6292.2007.12.277-289.

[4] M. Baravykaitė, R. Čiegis and J. Žilinskas. Template realization of generalized branch and bound algorithm. *Math. Model. Anal.*, **10**(3):217–236, 2005. Doi:10.1080/13926292.2005.9637283.

[5] H.C. Brinkman. A calculation of the viscous force exerted by a flowing fluid on a dense swarm of particles. *Appl. Sci. Res.*, **1**(1):27–34, 1947.

[6] L. Cai and R. White. Reduction of model order based on proper orthogonal decomposition for lithium-ion battery simulations. *J. Electrochemical Soc.*, **156**(3):A154–A161, 2009. Doi:10.1149/1.3049347.

[7] A.J. Chorin. Numerical solution of Navier–Stokes equation. *Math. Comp.*, **22**:745–760, 1968. Doi:10.1090/S0025-5718-1968-0242392-2.

[8] R. Čiegis, O. Iliev and Z. Lakdawala. On parallel numerical algorithms for simulating industrial filtration problems. *Comput. Methods Appl. Math.*, **7**(2):118–134, 2007.

[9] R. Čiegis, O. Iliev, V. Starikovičius and K. Steiner. Numerical algorithms for solving problems of multiphase flows in porous media. *Math. Model. Anal.*, **11**(2):133–148, 2006. Doi:10.1080/13926292.2006.9637308.

[10] Raim. Čiegis, Rem. Čiegis, M. Meilūnas, G. Jankevičiūtė and V. Starikovičius. Parallel mumerical algorithm for optimization of electrical cables. *Math. Model. Anal.*, **13**(4):471–482, 2008. Doi:10.3846/1392-6292.2008.13.471-482.

[11] C.A.C. Coello. An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys*, **32**(2):109–143, 2000. Doi:10.1145/358923.358929.

[12] M. Couplet, C. Basdevant and C. Sagaut. Calibrated reduced-order POD-Galerkin system for fluid flow modelling. *J. Comp. Phys.*, **207**(1):192–220, 2005. Doi:10.1016/j.jcp.2005.01.008.

[13] M. Dedering, O. Iliev, Z. Lakdawala, R. Čiegis, V. Starikovičius and P. Popov. Advanced CFD simulation of filtration processes. In *Proceedings of International Conference and Exihibition for Filtration and Separation Technology (FILTECH 2009)*, volume 1, pp. 440–444, Wiesbaden, 2009.

[14] C.A.J. Fletcher. *Computational Techniques for Fluid Dynamics*. Springer, 1991.

[15] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. www.mpi-forum.org, Version 1.1, 1995.

[16] R. Horst, P.M. Pardalos and Nguyen V. Thoai. *Introduction to Global Optimization*. Kluwer Academic Publishers, 2000.

[17] O. Iliev and V. Laptev. On numerical simulation of flow through oil filters. *Comput. Vis. Sci.*, **6**:139–146, 2004.

[18] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, **20**(1):359–392, 1999.

[19] P. Korošec and J. Šilc. Real-parameter optimization using stigmergy. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and Their Application, BIOMA 2006*, pp. 73–84, 2006.

[20] V. Kumar, A. Grama, A. Gupta and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, 1994.

[21] V. Laptev. *Numerical Solution of Coupled Flow in Plain and Porous Media*. PhD thesis, Technical University of Kaiserslautern, 2003.

[22] V. Starikovičius, R. Čiegis, O. Iliev and Z. Lakdawala. A parallel solver for the 3D simulation of flows through oil filters. In R. Čiegis, D. Henty, B. Kagstrōm and J. Žilinskas(Eds.), *Parallel Linear Algebra and Optimization: Advances and Applications. Springer Optimization and Its Applications. ISBN: 978-0-387-09706-0*, volume 27, pp. 185–196, New York, 2009. Springer.

[23] P.J. van Laarhoven and E.H. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publisher, 1992.