DE GRUYTER
OPEN

Utilization of parallel computing in chemical engineering

Matej Danko, Juraj Labovský, Ján Janošovský, Zuzana Labovská, Ľudovít Jelemenský

Institute of Chemical and Environmental Engineering,
Slovak University of Technology, Bratislava
ludovit.jelemensky@stuba.sk

Abstract: The main objective of the presented work was to explore the possibilities of parallel computing utilization in chemical engineering. Parallel computers and principles of parallel computing are in brief described in Introduction. The next part exposes the possibilities of parallel programming in Matlab and C# programming language environment. The next three parts provide case studies of parallel computing in chemical engineering. Each example of the benefits of HPC involves a comparison with its serial equivalents.

Keywords: parallel computing; parallel programming; chemical engineering

Introduction

The need for powerful computers has led to the construction of massive parallel super-computers enabling to understand phenomena such as galaxy formation, molecular dynamics and climate change, among others (Navarro et al., 2014). Data parallelism is a form of computing parallelization across multiple processors in parallel computing environments, focused on data distribution across different parallel computing nodes. Not long ago, parallel computers were associated with fairly large machines, most of them running Linux, designed for special computations requiring extreme performance, such as quantum mechanical calculations, computational fluid dynamics, molecular simulations, and chemical process optimization, among other applications relevant to chemical industry (Castier et al., 2014).
The first case study presented refers on a global optimization method exploiting parallel computers, used to fit the equation of state (EOS) parameters. Sometimes, model developers try to correlate experimental data as precisely as possible, which means that the global minimum of the objective function is the desirable target. Locating the global minimum of complicated functions, with many state parameters or a massive number of experimental data points, requires large computational effort and time consuming calculations, often carried out using a single processor in a personal computer (PC). Recent Windows versions of software such as Mathematica and Matlab exploit data parallelism to speed up the calculations. Nonetheless, many legacy sequential codes exist in languages such as Fortran and C#. Their adaptation for maximum performance in parallel computers may require extensive reprogramming but substantial performance gains are possible in certain applications with only a few changes in the existing codes (Castier et al., 2014). This case study shows that EOS parameter fitting is one of such applications comparing two types of software utilizing the Simplex algorithm in parallel computers, applicable both in single multiprocessor PCs and in cluster supercomputers.
The second presented case study shows the benefits of parallelization of a large system of nonlinear equations. The main goal was to design a technique allowing the utilization of existing codes designed for serial execution. As an example of this approach, a model of a distillation column for ammonia purification is presented.
The third case study refers to heavily stochastic modeling of virus membrane filtration, which is characterized by very low need for process communication. This fact represents the highest benefit of parallelization of the sequential code.

Case study — Global optimization

The problem of EOS parameters fitting is frequently used and well-known in the chemical engineering praxis. Probably the most challenging task is related to multiple minima in objective functions, each represented by a different parameter set. The most straightforward procedure is based on several runs of local optimization algorithms from different initial estimates or using global optimization methods. An important fact is that these local minima have very often similar values of the objective function but significantly different values of the EOS parameters. Another challenge of this approach is the selection of suitable starting points for the optimization and subsequent run of an efficient optimization process.

Optimization of the parameters of a non-random two-liquid model (short NRTL equation) for systems ethanol-benzene and diethylamine-ethanol was chosen as the case study. Ethanol-benzene equilibrium was described by 12 and diethylamine-ethanol equilibrium by 18 experimental isobaric points. The objective function used in the presented case studies was in the form:

$$f = \sum_{k=1}^{n_{exp}} \left(\left(y_{exp} - y_{calc} \right)^2 \right) + \sum_{k=1}^{n_{exp}} \left(\frac{\left(P_{exp} - P_{calc} \right)^2}{P_{exp}^{\ 2}} \right) \tag{1}$$

Here, the second equation term describes the impact of the temperature difference on the total pressure. Comparison of the experimental and optimized data is depicted in Fig. 1.

The presented case studies were run on an 80 cores computer cluster. To highlight the benefits of parallelization, two different languages (platforms) were used. The first chosen language was Matlab including Parallel Computing Toolbox™. This programming language has been developed by MathWorks and it is currently very popular in engineering and scientific disciplines as well as a teaching tool in many courses. Parallel Computing Toolbox™ allows solving computationally and data-intensive problems using multicore processors, GPUs, and computer clusters. High-level constructs, parallel for-loops, allows parallelizing applications without CUDA or MPI programming. The presented strategy is based on finding a suitable loop that allows the code (objective function calculation) to be executed in parallel. Initial estimates of the NRTL equation parameters from user defined interval were randomly generated. Endurance testing was based on the evaluation of 10 mil. of initial estimates and every test was repeated five times. The resulting time of endurance testing was defined as the average time of all five tests and differences in timings were less than 5 %. In each testing of global optimization, global minimum of the objective function was reached. Basic quantitative parameter, which compares the efficiency of the parallel algorithms, is the speedup (S_p) defined as the ratio between the base serial program's execution time (t_{seq}) and its parallel implementation execution time (t_{par}).

$$S_p = \frac{t_{seq}}{t_{par}} \tag{2}$$

Dependence of the execution time on the number of cluster cores and the speedup of the parallel code are depicted in the next series of figures (Fig. 2).

The second chosen platform was the .NET Framework and the programming language C#. To parallelize the algorithm, the MPI.NET library was used. MPI.NET is a high-performance, easy-to-use implementation of the Message Passing Interface (MPI) for Microsoft's .NET environment. MPI is nowadays a standard for writing parallel programs running on a distributed memory system, such as a compute cluster. MPI provides functions and subroutines to control parallel computations and pass information from one process to another. In a message-passing system, different concurrently-executing processes communicate by sending messages over a network. Unlike multi-threading, where different threads share the same program state, each of the MPI processes has its own, local, program state that cannot be observed or modified by any other process except in response to a message. Therefore, the MPI processes themselves can be as distributed as the network permits, with different processes running on different machines or even different architectures. Speedup of the parallel version is depicted in Fig. 3.

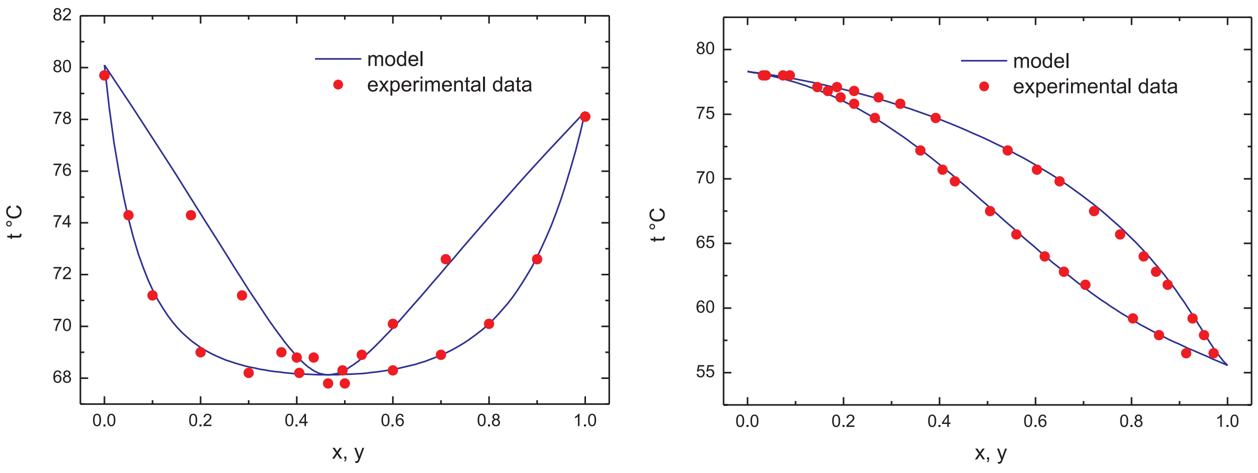

Fig. 1. t-x, y diagrams for binary systems used in the case studies: ethanol-benzen (left), diethylamine-ethanol (right).

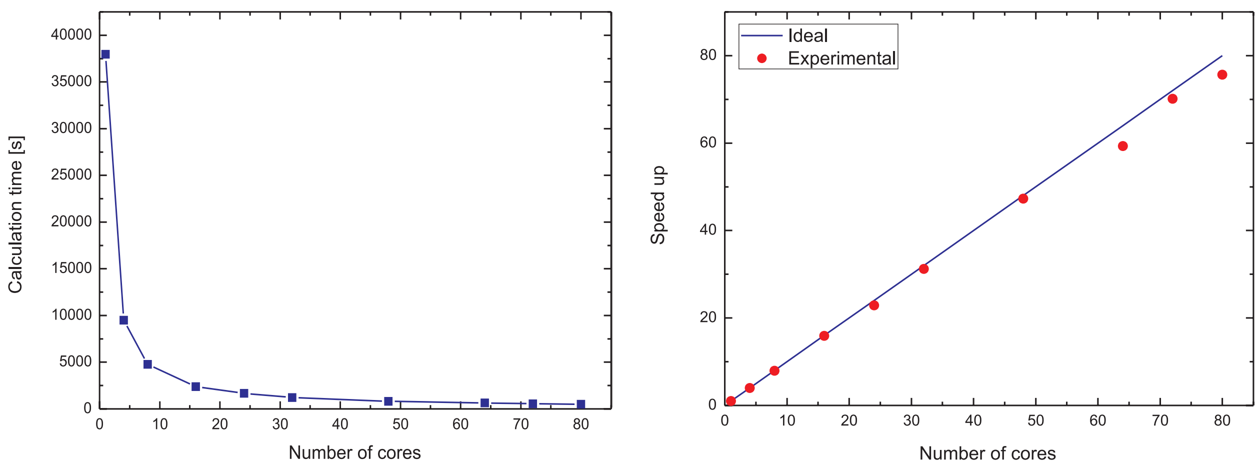

Fig. 2. Execution time and speedup as a function of the number of cluster cores.

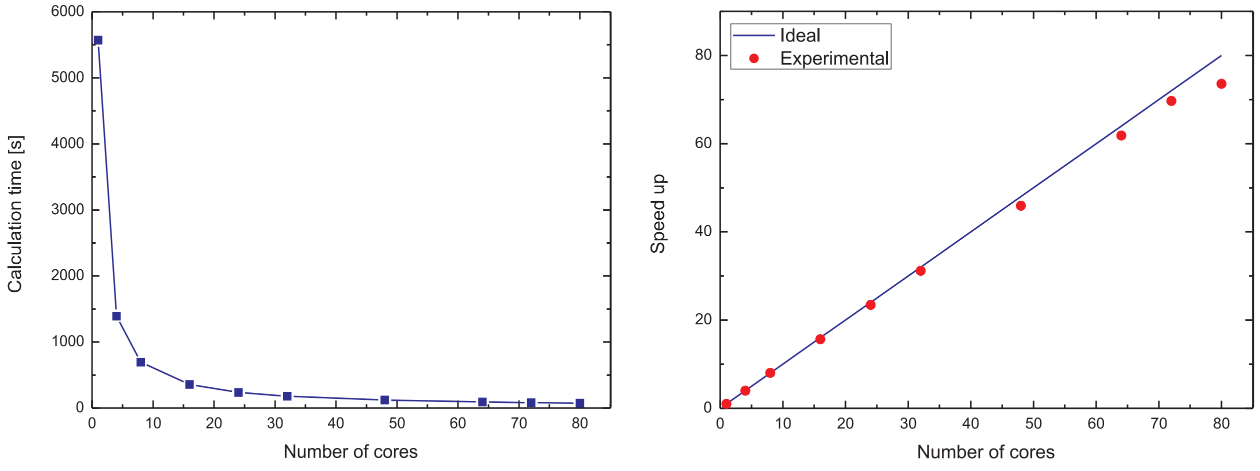

Fig. 3. Execution time and speedup as a function of the number of cluster cores.

From Figs. 2 and 3 it is clear that the speedup of parallel optimization is quite linear and the capacity utilization of computer sources is higher than 90 % in both cases. There is a huge difference in the timing of Matlab and MPI.NET implementations caused by fact that Matlab is a scripting programming language while a source code written in C# is compiled into an intermediate language (IL) that conforms to the Common Language Infrastructure (CLI) specification and to the native code when run.

Case study — Large system of nonlinear equations

Parallelization of solving a large system of nonlinear equations is the second presented case study. This problem occurs frequently in numerical methods in chemical engineering. To demonstrate the benefits of parallelization, the standard algorithm for solving a system of nonlinear equations implemented in Matlab was modified.

The Newton's method is a basic general purpose approach for solving nonlinear equations providing linear approximation to the nonlinear system based on a Jacobian matrix. In the general form of the Newton method, the iterations and phases of each step have to be performed one after another in order to preserve the correctness of the numerical algorithm. In general, the standard Newton's method comprises three basic steps:

1. estimation of the Jacobian — determined by backward differential formulas,
2. solution of a system of linear equations (very often large and sparse),
3. improvement of the previous estimate.

If the Jacobian is determined in each iteration, it is always the most time consuming part of the algorithm. On the other hand, parallelization of the estimation of the Jacobian is quite a trivial task as the columns of the Jacobian matrix can be processed independently.

The most universal solver for a system of nonlinear equations presented in Matlab is the function *fsolve* based on the interior-reflective Newton method and employing the subspace trust-region method. The default version of the algorithm uses sequential approach for the determination of the Jacobian. However, it allows overriding the estimation of the Jacobian via solver option settings. As our primary goal was to implement parallelization of the default version the solver (without significant modifications of the existing code), the sequential version of the Jacobian determination was replaced with a parallel version which is the most straightforward procedure. This approach allows utilizing the benefits of parallelization without modifying the existing users programs and default solver.

To demonstrate the simplicity and efficiency of the presented approach, a case study focused on the steady state simulation of a distillation column for ammonia purification was prepared. The mathematical model was described by MESH equations; MESH being an acronym referring to the different types of equation: Material balances, vapour–liquid Equilibrium equations, mole fraction Summations and enthalpy (H) balances. The main idea is in the assumption that the vapor and liquid streams leaving an equilibrium stage are in complete equilibrium with each other and thermodynamic relations can be used to determine the equilibrium stage temperature and relate the concentrations in the equilibrium streams at a given pressure (Perry et al., 1997). A complete distillation column is considered as a sequence of such stages. A distillation column model is an ideal candidate for the presented case study, as it allows increasing the number of trays, which results in the increase of the number of nonlinear equation to be solved. In general, the number of all column equations is $N_J\ (2N_I + 4) + 1$, where N_J is the number of column trays and N_I is the number of components of the separated mixture. Total material balance and material balances of components on trays are linearly dependent, which means that, the total number of nonlinear equations of the distillation column is $N_J\ (2N_I + 3) + 1$ which is at the same time also the number of unknown parameters. Table 1. presents the execution times for four different numbers of trays; all execution times are in seconds.

For better visualization of the results, a speedup of the parallel code is depicted in Fig. 4. From the pictures it is clear that the parallel version of the algorithm is more effective than its serial equivalent. On the other hand, in case of a 50 tray column (200 nonlinear equations), the benefits of paralleli-

Tab. 1. Execution time as a function of the number of equations and cores.

Number	Number of cores									
of trays	**1**	**2**	**4**	**8**	**16**	**24**	**48**	**64**	**72**	**80**
50	13.9	8.3	6.2	4.3	3.2	3.1	3.1	3.5	3.8	3.5
250	219.2	111.6	61.0	34.5	19.7	15.7	10.5	9.9	9.5	9.6
1000	1590.0	798.8	428.3	221.4	119.8	83.8	52.4	41.2	38.7	35.5
2500	15329.6	7697.3	4097.5	2099.8	1065.8	750.7	401.7	324.0	292.3	269.2

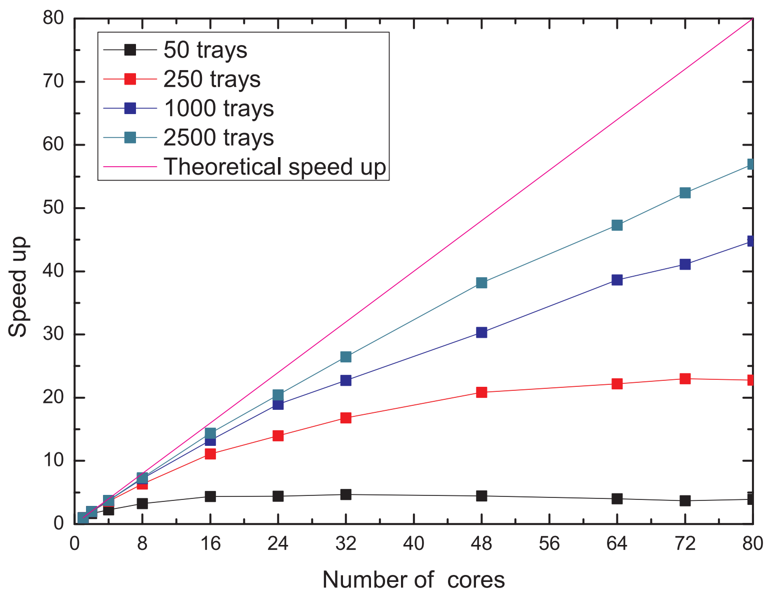

Fig. 4. Speedup of parallel code.

zation are evident when sixteen cores are employed. Further increase in the core number did not provide a significant increase in the speedup, probably due to the dominance of process communication over the time consuming single calculations. If the number of equations is higher, the capacity utilization increases radically. In case of the largest system investigated in the presented study, the measured speedup was higher than 55.

Case study – Stochastic hydrodynamic model of virus filtration

Solving problems of previous case studies required the use of numerical methods. The third case study on parallel computing in chemical engineering refers to hydrodynamic model of virus membrane filtration.

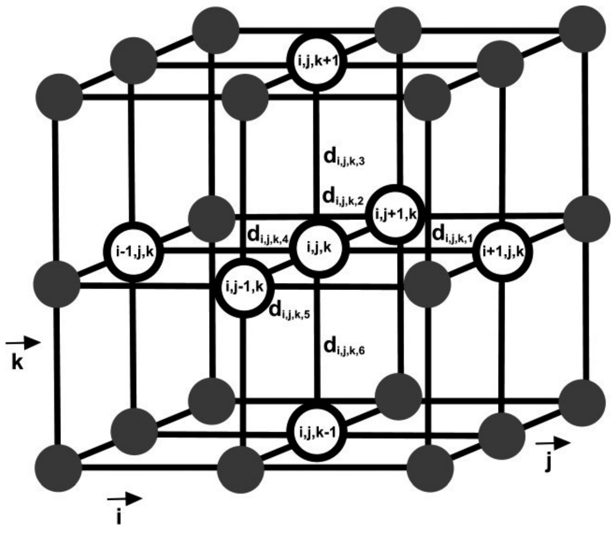

Fig. 5. Illustration of simple network model of membrane.

Virus filtration is a well-established method for the minimization of the inherent risk of viral contamination in the production of therapeutic proteins (Miesegaes et al., 2009). Several recent studies have reported a significant decline in virus retention during the course of filtration through different parvovirus filters (Lute et al., 2007; Hirasaki et al., 1994; Omar and Kempf, 2002; Lutz et al., 2004; Bolton et al., 2005). However, the mechanisms controlling this loss of virus retention are still not well understood (Bakhshayeshi et al., 2011). Our hydrodynamic model is able to simulate the path of a virus in the porous structure of a membrane. To reach relevant and objective results, a high number of viruses have to be tested on an adequate three dimensional network of the membrane. Because of the very small size of virus particles, the path of a virus in a membrane is random. This and other mentioned facts distinguish the third case study from the previous two and make the model stochastic. It means that partial tasks of the parallel program can be absolutely mathematically independent with very small need of process communication. To describe porous structure of the membrane, parameters like pressure and diameter of the pores were used. A simple network model of the membrane is shown in Fig. 5.

The movement of virus particles is affected only by hydrodynamics. The presented case study was run on an eight cores computer processor. As in the first case study, two different languages (platforms) were used to highlight the benefits of parallelization. The first chosen language was Matlab and the second one was C# utilizing the MPI.NET library. The parallelization strategy was based on a parallel for loop. Endurance testing was based on flow simulation of 100 mil. of virus particles through a network of the size of 20 × 20 × 20. Dependence

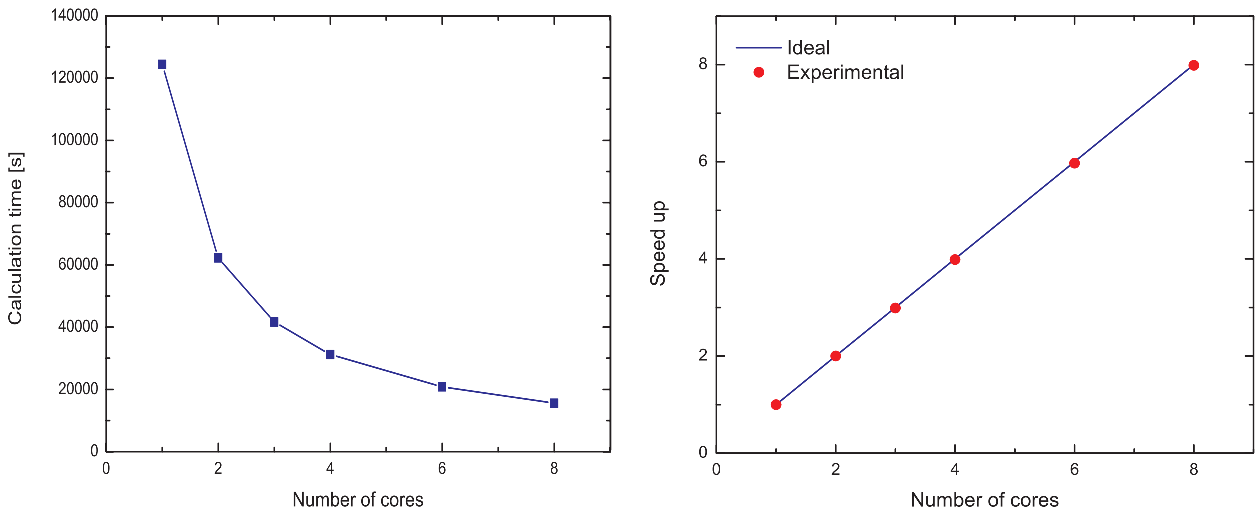

Fig. 6. Execution time and speedup as a function of the number of cluster cores (Matlab).

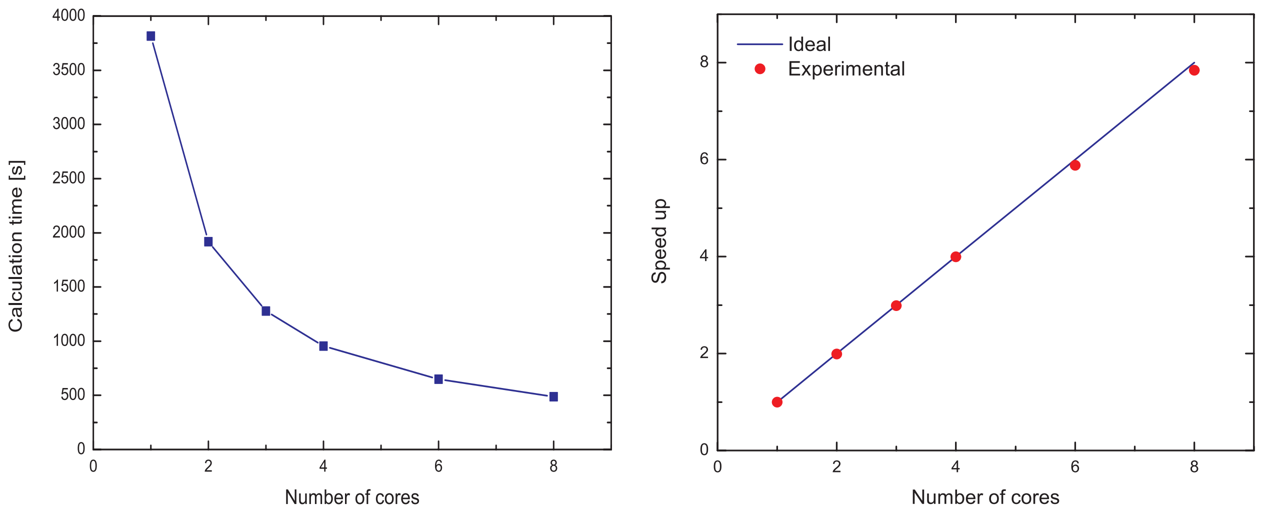

Fig. 7. Execution time and speedup as a function of the number of cluster cores (C#).

of the execution time on the number of computer processor cores and the speedup of the parallel code in Matlab and C# language are depicted in the series of figures below (Figs. 6 and 7).
From Figs. 6 and 7 it is clear that the speedup of the parallel optimization is quite linear and the capacity utilization of computer sources is higher than 99 % in both cases.

Conclusions

In this paper, utilization of parallel computing in chemical engineering simulations was investigated. Several examples of the benefits of HPC were presented and their comparison with serial equivalents is provided. To highlight the benefits of parallelization, two different languages (platforms) were used. The first chosen language was Matlab including Parallel Computing Toolbox™ and the second chosen platform was the .NET Framework and the programming language was C# using the MPI.NET library. The routine problem of chemical engineering praxis – fitting EOS parameters was investigated in the first presented case study. The speedup of parallel optimization was quite linear and the capacity utilization of computer sources was higher than 90 % for both programming languages. The second case study demonstrated benefits of parallelization of solving a large nonlinear equations system. This case study focused on the steady state simulation of a distillation column. In the case of the largest system (2500 trays), the measured speedup was higher than 55, using an 80 cores cluster. The most notable speedup of parallelization was reached in the third case study. The reason of such high speedup was the strictly stochastic character of our hydrodynamic model of virus filtration. From the presented case studies it is clear that parallel computing is an effective tool for solving chemical engineering problems. Despite the fact that modern operating systems and hardware are very well prepared for such applications, except for ANSYS and PROCAST programs, parallel computing has not yet been implemented in standard simulation programs.

Acknowledgement
This work was supported by the Slovak Scientific Agency, Grant No. VEGA 1/0749/15, the Slovak Research and Development Agency APP-14-0317and by the OP Research and Development of the project University Science Park STU Bratislava, ITMS 26240220084, co-financed by the Fund of European Regional Development.

References

Bakhshayeshi M, Jackson N, Kuriyel R, Mehta A, Reis RV, Zydney AL (2011) Journal of Membrane Science 379: 260.
Bolton G, Cabatingan M, Rubino M, Lute S, Brorson K, Bailey M (2005) Biotechnol Appl Biochem 42: 133–142.
Castier M, Checoni RF, Zuber A (2014) Brazilian Journal of Chemical Engineering 31: 993–1002.
Hirasaki T, Noda T, Nakano H, Ishizaki Y, Manabe S-I, Yamamoto N (1994) Polym J 26: 1244–1256.
Lute S, Bailey M, Combs J, Sukumar M, Brorson K (2007) Biotechnol Appl Biochem 47: 141–151.
Lutz H, Ireland T, Bolton G, Siwak M (2004) BioPharm International 17: 6.
Miesegaes G, Lute S, Aranha H, Brorson K, Flickinger MC (2009) *Encyclopedia of Industrial Biotechnology*: John Wiley & Sons, Inc.
Navarro CA, Hitschfeld-Kahler N, Mateu L (2014) Communications in Computational Physics 15: 285–329.
Omar A, Kempf C (2002) Transfusion 42: 1005–1010.
Perry RH, Green DW, Maloney JO (1997) *Perry's chemical engineers' handbook.* New York: McGraw-Hill.