Proceedings of the Eighth International Conference on
Parallel, Distributed, GPU and Cloud Computing for Engineering
Edited by: P. Iványi, J. Kruis and B.H.V. Topping
Civil-Comp Conferences, Volume 12, Paper 2.5
Civil-Comp Press, Edinburgh, United Kingdom, 2025
ISSN: 2753-3239, doi: 10.4203/ccc.12.2.5

Parallelization of Global Sensitivity Study of Nonlinear Systems Using Komondor HPC

F. Hajdu[1], C. Hajdu[2] and L. Környei[3]

[1]Department of Machine Design, Faculty of Mechanical Engineering, Informatics and Electrical Engineering, Széchenyi István University, Győr, Hungary
[2]Department of Informatics, Faculty of Mechanical Engineering, Informatics and Electrical Engineering, Széchenyi István University, Győr, Hungary
[3]Department of Mathematics and Computational Sciences, Faculty of Mechanical Engineering, Informatics and Electrical Engineering, Széchenyi István University, Győr, Hungary

Abstract

One way to develop a new nonlinear system model is to use its sensitivity study. The global sensitivity study of nonlinear systems requires a lot of parameter combinations and, therefore a lot of calculations. As the differential equations are independent of each parameter combination the process can be accelerated with parallelization. The aim of the research was to parallelize Sobol's sensitivity study of nonlinear systems with a Duffing-type vibration system as an example. Using Komondor HPC different configurations were tested from which the OpenMP with 128 threads gave the best results. With the OpenMP configuration a strong and weak scalability study was performed, and it was shown that the problem is well scalable. Using job arrays 16 computers were utilized and a 1397.95-fold speedup could be achieved. With the results it will be possible to carry out the sensitivity analysis of difficult nonlinear systems effectively.

Keywords: parallelization, sensitivity study, nonlinear system, numerical simulation, HPC, Sobol's method

1 Introduction

With sensitivity study, it is possible to develop a system model. It can be examined how the change in certain parameters affects the system's behaviour [1]. There are 3 main types of sensitivity analysis: screening, local and global sensitivity study [2]. With a global sensitivity study, a detailed parameter space can be examined. However,

the global sensitivity study of nonlinear systems with a lot of parameters requires a lot of calculations, which can be accelerated with parallelization [3].
There are several examples in the scientific literature of solving nonlinear systems numerically and their sensitivity study. Study [4] presents a nonlinear model predictive control algorithm in the case of nonlinear systems like an inverted pendulum and a semi-active damper. Monte Carlo simulations were calculated in parallel. With parallelization using a GPU the results could be calculated within a few milliseconds.
In reference [5] the sensitivity study of bar-space trusses is presented. Single-level and multilevel MPI-based parallel algorithms were used to accelerate the calculations. The parallelization was efficient, using 10 processors an 8-fold speedup could be achieved, which means 83.5% efficiency.
Study [6] proposes a novel methodology and post-processing of agent-based simulations. To accelerate the lot of computations the methods and the post-processing was parallelized using a supercomputer and Univa Grid Engine. With parallelization 273 CPU core days simulations were finished within several hours.
In reference [7] dynamic optimization problems using differential-algebraic equations were accelerated using parallel algorithms. Utilizing 8 cores a 4-fold speedup could be achieved in the case of 13500 differential equations and 75 parameters on a Windows-based system. The algorithm was tested on polymerization and chemical processes.
Study [8] describes PAPIRUS, which is a toolkit for sensitivity and statistical analysis for nonlinear systems. There was a large amount of data, which was broken into a series of instructions and discrete parts with parallelization. To accelerate the calculations multiprocessor workstation and a network of workstations were utilized.
In reference [9] a global sensitivity analysis and parameter identification of a microbial continuous culture is presented. The optimization task was accelerated with a parallel algorithm using broadcasting. After broadcasting a Monte-Carlo simulation was performed on the distributed data.
In reference [10] a new method is developed for sensitivity analysis in the case of optimization problems of dynamical systems. The method was effectively parallelized with a master-slave algorithm using MPI, where the system of differential equations was distributed over more processors.
In study [11] an inverse simulation task is solved with a Bayesian estimator in the case of an earthquake model. Since this method is computationally expensive, a supercomputer was used for the analysis.
Tutorial [12] presents the parallelization of Sobol's and Morris's global sensitivity study in the case of the Lotka-Volterra model. The parallelization can be utilized using a GPU.
Based on the literature review it can be stated that several different tasks related to sensitivity study can be solved effectively using parallelization. However, besides the mentioned tutorial [12] not many studies dealt with the sensitivity study of nonlinear systems using Sobol's method. In this paper Sobol's sensitivity study is performed in parallel in the case of a relatively simple nonlinear system using the Komondor HPC (high performance computing) cluster. The aim of this research was to test the parallelization possibilities of Sobol's sensitivity study of nonlinear systems to carry

out the sensitivity study of more difficult systems in further research. This paper is organised as follows: first, the selected sensitivity study and the chosen nonlinear system are briefly described. After that, the parallel algorithms and the metrics of parallelization are presented. It is followed by the presentation of the achieved results. The paper concludes with further research tasks.

2 Methods

There are several methods for global sensitivity analysis, like FAST and Morris [2]. For this study Sobol's variance-based method was selected as it can give quantitative results via indices [13]. The indices can be calculated as follows:

$$S_T = \frac{var(\mu_T(x_T))}{var(Y)} \tag{1}$$

$$S_i = \frac{var(\mu_i(x_i))}{var(Y)} \tag{2}$$

$$S_{ij} = \frac{var(\mu_{ij}(x_{ij}))}{var(Y)} \tag{3}$$

where $var(\mu_T(x_T))$, $var(\mu_i(x_i))$ and $var(\mu_{ij}(x_{ij}))$ are the conditional variance of a selected parameter and $var(Y)$ is the variance of the output variable [14].
To perform a sensitivity analysis a parameter set and an output variable is necessary. The parameter set is generated with Saltelli's sampling method [14].
To test the parallel program a simple system, a Duffing-type nonlinear vibration system was selected (Figure 1).

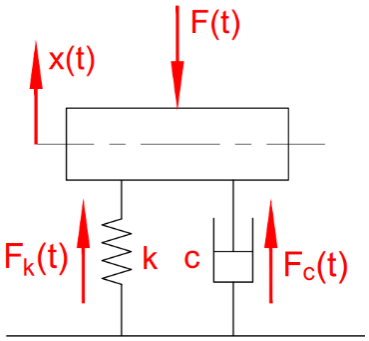

Figure 1: Duffing-type nonlinear vibration system

The system equation can be given by the following formula.

$$\frac{d^2}{dt^2}x(t) + \frac{c}{m}\frac{d}{dt}x(t) + \frac{k}{m}\left(-x(t) + 1000x(t)^3\right)x(t) = Asin(\omega t) \tag{4}$$

where m is the mass, k is the spring stiffness, c the damping coefficient, A the amplitude of the excitation signal, and ω is the angular velocity of the sinusoid excitation signal. The boundaries of the variables are shown in the Table 1.

Variable	Minimum	Maximum
m	100	800
k	10000	60000
c	300	3000
A	0	0.5
ω	1	30

Table 1: Boundaries of the parameters

The root mean square (RMS) of acceleration was selected as the output variable as it is widely used in the case of vibration measurements. For that, the system of differential equations was solved numerically. The simulation time was 50 s and the timestep was 0.01 s. The acceleration was calculated at each timestep and then the results were stored in an array. After that the RMS of acceleration can be calculated with the following formula [15]:

$$RMS = \sqrt{\frac{\sum_{i=1}^{n} a_i^2}{n}} \tag{5}$$

where a_i is the i_{th} value of acceleration and n is the number of acceleration values.
The global sensitivity study of the presented system with 6144 parameter combinations was already carried out using an HP Omen laptop with Intel® Core (TM) i7-7700 HQ CPU @2.80 GHz processor with 8 nodes. For the sensitivity study Python's SALib library was used [16]. The global sensitivity study usually requires a lot more parameter combinations which means a lot of calculations. Since each task (calculation with different differential equation) is independent and there is no shared data the task can be easily parallelized.

First a Python and a C++ program were compared with 192 parameter combinations in the case of 1 node using Komondor. The execution time of the C++ program was 0.461 seconds, while in the case of the Python program, it was 50.87 seconds. Using the same configuration on the aforementioned laptop the simulation runtime was 80.76 s using Python. To accelerate the calculations therefore the C++ program was used for solving the system of differential equations. The parameter generation and the sensitivity study were still carried out with Python [17], only the calculations were performed with an HPC and C++ (Figure 2).

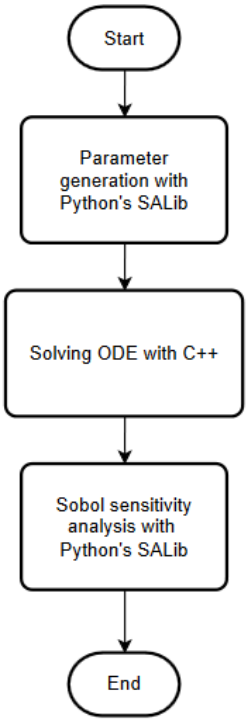

Figure 2: Flow-chart of the sensitivity study program

The parameters are stored in a text file, which is read by the C++ program. BoostODEInt library was used for creating the state variables and solving the differential equation system with Runge-Kutta Dormand-Prince 5 (runge_kutta_dopri5) solver [18]. The results (values of RMS of acceleration) are stored in a vector and then this vector is written to a text file.
In order to find the optimal configuration both OpenMP, MPI, and hybrid programs were tested. The calculation time of solving the system of differential equations was measured by each.
In the case of the OpenMP program *#omp pragma parallel for* was used. The number of threads was set in a batch script. The pseudo-code is shown in Table 2.

```
define system of equations
int main (int argc, char** argv) {
  thread_id=number of threads from batch file
  read input data
  #pragma omp parallel for num_threads (thread_id){
      set parameters for differential equations
      solve differential equation system
      calculate RMS of acceleration
      store results in a vector
  }
  write results to a txt file
  return 0
}
```

Table 2: Pseudo code of the OpenMP program

In the case of MPI first the MPI environment is initialized. Then the vectors, which store the parameters are initialized. Then the data is distributed amongst the nodes using MPI_scatter. Then the systems of differential equations are solved and the results are stored in a vector. Then the data is collected using MPI_gather. Then the results are written to a text file and the MPI is finalized. The pseudo-code is shown in Table 3.

```
define system of equations
int main (int argc, char** argv) {
  MPI_Init (&argc, &argv)
  int numProc, rank
  MPI_Comm_size (MPI_COMM_WORLD, &numProcs)
  MPI_Comm_rank (MPI_COMM_WORLD, &rank)
  define vectors to store input data
  int dataperproc=number_of_rows/numProc
  if (rank==0) {
     read input data
    }
MPI_scatter (input.data(), dataperproc, MPI_DOUBLE, received.data(),
             dataperproc, MPI_DOUBLE, 0, MPI_COMM_WORLD)
define the vectors to store the results
for (int=0; i<dataperproc; i++) {
    solve the system of equations
    calculate the RMS of acceleration
    }
  if (rank==0) {
     define vector for gathered data
    }
MPI_gather (RMS.data(), dataperproc, MPI_DOUBLE, gathered.data(),
            dataperproc, MPI_DOUBLE, 0, MPI_COMM_WORLD)
if (rank==0) {
   write results to a txt file
  }
MPI.finalize ()
 return 0
}
```

Table 3: Pseudo code of the MPI program

In the case of the hybrid approach first the MPI environment is initialized. Then the vectors, which store the initial data are initialized. Then the data is distributed amongst the nodes using MPI_scatter. Then the data is further distributed amongst the threads with #omp pragma parallel for. Then the vectors that store the results are set and the results are collected using MPI_gather. Finally, the results are written to a text file and the MPI is finalized. The pseudo-code is seen in Table 4.

```
define system of equations
int main (int argc, char** argv) {
  MPI_Init (&argc, &argv)
  int numProc, rank
  MPI_Comm_size (MPI_COMM_WORLD, &numProcs)
  MPI_Comm_rank (MPI_COMM_WORLD, &rank)
  thread_id=number of threads from batch file
  define vectors to store input data
  int dataperproc=number_of_rows/numProc
  if (rank==0) {
     read input data
    }
MPI_scatter (input.data(), dataperproc, MPI_DOUBLE, received.data(),
                           dataperproc, MPI_DOUBLE, 0, MPI_COMM_WORLD)
define the vectors to store the results
#pragma omp parallel for num_threads (thread_id){
    solve the system of equations
    calculate the RMS of acceleration
    }
  if (rank==0) {
     define vector for gathered data
    }
MPI_gather (RMS.data(), dataperproc, MPI_DOUBLE, gathered.data(),
                        dataperproc, MPI_DOUBLE, 0, MPI_COMM_WORLD)
if (rank==0) {
   write results to a txt file
  }
MPI.finalize ()
 return 0
}
```

Table 4: Pseudo code of the hybrid program

The metrics to measure the efficiency of a parallel algorithm are among others speedup, efficiency and scalability. The speedup (S) is the ratio of the sequential execution time (T_s) to the parallel program execution time (T_p) [19].

$$S = \frac{T_s}{T_p} \tag{6}$$

Efficiency (E) is the ratio of the speedup (S) and the number of resources used (n) [20]:

$$E = \frac{S}{n} \tag{7}$$

Strong scaling shows how increasing the number of resources affects performance in the case of the same task size [21]. Weak scaling shows how the same amount of jobs is performed on different numbers of resources [22].
For the tests, Komondor HPC (184 nodes, each 2 pieces of 64 core AMD EPYC™ 7763 (Milan) CPU and 256 GB RAM (23552 core in total), 200Gb/s Slingshot interconnect) was used.

3 Results

First, it was examined which configuration gave the best results from OpenMP, MPI, and hybrid approach. There was 786432 parameter combinations. 5 tests were run to find out the best combination. The time was measured with Chrono library's high-resolution clock. The runtime of solving the differential equations was measured, therefore the starting time was right before entering the cycle and the end time was after the cycle. The results are shown in Figure 3.

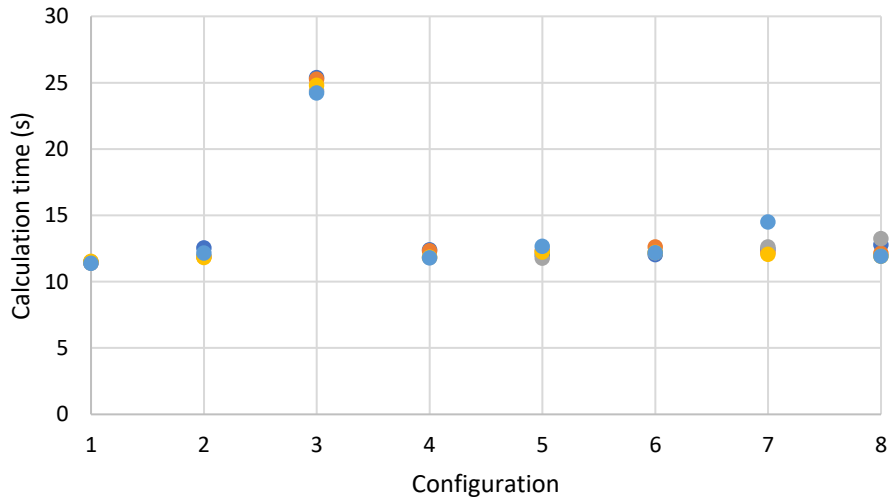

Figure 3: Calculation time in the case of different configurations (1: OpenMP128t, 2: MPI128p, 3: hybrid 2x64, 4: hybrid 4x32, 5: hybrid 8x16, 6: hybrid 32x4, 7: hybrid 32x4, 8: hybrid 64x2)

It can be seen that the OpenMP128t was the best combination utilizing a full computer, therefore it was used for further research. A scalability test was carried using OpenMP. The results are shown in Figure 4.

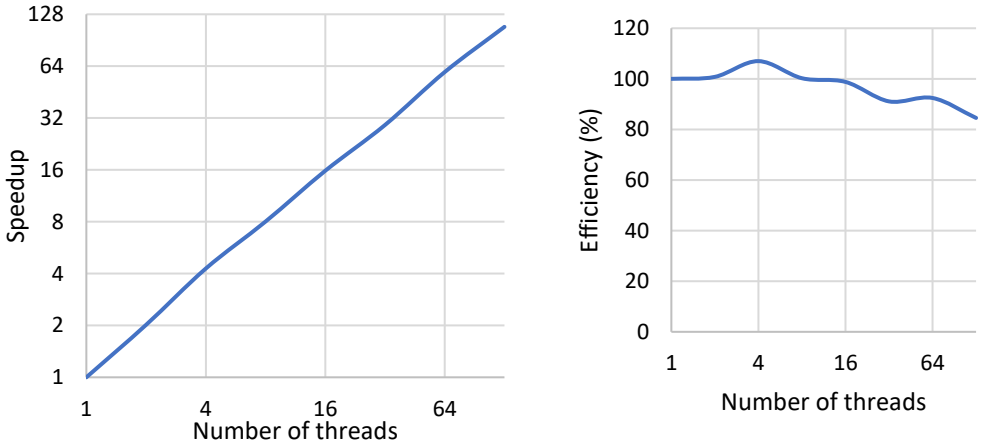

Figure 4: Speedup (left) and efficiency (right) in the case of different number of threads

It can be seen that the problem is scalable, as with increasing the number of threads the speedup increases nearly linearly. A 108.22-fold speedup could be achieved at the largest number of threads. The efficiency decreased to 84.55% as the number of threads increased.
A weak scaling test was also carried out starting with starting with 6144 parameters with 1 thread. The results are shown in Figure 5.

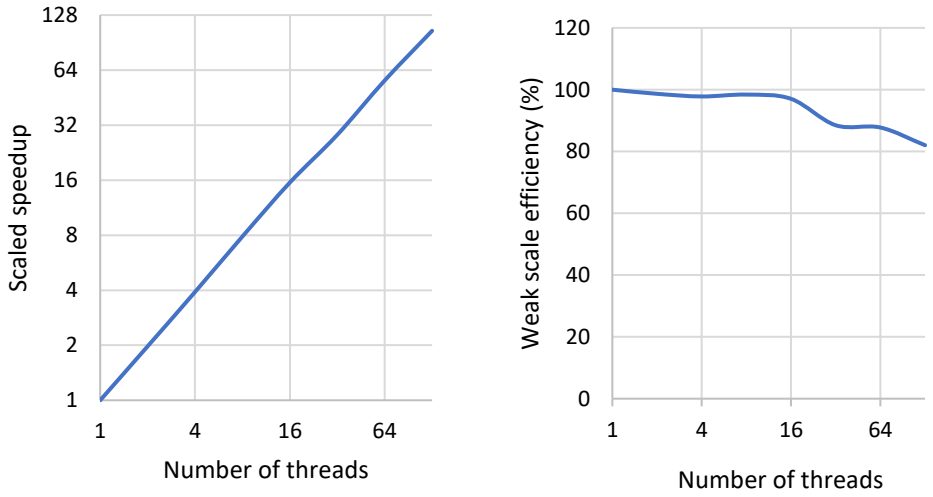

Figure 5: Scaled speedup and weak scale efficiency in the case of using OpenMP

It can be seen that the weak scaling test is nearly linear. However, the weak scale efficiency decreased as the number of threads increased.
As there are more difficult systems and even more parameters it might be necessary to distribute the calculations on more computers. It can be done using OpenMP and array jobs. A test run was performed using more computers. The results are shown in Figure 6 and Figure 7.

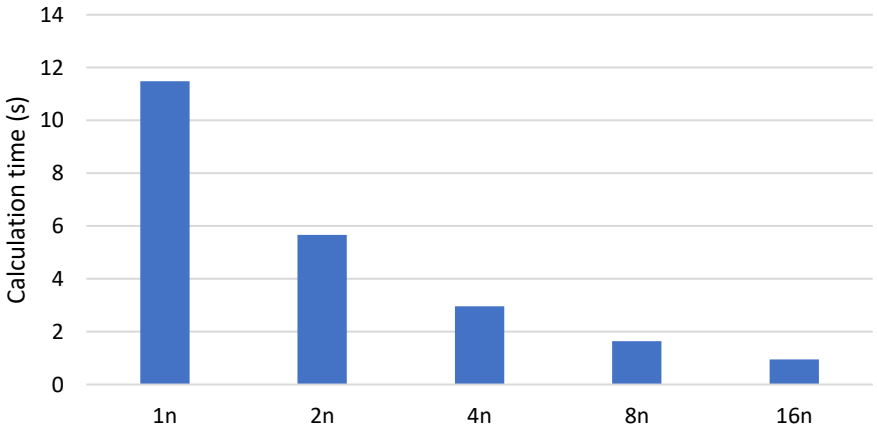

Figure 6: Calculation times with more computers using array jobs

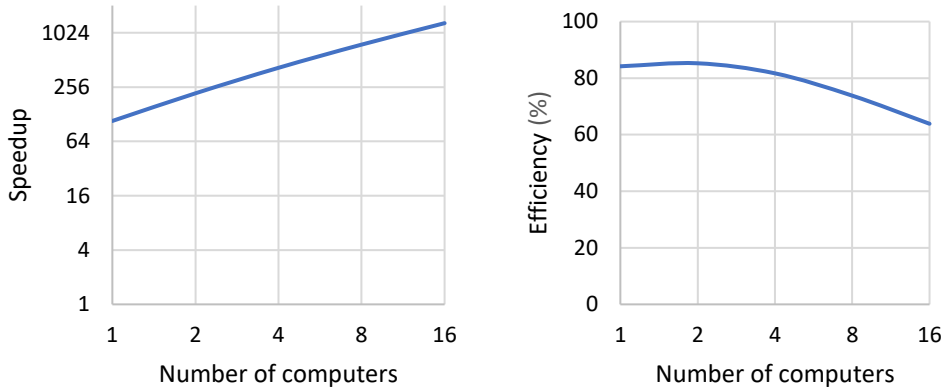

Figure 7: Speedup (left) and efficiency (right) in the case of different number of computers using array jobs

The calculation time decreased from 11.482 to 0.946 s. It can be seen that a linear speedup could be achieved. A 1397.95-fold speedup could be achieved using 16 computers. It means that even a higher amount of speedup could be achieved using more computers. As it was a test to examine the performance of array jobs no further computers were utilized. Besides the efficiency also decreased from 85% to 63%. It is the task of further research to find out the cause of it. The achieved results are still promising because using HPC 786432 parameter combinations could be solved in less than a second. In the case of more difficult systems, it can save significant time.
The results of the sensitivity analysis using 786432 parameter combinations is shown in Figure 8.

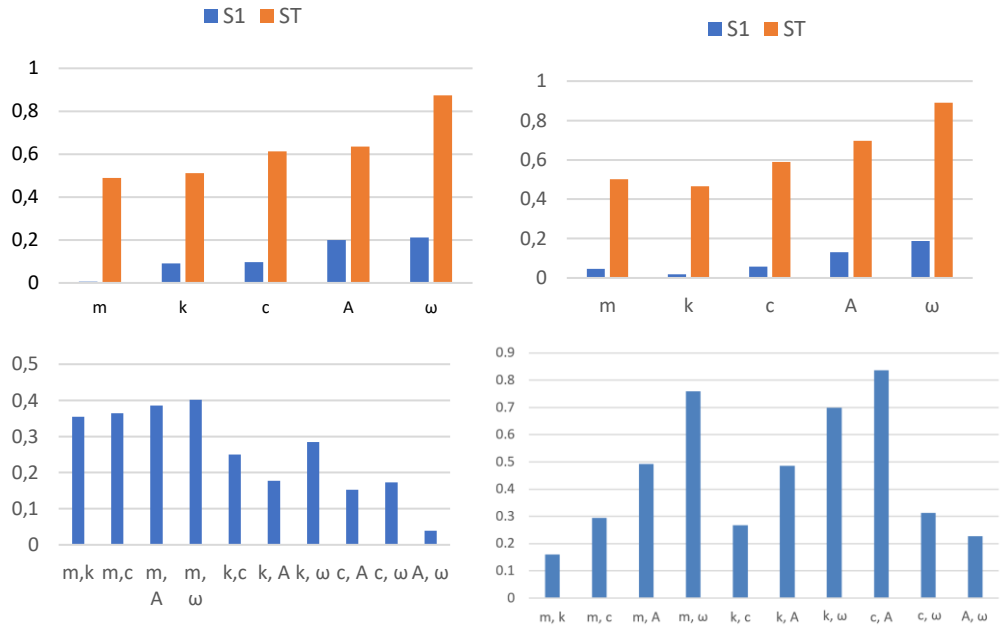

Figure 8: Sensitivity indices in the case of 786432 parameter combinations (left) and 6144 parameter combinations [16] (right) (above: first order and total sensitivity indices, below: second order sensitivity indices)

It can be seen that more parameter combinations changed the sensitivity indices, and even the order of parameter sensitivity changed. The least influential parameter was the mass and the most influential parameters were the amplitude and the angular velocity of the excitation signal. The second-order sensitivity indices also became smaller, which means that there were fewer parameter interactions. However, it was still larger than the first-order sensitivity indices, which means that the parameter interactions had more influence on the RMS of acceleration, than a single parameter alone.

To summarize the results for solving a lot of different differential equation systems with different parameters the OpenMP with 128 threads configuration gave the best results using Komondor HPC. A scalability test was also performed including weak and strong scaling from which it was observed that the problem is scalable, however the efficiency decreased as the number of threads increased. A test was also performed in which more computers were utilized using array jobs. With this even more computing resources could be utilized and even a larger speedup (1397.95-fold) could be achieved. With the presented study it was shown that the global sensitivity study of nonlinear systems can be effectively parallelized in the case of a simple system. From the results and the experience from the presented study, the sensitivity study of more difficult nonlinear systems with much more parameters can be effectively carried out. As seen in the Introduction several studies dealt with the parallel global sensitivity study of different systems, but they were mostly biological systems. In this study, the global sensitivity study of a nonlinear vibration system was carried out in parallel. Also, no examination of different configurations was carried out in the related scientific paper. The main drawback of the presented study is that it requires a lot of calculations, therefore it might cost a lot of CPU hours. Another limitation of the current study is that it was only tested on a relatively simple system. An important further research task is to carry out the sensitivity study of more difficult nonlinear systems.

4 Conclusions

In this paper, the parallelization of the global sensitivity study of nonlinear systems was presented using a nonlinear Duffing-type vibration system and the RMS of acceleration as the output variable. The most time consuming-part, which is solving the differential equation system was parallelized using a C++-based program on an HPC. It was tested which configuration was the best from pure OpenMP, pure MPI, and hybrid approaches. It was found that the program with OpenMP could solve the system of equations the fastest, therefore it was selected for further examinations. Strong scaling and weak scaling tests were performed using OpenMP. It was found that the problem is well scalable as the speedup and the scaled speedup increased nearly linearly as the number of threads was increased. A test was also performed to utilize more computers for the calculations using job arrays. It was found that using 16 computers with 128 threads a 1397.95-fold speedup could be achieved. This speedup showed that job arrays can be effectively used for solving a lot of differential equations with a lot of different parameter combinations. As the results are promising further task is to carry out the global sensitivity study of more difficult nonlinear

systems like a fire truck suspension and different mobile robots. Another research task is to utilize GPU as well.

Acknowledgements

The Authors acknowledge KIFÜ (Governmental Agency for IT Development, Hungary) for giving us access to the Komondor HPC facility based in Hungary.

References

[1] A. Saltelli, K. Aleksankina, W. Becker, P. Fennell, F. Ferretti, N. Holst, S. Li, Q. Wu, "Why so many published sensitivity analyses are false: A systematic review of sensitivity analysis practices", Environmental Modelling & Software 114, 29-39, 2019

[2] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola, "Global Sensitivity Analysis" The Primer, John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, England, 2008

[3] D. Bartuschat, U. Rüde, "A scalable multiphysics algorithm for massively parallel direct numerical simulations of electrophoretic motion", Journal of Computational Science 27 147-167, 2018

[4] S. Ohyama, H. Date, "Parallelized nonlinear model predictive control on GPU", in "Proceedings of 11th Asian Control Conference (ASCC) ", 1620-1625., 2017 https://doi.org/10.1109/ASCC.2017.8287416

[5] P. K. Umesha, M. T. Venuraju, D. Hartmann, K. R. Leimbach, "Parallel Computing Techniques for Sensitivity Analysis in Optimum Structural Design", Journal of Computing in Civil Engineering 21 (6) 463-477, 2007

[6] A. Niida, T. Hasegawa, S. Miyano, "Sensitivity analysis of agent-based simulation utilizing massively parallel computation and interactive data visualization", PLoS One 14 (3), Paper e0210678, 2019

[7] A. Hartwich, K. Stockmann, C. Terboven, S. Feuerriegel, W. Marquardt, "Parallel sensitivity analysis for efficient large-scale dynamic optimization", Optimization and Engineering 12, 489-508, 2011

[8] J. Heo, K. D. Kim, "PAPIRUS, a parallel computing framework for sensitivity analysis, uncertainty propagation, and estimation of parameter distribution", Nuclear Engineering and Design 292, 237-247, 2015

[9] K. Gao, X. Zhang, E. Feng, Z. Xiu, "Sensitivity analysis and parameter identification of nonlinear hybrid systems for glycerol transport mechanisms in continuous culture", Journal of Theoretical Biology, 347, 137-143, 2014

[10] R. Serban, "A parallel computational model for sensitivity analysis in optimization for robustness", Optimization Methods and Software 24, 105-121, 2009

[11] K. Nakao, T. Ichimura, K. Fujita, T. Hori, T. Kobayashi, H. Munekane, "Massively parallel Bayesian estimation with Sequential Monte Carlo sampling for simultaneous estimation of earthquake fault geometry and slip distribution", Journal of Computational Science 81, Paper 102372, 2024

[12] V. K. Dixit, C. Rackauckas, "GlobalSensitivity.jl: Performant and Parallel Global Sensitivity Analysis with Julia", Journal of Open Source Software 7 (76), Paper 4561, 2022
[13] M. Tosin, A. M. A. Côrtes, A. Cunha, "A Tutorial on Sobol' Global Sensitivity Analysis Applied to Biological Models", in: da Silva F.A.B., Carels N., Trindade dos Santos M., Lopes F.J.P. (eds), "Networks in Systems Biology: Applications for Disease Modeling", Springer International Publishing, 93–118, 2020
[14] A. Saltelli, "Making best use of model evaluations to compute sensitivity indices", Computer Physics Communications 145 (2), 280–297, 2002
[15] M. Arraigada, M. Partl, "Calculation of displacements of measured accelerations, analysis of two accelerometers and application in road engineering", in "6th Swiss Transport Research Conference (STRC 2006) ", 2016
[16] F. Hajdu, "Global Sensitivity Study of a Duffing-Type Nonlinear Vibration System", Strojnicky Casopis / Journal of Mechanical Engineering 74 (2), 17-24., 2024
[17] "SALib - Sensitivity Analysis Library in Python" https://salib.readthedocs.io/en/latest/user_guide/basics.html (acessed: 04.08.2024)
[18] K. Ahnert, M. Mulansky , "Odeint – Solving Ordinary Differential Equations in C++", AIP Conference Proceedings 1389 (1) 1586–1589, 2011
[19] L. Környei, G. Kallós, D. Fülep, "Parallel Computations on the Blade Server at Széchenyi István University", Acta Technica Jaurinensis, 3 (1), 111-126, 2010
[20] G. Lencse, I. Derka, "Testing the Speed-up of Parallel Discrete Event Simulation in Heterogeneous Execution Environments", in "Proceedings of the the ISC'2013, 11th Annual Industrial Simulation Conference", pp. 101-107, 2013
[21] F. Magoules, F.-X. Roux, P. Iványi, "Parallel Calculation Methods (in Hungarian) ", Pollack Press, Pécs, Hungary 2018
[22] M. Aldinucci, V. Cesare, I. Colonnelli, A. R. Martinelli, G. Mittone, B. Cantalupo, C. Cavazzoni, M. Drocco, "Practical parallelization of scientific applications with OpenMP, OpenACC and MPI", Journal of Parallel and Distributed Computing 157 13-29, 2021