SIMULTANEOUS OPTIMISATION: STRATEGIES FOR USING PARALLELIZATION EFFICENTLY

Wünsch, Andreas; Jordan, André; Vajna, Sándor
Otto-von-Guericke University Magdeburg, Germany

Abstract

Efficiency plays a major role in any facet of product development. The product has to be efficient itself, but the processes of product development have to be efficient as well. Analysis and simulation enable engineers to evaluate new products without the need for physical prototyping and improve them by optimisation methods. Since parallel computing was introduced the process of product optimisation became more efficient. However, it is very susceptible to bottlenecks like any the lack of available resources.
We transfer the approach of simultaneous engineering to an optimisation framework. The term simultaneous optimisation is introduced in order to use the available resources in an efficient way and to reduce idleness of the resources. To verify the framework it was tested in an industry-related workstation cluster. In this approach resources are floating and can become available or busy in order to the current use of a machine or the unavailability of licenses. The presented approach works in both homogenous and heterogeneous workstation clusters.

Keywords: Concurrent Engineering, Design informatics, Integrated Product Development, Optimisation, Simulation

Contact:
Andreas Wünsch
Otto-von-Guericke University Magdeburg
Chair of Information Technologies in Mechanical Engineering
Germany
andreas.wuensch@ovgu.de

Please cite this paper as:
Surnames, Initials: *Title of paper*. In: Proceedings of the 20th International Conference on Engineering Design (ICED15), Vol. nn: Title of Volume, Milan, Italy, 27.-30.07.2015

1 INTRODUCTION

Markets are becoming increasingly competitive. In order to sustain market share, organisations have to work in a customer-oriented approach for designing and producing high-quality, high-value products. Analysis and simulation enable engineers to evaluate new products without the need for physical prototyping. The area of computer aided engineering has seen an enormous growth in the past years. It has established itself as an important part of the product development process.
However, efficiency plays a big role in this process in any facet. Obviously the product has to be efficient itself in the use phase, but as well in the manufacturing phase and in the end of life phase. Furthermore, mostly from the company's perspective the product development process has to be efficient, too. Bottlenecks should be avoided and the available resources should not become idle.
The research question can be formulated like this: How can parallelization methods be used to solve optimisation problems in an efficient way regarding available resources?
The structure of this paper is as follows. Firstly, we present how parallelization is used in the product development process. Then, we focus on the application of parallelization methods in optimisation problems.
Since the activities of a product developer can be seen as a continuous optimisation processes, we transfer the approach of simultaneous engineering to an optimisation framework by using an open source Software for High Throughput Computing.
The term simultaneous optimisation is introduced in order to use the available resources in an efficient way. This approach works in both homogenous and heterogeneous workstation clusters. A homogenous cluster is defined as one having identical workstations connected by a switch, whereas a heterogeneous cluster is one where the workstation hardware and available software are not identical. Finally, a case study and some concluding remarks are presented.

2 PARALLELIZATION OF ENGINEERING PROCESSES

Working on tasks in a serial way is the most intuitive and natural way of working. In this case only a little amount of organisation and planning is needed. One of the first models to be proposed is the waterfall model that was introduced by Royce (1970) to prescribe software development activities. In this model the stages are depicted as cascading from one to another. So, one development stage has to be completed before the next begins. The waterfall model presents a very high-level view on the processes during development and it suggests to developers the sequence of activities and processes they should expect to encounter. Milestones and deliverables are associated with each process activity. Project managers can use the model easily to gauge how close the project is to the completion at a given point in time (Pfleeger, 1998).
Simultaneous engineering (SE) describes a philosophy for parallelizing engineering work during the product development process and has been proposed as a potential to improve product development practice. It involves simultaneously satisfying the functionality, reliability, manufacturability, and marketability concerns of new products in order to reduce product development time, to achieve higher product quality and value, and to reduce the time taken to introduce new products (Molina et al. 1995). In this context the objectives focus on (Winner et al. 1988, Nevins and Whitney 1989):

- Reduction of development lead times of the product
- Improvement of quality of the product
- Reduction of life cycle costs of the product

Winner et al. (1988) define SE as a systematic approach to the integrated and concurrent design of products and their related processes including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality, cost, schedule, and user requirements.
Cleetus (1992) has proposed a new definition of SE or concurrent engineering (CE). In his definition CE is a systematic approach to integrated product development that emphasizes response to customer expectations and embodies team values of cooperation, trust, and sharing in such a manner that decision making proceeds with large intervals of parallel working by all life cycle perspectives, synchronised by comparatively brief exchanges to produce consensus.

Since SE and CE were introduced many papers dealt with research on computer aided systems to support simultaneous engineering and its collaborative processes. These papers were presented, reviewed, and classified by Molina et al. (1995).
In both previously introduced definitions there is no separation between SE and CE. Both concepts are quite similar. However, the authors of this paper distinguish between SE and CE. In product development parallel processing can be done either by SE or CE or the combination of both. In the SE philosophy different activities can be overlapped and can be done in parallel to work efficient in order to costs and time, e.g. design processes and the planning of the manufacturing processes can be done in parallel.
Within CE, a complex task is divided among several people of a team. The team members work in parallel on their particular piece of the complex task. In CE the definition of physically and logically demarcated areas is necessary with clear interfaces (e.g. design spaces), which are merged and matched at the end of an activity and compared with each other (Figure 1).
The most important criterion for parallelizing processes and activities by SE and CE is the question when the results of the previously begun activity are stable so far, that the statistical probability of a change and the associated change in costs are lower than the costs, which are caused by continue working too late (Vajna et al., 2009).

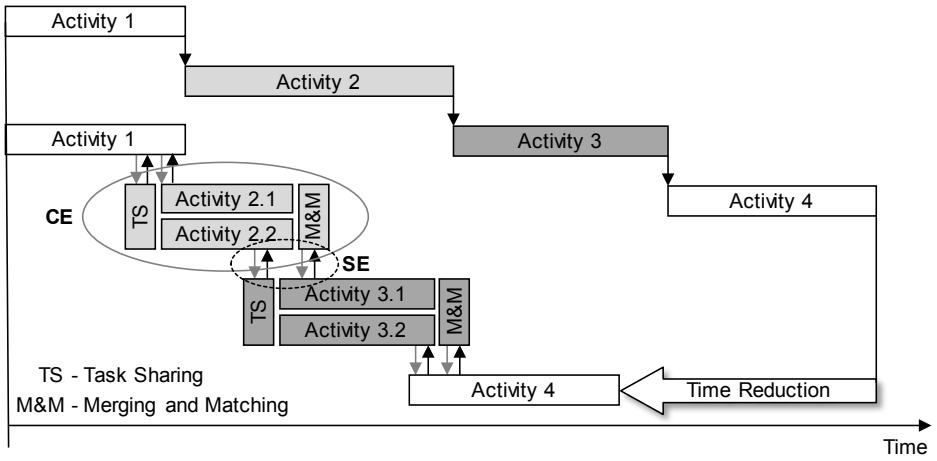


Figure 1. Simultaneous engineering and concurrent engineering (Vajna, 2014)

3 PARALLELIZATION METHODS IN OPTIMISATION

Optimisation problems often lead to high computational cost, especially when numerical simulation is used to evaluate the different solutions of a product (e.g. finite element analysis, computational fluid dynamics). One of the general aims of an optimisation problem is to find the global optimum of the particular problem. Stochastic optimisation methods can help to find the global optimum.
Contrary to deterministic optimisation methods stochastic methods deliver a high probability to find the global optimum. However, stochastic methods yield to additional high computational cost. The most important stochastic methods are Genetic Algorithms (GA), Monte Carlo, Particle Swarm, and Simulated Annealing. Due to the fact that stochastic methods work stepwise, the different evaluations are independent from each other. Since the single evaluations are independent, different methods and frameworks have been established to reduce the computational cost. These methods are presented in the following section including their strengths and weaknesses. Furthermore we introduce a new strategy for parallelization: Simultaneous Optimisation (SO).

3.1 Parallelization in Optimisation

Genetic algorithms (GA) are naturally parallel, since the individuals are evaluated per generation. The individuals of one generation are computed and analysed independently. Two popular approaches exist to configure the system when parallel processing is combined with GA's:

- Master-slave model
- Island-migration model

The most popular approach is the master-slave model, where the master process directs the optimisation process and the program flow by assigning tasks to the slave processes. Typically, the slave processes evaluate the individuals and compute the fitness values. The master process uses this information and creates the next generation.
In the island-migration model, the entire population is divided into subpopulations (so-called islands) that are associated with different processors or machines. The best individuals are exchanged between the subpopulations (so-called migration) when disparate populations evolve periodically (Rajan and Nguyen, 2004). In this paper island-migration is not considered, since this is leads to increased effort in the configuration of the algorithm, but we can mention that the introduced method will work in island-migration models, too.
As stochastic optimisation methods design space exploration methods like Design of Experiment studies (DoE) consist of independent evaluations. That makes them very applicable for parallelization. Tzannetakis and Van de Peer (2002) present an approach to use optimisation combined with parallel-generated surrogate models. In this approach a number of heterogeneous workstations in a cluster can be assigned in order to execute the DoE plan.
Using a GA two basic approaches of implementing distributed computing were introduced by Fritzsche et al. (2012):

- Parallelization of each individual evaluation of a generation
- Parallelization by using a distributed virtual machine

The first approach is very obvious and similar to the master-slave model. Each complete individual of a generation is sent to a slave that computes the entire evaluation of the related individual. This works only in a homogeneous cluster. All workstations have to have installed all the associated software. The further development of this approach is to share only parts of the evaluation process, which makes it more flexible. But in this approach no flexible resource management is included and the introduced optimisation software has to run as a logged in user on any machine. However, no other user can use a workstation of this cluster spontaneously without interrupting the optimisation run.
The second approach compromises the application of a distributed virtual machine which consists of many different single workstations. The optimisation process runs on this virtual machine with a large amount of available CPU and memory. The disadvantage of this approach is the high administrative effort to make it work stable and to get the software running in this special virtual environment (Fritzsche et al., 2012).
A totally different approach for using parallelization in optimisation methods is to parallelize different algorithms in order to increase the possibility to find the global optimum to the problem. In this multi-algorithm infrastructure different algorithms compete in parallel for a contribution towards a single global stopping criterion. For this kind of parallelization a cluster of up to 128 machines was used (Groenwold and Hindley, 2002). To run this method efficiently a homogenous cluster should be preferred.
The previously mentioned parallelization methods show the wide bandwidth of using parallelization in optimisation and design space exploration. For engineering design problems, from a computational viewpoint, the evaluation of a model is the most expensive step. This evaluation includes the model generation, solving, and result extraction. Regarding a stochastic optimisation method such as a GA this evaluation is called fitness evaluation which delivers a so-called fitness value to evaluate a model or individual. There are two common approaches for parallelizing the fitness evaluation (see Figure 2). In both examples we distinguish between creation of individuals (e.g. creating CAD geometry, meshing, or setting up loadcases) and evaluation (e.g. numerical solving).

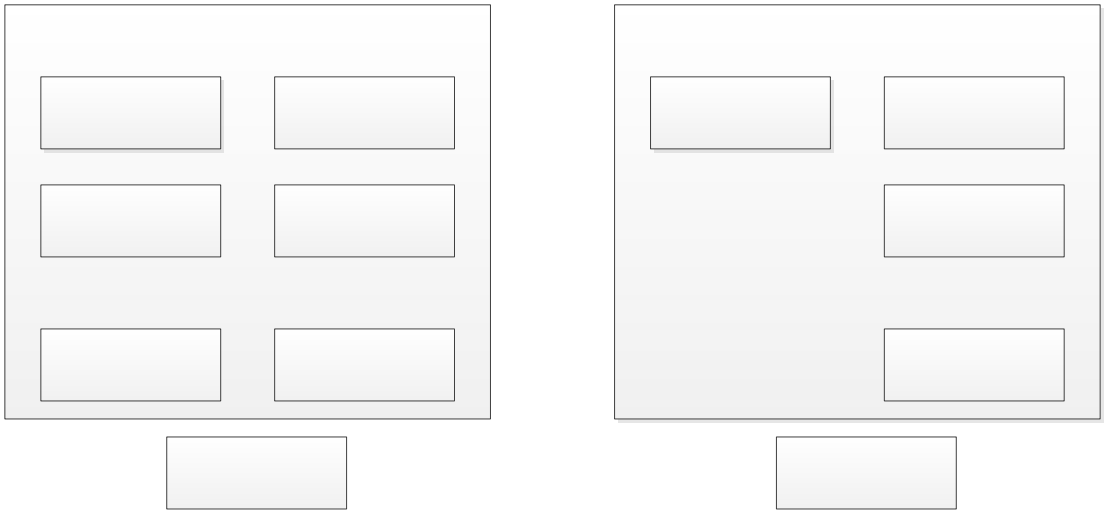


Figure 2. Full parallelization (left) and load case parallelization (right)

The major advantage of both methods is the relatively low administrative effort. The full parallelization of a whole generation is very efficient, because all individuals are independent from each other. The most efficient way is to use the same number of workstations as the population size, which leads to high computational costs in hardware resources and available licenses since population size becomes high. The second risk of this method is the possibility of bottlenecks in the evaluation process, especially when it consists of many different load cases or simulations. The whole process has to wait for the slowest simulation and workstations become idle.
The second method of parallelizing different loadcases is very easy to configure. This method should be very useful for the use in deterministic optimisation methods. Using this method in stochastic optimisations is not very efficient.
However, in real world applications resources play a major role. This includes hardware resources and available licenses. In our opinion a flexible resource management system has to be considered in any parallelization method. The second aim of using parallelization efficiently is to avoid idleness of machines.

3.2 Simultaneous Optimisation and Concurrent Optimisation

To avoid machines become idle due to bottlenecks in the evaluation process and using the available resources efficiently we transferred product development processes to optimisation strategies. The major goal of both approaches is quite similar: using the available resources efficiently. Due to this transfer we introduce two terms: concurrent optimisation and simultaneous optimisation. As in product development processes the concurrent optimisation approach a complex evaluation task is divided among several machines of a cluster (see 3.1). In simultaneous optimisation different parts of the evaluation process can be overlapped and can be done in parallel. The aim of this approach is to parallelize the evaluation process maximally regarding available resources (machines, licences).
An example application is shown in Figure 3. The bottleneck in this example is the creation of new individuals (e.g. meshing due to limited licenses). Evaluation of the individuals is not limited. Due to the limited resource of e.g. only one available license for meshing the optimisation study must be executed serially which is not efficient. The optimisation study can be done more efficient with the simultaneous optimisation approach. In this scenario the creation of a new individual starts when the previous individual was created and the license for meshing becomes available.
The main issues of simultaneous optimisation can be formulated as the following: as soon needed resource becomes available and the needed information and inputs are available as well, the process will be executed.

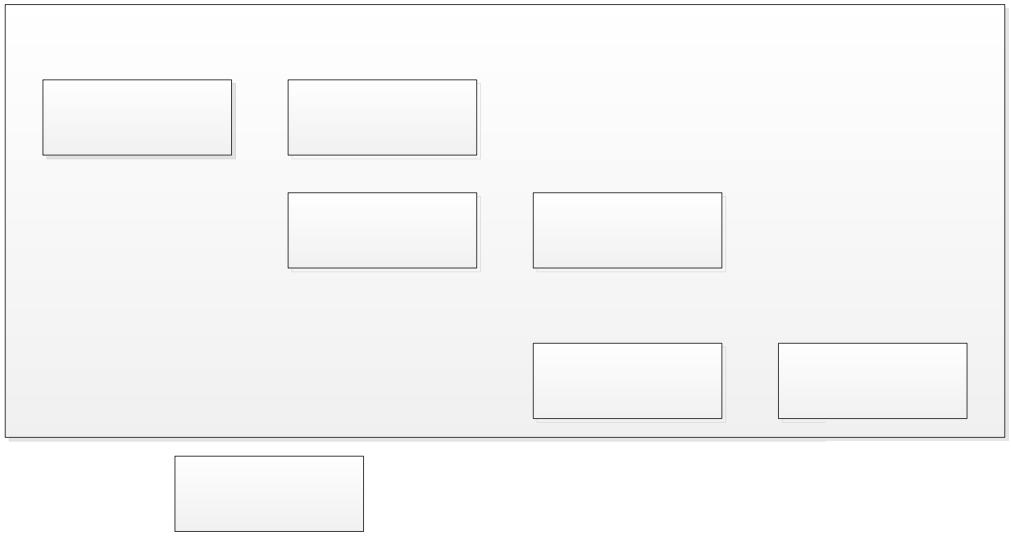


Figure 3. Simultaneous optimisation

As product development processes show, performing tasks concurrently or simultaneously leads to high effort in communication and management, which is similar to simultaneous optimisation, too. To handle this effort we designed a framework that manages the communication and available resources using methods of cluster computing. In this framework the user has just to define the available resources.

3.3 Cluster Computing

High Performance Computing (HPC) or and High Throughput Computing (HTC) are a necessity for solving computationally intensive problems in scientific research, engineering and product development such as numerical simulation and optimisation. Using supercomputers was previously reserved for large institutions because of their very high costs.
In recent years, the technology has evolved so that the performance of commercial workstations has more approximated to the level of supercomputers. Thus, low-cost HPC environments for all use of engineering calculations can be built by using computers interconnected via high-speed networks. In essence, the rapid increase in microprocessor performance and network bandwidth has made clustering a practical, cost effective computing solution which is readily available to the masses. At the hardware level, a cluster is simply a collection of independent systems, typically workstations, connected via a commodity network.
Cluster computing has emerged as practical, cost-effective complement to HPC environments for several reasons (Zomaya, 1996):

- Easy to build up: Anyone with two or more workstations connected via a network can create a cluster. Little or no additional cost is involved.
- A cluster provides a readily available environment for research into parallel computing.
- Unused computing cycles can be scavenged which provides additional computing capacity at no additional cost.
- Robust and stable software systems for clustering are commonly available.

Especially workstation clusters offer many benefits when compared to other computing solutions. The most favourable characteristic is that clustering is relatively simple in implementation and administration, and involved only with no or little additional cost. Workstations are designed to be excellent at serving the interactive job requirements of users. However, most workstations are used for interactive work during normal business hours, too. Due to this fact a plenty number of CPU resources is available in nonprime hours. A properly configured cluster can provide both prime-time interactive cycles and nonprime-time batch cycles which yields to a more cost effective computing environment (Zomaya, 1996).
Although there are many benefits of workstation clusters, they should be seen as practical replacements for general-purpose supercomputers. However, clusters should be treated as complementary components of high-performance computing environments.

Because clusters are relatively simple to configure, it is important to categorize which jobs are most conducive to this environment. Studies by several supercomputing centres have shown that many applications presently executing on supercomputers do not absolutely require supercomputing resources (Zomaya, 1996).
With distributed applications, tasks and calculations are carried out simultaneously on several workstations, in which the tasks are independent from each other. Parallel applications however also expect the same time on multiple machines, but here the tasks are interdependent. To set up, configure and direct the mentioned kinds of cluster a software cluster management system is needed. Most of these systems include a job scheduling tool as well. To find an appropriate solution we formulated the following requirements:

- Platform: Windows 7, 64 Bit
- Using workstations in an undergraduate computer lab
- No disadvantages for students who work on the workstations
- Easy to administrate

Regarding to the requirements we searched for an appropriate solution. The systems we found were evaluated by a simple comparison of advantages and disadvantages (see appendix). Due to its versatility, simplicity, and features of a classic batch system we selected HTCondor (Center for High Throughput Computing, 2014). The major features and its architecture are described in the following section.

3.4 HTCondor

HTCondor is a software system that creates a HTC environment by using the computing power of workstations that communicate over a network to build a workstation cluster. To submit a job to the cluster the job just must be submitted to the software system that finds an available workstation in the cluster and starts running the job on that workstation.
The software has the capability to detect that a machine running a job is no longer available (e.g. when the owner of the machine came back from lunch and started typing on the keyboard). It can checkpoint the job and move the job to a different machine, which would otherwise be idle and which continues the job on this new machine from precisely where it was left off.
There is no need to share a common file system between the workstations. The necessary files are transferred from on workstation to another. Due to this fact machines across an entire enterprise can run a job, including machines in different administrative domains.
To run jobs on a remote machine no login on the remote machine is needed because the software uses its remote system call technology, which traps library calls for such operations as reading or writing from disk files.
Compared to other cluster management systems that attach properties to the job queues themselves, HTCondor provides a resource management by so-called match-making resource owners with resource consumers. That makes it quite easy to handle resources in heterogeneous clusters, e.g. licenses, installed software, different hardware (Center for High Throughput Computing, 2014).
HTCondor uses the Master-Worker (MW) system. The MW principle is very suitable to solve a problem of indeterminate size on a large and unreliable workforce. It is well-suited for parameter searches and optimisation problems where large parts of the problem space may be examined independently.
In this model the master directs the computation with the assistance of as many remote workstations as the computing environment can provide. It contains three components: a work list, a tracking module, and a steering module. The work list is simply a record of all tasks to be done. The tracking module tracks the worker processes and assigns uncompleted tasks. The steering module controls the computation. It examines results, modifies the work list, and provides a sufficient number of worker processes. Due to the fact that the workers could be unreliable e.g. they disappear when machines crash and they reappear as new resources when machines become available. If a worker disappears while processing a job, the tracking module returns the job back to the work list (Thain et al., 2004).
Basney et al. (1999) present a framework for conducting large scale Monte Carlo studies by using HTCondor. Furthermore, the software was already used for a huge number of different problems and applications such as the simulation of engines, neural networks, high energy physics events, computer hardware and software, the behaviour of crystals and randomized optimisation techniques.

4 CASE STUDY

For this study we built up a workstation cluster that consists of 25 workstations (Windows 7 64 Bit, CPU: Intel Xeon E1290 3,60 GHz, 16 GB RAM) in an existing undergraduate computer lab. Similar clusters may be built up at most institutes, where a large number of free standing workstations is available, at very little cost.
This case study consists of a simple optimisation problem using a GA. The evaluation process consists of five processes: on process for meshing and preprocessing and four loadcases of numerical solving. The average processing time of each subprocess is shown in Figure 4.

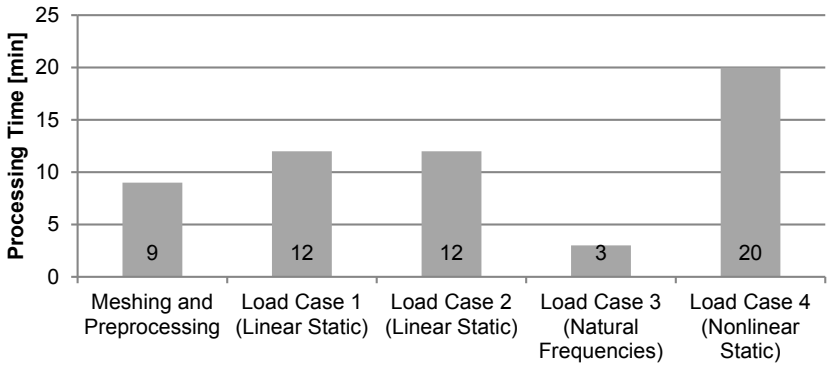
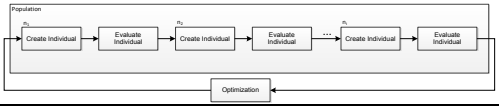
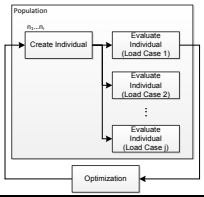
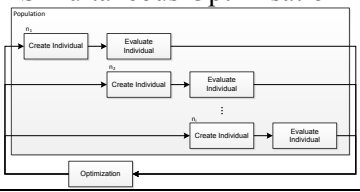
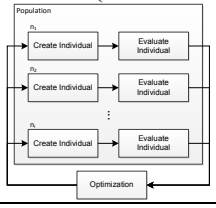


Figure 4. Processing time of the processes of the simulation

The optimisation study is executed in four different ways of parallelization: serial, load case parallel, simultaneous optimisation, and fully parallel (ideal, no bottle necks). Using a GA only the evaluations in a generation are independent from each other. Due to this fact, only the evaluations in a generation can be evaluated in parallel. The population size in this example is 50. The results of the case study are shown in Table 1.

Table 1. Results of the case study

Parallelization Method	Time per Generation	Resources
Serial 	2800 min / 46,6 h	Machines: 1 Lic. PP: 1 Lic. Solv.: 1
Load Case Parallel 	1450 min / 24,16 h	Machines: 5 Lic. PP: 1 Lic. Solv.: 4
Simultaneous Optimisation 	470 min / 7,83 h	Machines: 9 Lic. PP: 1 Lic. Solv.: 8
Fully Parallel (no bottle necks) 	56 min / 0,93 h	Machines: 50 Lic. PP: 50 Lic. Solv.: 50

Obviously, the full parallelization is very fast and efficient, but it needs a lot of resources. It shows the ideal parallelization and is more a theoretical example of the ideal situation. In this study we used simultaneous optimisation very efficient using only one license for preprocessing and 8 licenses for solving. If more licenses or machines were available the processing time per generation would decrease as well. As more resources would be available as more the simulations optimisation would be aligned to the ideal of full parallelization.
The major risks of executing the optimisation fully parallel are bottlenecks. If the number of resources becomes lower than the population size or a factor of it (e.g. a machine fails or a license is needed by another user) the process becomes very inefficient. If a machine would fail in a simultaneous optimisation, the evaluation of one generation needs more time, but in the framework the subprocesses would be distributed on the available machines autonomously. The machines in this framework don't need to be homogenous. The only requirement to any machine is to be able to execute a task of the evaluation process.
The framework was tested, while the computer lab was periodically and spontaneously used by undergraduate students. When a student started to use a machine of the cluster, the machine finished the current job and did not execute any other jobs of the evaluation process. When the machine became free again, it started to execute jobs again. In this scenario the optimisation needs more time, but it keeps using its available resources efficiently. These so-called floating resources could not be used in a similar way in the fully parallel approach.

5 CONLCUSION AND OUTLOOK

In this paper we transferred simultaneous engineering and concurrent engineering from product development to optimisation methods and introduced the terms simultaneous optimisation and concurrent optimisation as an approach to use available resources such as machines and licenses efficiently.
The developed framework was tested in a case study in an existing undergraduate computer lab while students worked periodically and spontaneously on different workstations in this lab. This case study is similar to an industry-related environment where workstations in a cluster could be available while breaks at night or on weekend. The framework works well in both homogenous and heterogeneous network of workstations clusters. Any idle workstation can be implemented in the cluster very easily and users can work on the workstations interactively at any moment. By these facts our approach stands out from other applications in industry.
The simultaneous optimisation approach was developed and tested using a GA since it is easy to parallelize, but it is possible to use it in different stochastic optimisation methods and hybrid optimisation (combination of deterministic and stochastic optimisation) methods as well. This issue merits further research. Further research will also be done on managing hierarchies between tasks of an optimisation study and simulation tasks of the engineer's daily work. If no resource is available to execute a simulation of daily work the optimisation study should vacate and share its resources.

REFERENCES

Basney, J.; Raman, R.; Livny, M. (1999), "High Throughput Monte Carlo", Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas.

Center for High Throughput Computing, University of Wisconsin-Madison (2014), HTCondor™ Version 8.2.3 Manual.

Center for High Throughput Computing, University of Wisconsin-Madison (2014), Computing with HTCondor™, http://research.cs.wisc.edu/htcondor/ (10.11.2014)

Cleetus, K. J., 1992, "Definition of Concurrent Engineering", CERC Technical Report Series, CERC-TR-RN-92-003.

Fritzsche, M., Kittel, K., Blankenburg, A. and Vajna, S. (2012), "Multidisciplinary design optimisation of a recurve bow based on applications of the autogenetic design theory and distributed computing", Enterprise Information Systems, Vol. 6 No. 3, pp. 329–343.

Gordon, R. (2013), Oddjob, http://rgordon.co.uk/oddjob/index.html (10.11.2014)

Grid Engine (2014), Open Grid Scheduler, http://gridscheduler.sourceforge.net/index.html (10.11.2014)

Groenwold, A.A. and Hindley, M.P. (2002), "Competing parallel algorithms in structural optimisation", Structural and Multidisciplinary Optimisation, Vol. 24 No. 5, pp. 343–350.

JPPF (2014), JPPF, http://www.jppf.org/ (10.11.2014)

Molina, A., Al-Ashaab, A., Ellis, T., Young, R. and Bell, R. (1995), "A review of computer-aided Simultaneous Engineering systems", Research in Engineering Design, Vol. 7 No. 1, pp. 38–63.
Nevins, J. L. and Whitney, D. E. (1989), Concurrent Design of Product and Processes: A Strategy for the Next Generation in Manufacturing, McGraw-Hill, New York.
Pfleeger, S.L. (1998), Software engineering: Theory and practice, Prentice Hall, Upper Saddle River, NJ.
Rajan, S.D. and Nguyen, D.T. (2004), "Design optimisation of discrete structural systems using MPI-enabled genetic algorithm", Structural and Multidisciplinary Optimisation, Vol. 28 No. 5, pp. 340–348.
Royce, W.W., "Managing the development of large software systems", in Proceedings of IEEE Wescon 1970, p. 382.
SOS GmbH (2014), JobScheduler, http://www.sos-berlin.com/modules/cjaycontent/index.php?id=62&page=osource_scheduler_introduction_en.htm (10.11.2014)
Thain, D., Tannenbaum, T. and Livny, M. (2004), Distributed Computing in Practice: The Condor Experience.
Tzannetakis, N. and Van de Peer, J. (2002), "Design optimisation through parallel-generated surrogate models, optimisation methodologies and the utility of legacy simulation software", Structural and Multidisciplinary Optimisation, Vol. 23, pp. 170–186.
University of Liverpool (2014), High Throughput Computing using Condor, http://condor.liv.ac.uk/DTCondor/ (10.11.2014)
Vajna, S. (Ed.) (2014), Integrated Design Engineering: Ein interdisziplinäres Modell für die ganzheitliche Produktentwicklung, Aufl. 2014, Springer, Berlin, Heidelberg.
Vajna, S., Weber, C., Bley, H. and Zeman, K. (2009), CAx für Ingenieure: Eine praxisbezogene Einführung, 2., völlig neu bearb. Aufl., Springer, Berlin, Heidelberg.
Winner, R. I., Pennell, J. P., Bertrand, H. E. and Slusarezuk, M. M. G., 1988, "The Role of Concurrent Engineering in Weapon System Acquisition", IDA-Report R-338.
Zomaya, A.Y. (1996), Parallel and distributed computing handbook, Computer engineering series, McGraw-Hill, New York.

APPENDIX

Table 2: Comparison of scheduling software

ID	Name	Platform	Availability	Advantages	Disadvantages	Reference
1	DTCondor	Windows	Freeware	GUI, little code, monitoring, heterogeneous cluster, knowledge of HTCondor not needed	Optimised for application at University of Liverpool	University of Liverpool (2014)
2	HTCondor	Linux, Mac OS X, Windows	Open source	features of classic batch system, flexible resource management, heterogeneous cluster, versatility, full documentation and source code	no GUI, monitoring by external software	Center for High Throughput Computing (2014)
3	Job Scheduler	Linux, Windows, Cloud	Open source	German documentation, web GUI (XML) or console, backup-Cluster, API-Interface to Java, Perl, VBScript	Requirements: MySQL database and http Server, complex user guide	SOS GmbH (2014)
4	JPPF	Java-Environ-ment	Open source	Secure network, prepared for the use of cloud, sophisticated management and monitoring, FTP, no additional configuration	Peculiar API, no details about license and complexity	JPPF (2014)
5	Oddjob	Linux, Windows	Open source	Visual configuration, monitoring and control of jobs over network, XML, configuration files, full documentation	Requirements: Java and database	Gordon (2013)
6	Open Grid Scheduler	Linux, Mac OS X, Windows	Open source	Console, easy monitoring, API-Interface to C/C++, Java, Perl, Python	Complex user guide, little documentation, compilation of source code, no GUI	Grid Engine (2014)