Parallel Software Framework for Fatigue Analyses Based on Software Agents

Andrés Wellmann Jelic, Jochen Bilek, Mozes Gálffy, Dietrich Hartmann

1 Institute for Computational Engineering, Universitätsstr. 150, Ruhr-University Bochum, 44801 Bochum, Germany, Phone +49 234/32-26172, FAX +49 234/32-06172, andres.wellmann@rub.de

Abstract

An estimation of random fatigue processes in structural steel components requires the time-expensive execution of multiple numerical simulations. In the given case of wind loaded structures, the randomness is implied by the stochastic properties of the material and, additionally, the stochastic nature of the oncoming wind. To minimize the necessary computational runtime, a new software framework for parallel computations – based on agent technologies – has been implemented and tested. The parallelization has been accomplished by the usage of mobile software agents which can migrate from their original workstation to other computers.

Introduction

Robust structural design requires the realistic modeling of loads and material properties. For that, both load and material properties have to be represented mathematically according to their real stochastic characteristics. Due to the increasing number of slender structures, also, fatigue analysis of relevant structural details is getting more and more essential. Additionally to the already mentioned stochastic characteristics, a fatigue analysis has to take into account both a time-dependent formulation of actions and stresses, e.g. the non-stationary wind actions or the fatigue of steel. As a consequence, the numerical simulations associated with fatigue analyses result in a high computational as well as time-expensive effort that can only be mastered by means of reasonable simplifications or, preferably, by appropriate computing methods.

The total runtime of time-variant fatigue simulations can be shortened if parallel computing methods are used, by subdividing the total simulation into individual computation parts and distributing these parts on different processors. In the present paper, a new software approach for parallel computing based on multiagent-systems is to be presented. This new approach results from the cooperative work carried out currently in the Collaborative Research Center 398 “Lifetime-oriented design concepts” (CRC398) as well as the Priority Program 1103 “Network-based Co operative Planning Processes In Structural Engineering” (PP1103) at the Ruhr-University.

Main task of the subproject C5 in the CRC398 is the development of a parallel/distributed software framework for lifetime-oriented structural design. One central characteristic of this framework is the stochastic and time-dependent formulation of load processes and material properties. The subproject in PP 1103 aims at the development of an agent-based workbench, modeling and supporting concurrent processes in collaborative structural design.

This contribution mainly presents a parallel computing framework which has been developed and already tested for time-expensive numerical simulations of fa-
tigue processes. Chapter 1 explains the theoretical background of the simulations. In chapter 2, an overview on the design for the parallel computing framework is given. Subsequently, the implementation of the framework in JAVA is shown in the chapter 3. Finally, some preliminary runtime results of this framework are summarized in chapter 4.

**Theoretical background of simulations**

Main topic of the researched numerical simulations is the assessment of fatigue processes at structural steel components. Actually, arched steel bridges (Figure 1a) with vertical tie rods are chosen as a reference system. In particular these tie rods, having circular cross sections, have been proved by Schütz (1992) as highly sensitive against vortex-induced vibrations. Such vibrations cause crack propagation in the connecting plate (see Figure 1b) between the tie rod and the main horizontal girder.

![Reference bridge and connecting plate](image)

A simplified illustration of the creation of vortex-induced vibrations is given in Figure 2. These vibrations occur when vortices are alternately shed from opposite sides of the vertical tie rods, loaded by random wind series with a velocity \( u(t) \). This shedding causes forces in across-wind directions. A dominant role in the excitation process, and therefore in the fatigue process, plays the synchronization of the vortex shedding to the motion of the tie rods for wind velocities within a critical range – called the Lock-In effect. More details on this Lock-In effect and its analytical formulation will be given in the following subsection.

![Vortex-induced vibrations](image)
Each of the abovementioned simulation runs can be decomposed into individual computational stages as follows:

- Stochastic generation of wind series
- Modeling of correspondent across-wind vibrations
- Structural analysis of dynamically loaded structural components
- Estimation of resulting damages

The following subsections will explain these computational stages in more detail.

**Model for wind loading**

At first, wind series have been generated artificially as an input for the load model, based on stochastic stationary processes having a duration of \( d = 60 \text{ min} \) and predetermined statistical values (mean wind velocity \( \bar{u} \), turbulence intensity \( I_u \)). These processes have been adapted to the widely-used *Fichtl-McVehil-Spectrum* (1)

\[
\frac{fS_u(f)}{\sigma_u^2} = \frac{\alpha \frac{fL_{ag}}{\bar{u}}}{1 + b \left( \frac{fL_{ag}}{\bar{u}} \right)^r} \frac{5}{3r}
\]

where \( S_u(f) \) is the spectral density of the wind, \( \bar{u} \) and \( \sigma_u \) are the mean value and the standard deviation of the wind velocity \( u(t) \), respectively.

Depending on these generated wind series, across-wind forces have been computed by means of the enhanced loading model suggested by Gállfy et. al. (2004) which is based on the approach of Lou (1997). This enhanced loading model takes into account both non-stationary and turbulent characteristics of the natural wind. Particularly, the *Lock-In* effect is embedded sufficiently.

The lateral loads due to vortex shedding are described by

\[
F(t) = \frac{\rho}{2} KDC_l L_W \int_0^t u^3(\tau) e^{\alpha(t,\tau)} \cos \varphi(t,\tau) d\tau
\]

with

\[
\alpha(t,\tau) = \int_t^\tau \xi(\theta) d\theta, \quad \varphi(t,\tau) = \int_t^\tau \omega(\theta) d\theta + \varphi_0(t),
\]

\[
\xi(\theta) = \sqrt{\ln 4} I_u \omega(\theta), \quad \omega(\theta) = \frac{2\pi S}{D} \bar{u}(\theta)
\]

The quantity \( \delta \) describes the air density, \( D \) is the diameter of the tie rod, \( C_l \) the aerodynamic lift coefficient, \( \bar{u}(\tau) \) the time-dependent wind velocity and \( I_u \) the turbulence intensity of the wind. \( S \) stands for the Strouhal number and \( \omega(\theta) \) for the Strouhal angular frequency, respectively. \( L_w \) is the length-scale of the synchronized vortex shedding on the tie rod. The quantities \( K \) and \( n \) are fit parameters while \( \varphi_0 \) is controlling the *Lock-In* effect on the wind loading.

Wind series and correspondent lateral loads have been generated artificially for varying parameter sets \( \bar{u} \) (mean wind velocity) and \( y_0 \) (initial displacement of the tie rod).

**Damage assessment**

Based on the abovementioned wind series, correspondent stress-time histories \( \sigma(t) \) have been computed by means of *finite element analyses* (FEA) for discrete material points in the connecting plate. These material points were selected with respect to maximum fatigue sensitivity, based on the relationship between acting and allowable
stresses, respectively. Allowable stresses are defined by notch classes corresponding to the material point.

Damage increments due to the stress histories \( \sigma(t) \) have been estimated by means of an enhanced, stochastically defined hot spot stress (HSS) method. Generally, all stress based proving methods for fatigue – such as the HSS method or the nominal stress concept proposed in standards (e.g. EC3 1993) – comprise the same computational steps as follows.

At first, the maximum allowable number of cycles
\[
N_f(\Delta \sigma_f) = 5 \cdot 10^6 \left( \frac{\Delta \sigma_D, k / \gamma_D}{\Delta \sigma_f \cdot \gamma_f} \right)^m
\]  

for a given stress range \( \Delta \sigma_f \) and a fatigue strength \( \Delta \sigma_D \) is computed, using standardized S-N-curves. The \( \gamma \) quantities are partial safety factors. The partial damages \( d_f \)
\[
d_f = \frac{n(\Delta \sigma_f)}{N_f}, \quad D = \sum_i d_{f,i}
\]  

– defined by the ratio of the actual number of cycles \( n(\Delta \sigma_f) \) and the maximum number \( N_f \) – are accumulated to the total damage \( D \) in the right part of Eq.(5). The collapse of the considered structural component is defined for a Damage \( D_{\text{limit}} = 1.0 \).

In contrast to this standardized fatigue analyses, the enhanced HSS-method describes central parameters of the S-N-curves with stochastic density functions \( f_X(x) \) which have been adapted to experimental results related to fatigue (e.g. Brozzetti 1989). Figure 3 shows a simplified diagram of this modified HSS method. Additionally, the limit damage \( D_{\text{limit}} \) has been introduced as a third random variable, based on the results published in ASCE (1982).

![Figure 3: Enhanced S-N-approach](image)

As a consequence of this stochastic formulation, a damage analysis of each single wind series realization leads to different estimates of damage values. In order to get reliable statistical estimates of the damage values multiple fatigue analyses have to be performed implying a time-expensive computational effort.

**Agent-based concept for parallelization**

To minimize the necessary runtime of the analyses, parallel computation methods are to be employed. At present, several different software solutions for parallel computing already exist, e.g. OpenMP or MPI as parallel programming libraries, or grid-enabling workload management systems such as CONDOR or Platform LSF\(^1\).

Motivation for new concept
The motivation for a new parallelization concept arose from several prerequisites. At first, the existing hardware at the author’s institute – for the parallel computations – is a heterogeneous CAE-pool with 49 PCs, each computer running under Windows or Linux, respectively. This distributed system already eliminates the usage of software solutions with shared memory such as OpenMP. Furthermore, the PCs are mainly used by students switching between the available operating systems so that a platform independent software solution for heterogeneous computing resources is required. Another restriction arises from the already developed software modules in the subproject C5 of CRC398. The individual software modules are implemented in different programming languages – like JAVA or C++. Also commercial software products such as MATLAB are applied which have to be started externally as batch jobs. Finally, an Open Source solution was favored in order to improve and adapt the parallelization framework with respect to specific needs in the CRC398.

Based on these restrictions, an efficient and extendable software solution was examined. The development of a new parallelization framework was alleviated by means of already existing software modules for distributed working in the scope of the already mentioned Priority Program PP1103.

Software design
The agent technology has been proved as very favorable for the parallel usage of different PCs. Software agents are autonomous software-units performing specifically assigned tasks, eventually in interaction with other agents in the computer network. The cooperation of individual agents provides the complete solution of a researched problem, finally. These software agents require an agent runtime environment, a so-called agent platform, which offers the necessary communicational infrastructure and further services. Specialized software agents are mobile agents which can migrate from one agent platform to another in order to fulfill their tasks. The linking of individual agent platforms into one single unit results in a dynamic multiagent system (MAS), which easily can integrate all available personal computers. To this end, an agent platform must be installed on all PCs running as a service in the background. A more extensive introduction on software agents and multiagent systems can be read in Weiss (1999).

The parallelization framework mainly consists of one central master-agent (MA) and several mobile slave-agents (SA). The MA provides a graphical user interface (GUI) by which the user can control and store all information needed for the execution of the numerical simulations. On each of the available computers in the cluster, the corresponding SA is charged with a specific task, representing a partial computation within the complete numerical simulation. These specific tasks are distributed according to a certain algorithm named taskscheduler. According to its task, the SA computes the correspondent results and returns them to the MA. Afterwards, the SA waits for further tasks until all simulations have been executed. Once all tasks are accomplished, the SAs can be deleted.

A strongly simplified view of this framework is summarized in Figure 4. The benefits of this new parallel framework are as follows:

- Reliability: When a computer with a running SA crashes, the task is reassigned to other available machines.
- Monitoring: The user can overview and control the current state.
- Simple scalability: Complicated parallel methods don’t have to be necessarily implemented in the programming language.
- Platform independency: This framework is implemented in Java which is available for most operating systems.
- Flexibility: New SAs can be added dynamically during runtime.

![Image of basic concept of the agent-based parallelization framework]

**Implementation**

Based on the conceptual design presented above, a prototype of the parallelization framework has been implemented in Java. To this end, all available PCs at the authors’ institute had to be incorporated into the MAS. Additionally, master- and slave-agents have been developed and adapted to researched fatigue simulations.

For the realization of the agent platform, the freely available, FIPA\(^2\) conformance agent system JADE (“Java Agent Development Framework”) was used. JADE is a runtime environment by which software agents can be developed and executed. Individual agent runtime environments can be connected dynamically at runtime. Additionally, JADE supports mobility features for agents and offers rudimentary safety mechanisms.

**Integration of PCs into the MAS**

A MAS implemented with JADE always consists of a so-called main container (= main agent platform) as well as an arbitrary number of further containers (= agent platform) which are connected to the main container. These further containers were proposed to be running on all PCs in the CAE-pool. For that, the service JADE-ContainerLauncherService was developed and installed on each computer (see Figure 5). Since the mechanism of background services with Java is not feasible, the freely available Java package Java-Service-Wrapper\(^3\) has been included. This package enables the installation of Java-services for Windows- and Linux-based operating systems, respectively.

---

\(^2\) Foundation for Intelligent Physical Agents, [http://www.fipa.org](http://www.fipa.org)

\(^3\) Tanuki Software: Java Service Wrapper, [http://wrapper.tanukisoftware.org](http://wrapper.tanukisoftware.org)
The JADEContainerLauncherService is incorporated in a thread which evaluates a central management file – accessible for all personal computers – in predefined time intervals. In this management file, information about the main container (computer name, port) and additional instructions for the JADE container are collected (e.g. "start", "shutdown"). Due to this centralized configuration of the MAS, a high flexibility of the required PC cluster has been achieved.

Implementation of Master- and Slave-agents

Master- and slave-agents were designed on a highly general level to enable their application, additionally to the parallel fatigue computations, for further parallel computational tasks.

Mobile slave-agents were implemented as a generic type of agents which can be executed with variable runtime parameters such as computational module, targeted workstation or master-agent ID. At the initialization of the slave-agent, the considered execution module – e.g. a binary file or a Java class – is loaded and can be easily exchanged during the runtime. The master-agent exhibits a higher complexity than the slave-agents. Central components are the graphical user interface (see Figure 6), the administration of slave-agents with the correspondent definition of execution modules and, additionally, the task management by means of the algorithm taskscheduler.

This task management differentiates between “chosen tasks” (= tasks, which are to be accomplished), “working tasks” (= tasks, which are currently executed by SAs) and “finished tasks” (= tasks, which were finished successfully). Each idle SA receives new “chosen tasks” until all tasks are executed. If the number of SAs exceeds the number of available tasks, the taskscheduler selects the SAs working on workstations.
with faster hardware resources (CPU frequency, RAM size). For the purpose of load balancing, a list of all connected workstations together with their abovementioned hardware resources and, additionally, the runtime of each task is continuously updated. The message-based communication between the master-agent and the slave-agents is based on a simple ontology (MasterSlaveOntology).

**Parameter Studies**

To investigate the runtime minimization by means of the agent-based parallelization concept, the fatigue analyses described above have been executed both in sequential and in parallel manner. Since this framework is still part of the ongoing research only some preliminary results are to be shown below.

**Sequential execution**

Multiple realizations of wind series – with varying mean values and initial displacements of the tie rod – have been generated and exported to separate files. Based on these wind data, correspondent partial damages have been computed within on single – sequentially executed – computational routine.

As an exemplary evaluation, 1226 realizations – and input files, respectively – were generated and analyzed with respect to fatigue. The parameters mentioned above were varied in the following ranges: $\bar{u} = 0.1\text{ - }10 \text{ m/s and } y_0 = 0\text{ - }10 \text{ mm}$. The correspondent fatigue results are shown in Figure 7.

![Figure 7: Partial damages for different parameter sets](image)

The sequential execution resulted in a total runtime of 389.3 min$^4$.

**Parallel execution**

Additionally, the fatigue analyses of the 1226 realizations have been executed within the parallelization framework. The number of the connected agent platforms varied from 10 to 20, depending on the activities in the computer network (used by students). Figure 8 summarizes the results obtained from different MAS configurations, mainly consisting of the same hardware configurations as for the sequential execution. However, it must be stated that the number of connected workstations customarily varies during the complete simulation. As a consequence only the mean number of attached platforms is indicated.

---

$^4$ Main hardware configuration: AMD Athlon XP+ 1600, 256 MB RAM
Based on runtimes shown in Figure 8, the resulting efficiencies for the parallel executions ranges from $E_p = 0.70$ (1 SA per workstation) to $E_p = 1.0$ (3 SAs per workstation, respectively. The results indicate a strong influence of the number slave-agents per workstation with respect to the total runtime. For example, the efficiency for 16 PCs – as a mean number – is increased from 0.70 to 0.91 by executing two simulations on one PC simultaneously. This enhancement is probably because of a more effective exploration of local hardware resources.

![Figure 8: Total runtime for different MAS configurations](image)

However, the exemplary results demonstrate a very good usability of the newly developed parallelization framework. By means of the presented Master-Slave-MAS the runtime of numerical simulations has been improved significantly.

**Conclusion**

A newly developed parallelization framework based on multiagent systems has been presented. The framework created has been evaluated and tested in the context of time-expensive fatigue analyses of structural steel members. Main components of the parallelization framework are mobile software agents which operate simultaneously on locally available distributed workstations. Computed results and the status of execution, respectively, are controlled by a single central, newly implemented master-agent that, additionally, offers a graphical user interface.

The utilization of this parallelization framework for the above named fatigue analyses leads to drastically reduced computational runtimes. Preliminary results indicate a very good scalability of the considered numerical problem.

However, the application of this framework is not only restricted to the execution of fatigue simulations. Because of the generic design of the master- and slave-agents the transferability and applicability to other technical fields of interests is granted.

**Acknowledgement**

The research work, reported here, is currently supported by funds of the German Research Foundation (DFG), within the scope of the Collaborative Research Center 398. All authors gratefully acknowledge the valuable funding provided by the DFG.
References


