Structural Engineering and Mechanics, Vol. 32, No. 1 (2009) 95-126
DOI: http://dx.doi.org/10.12989/sem.2009.32.1.095

Parallel processing in structural reliability

M.F. Pellissetti†

Institute of Engineering Mechanics, University of Innsbruck, 6020 Innsbruck, Austria

(Received August 21, 2008, Accepted March 3, 2009)

Abstract. The present contribution addresses the parallelization of advanced simulation methods for structural reliability analysis, which have recently been developed for large-scale structures with a high number of uncertain parameters. In particular, the Line Sampling method and the Subset Simulation method are considered. The proposed parallel algorithms exploit the parallelism associated with the possibility to simultaneously perform independent FE analyses. For the Line Sampling method a parallelization scheme is proposed both for the actual sampling process, and for the statistical gradient estimation method used to identify the so-called important direction of the Line Sampling scheme. Two parallelization strategies are investigated for the Subset Simulation method: the first one consists in the embarrassingly parallel advancement of distinct Markov chains; in this case the speedup is bounded by the number of chains advanced simultaneously. The second parallel Subset Simulation algorithm utilizes the concept of speculative computing. Speedup measurements in context with the FE model of a multi-story building (24,000 DOFs) show the reduction of the wall-clock time to a very viable amount (<10 minutes for Line Sampling and ≈ 1 hour for Subset Simulation). The measurements, conducted on clusters of multi-core nodes, also indicate a strong sensitivity of the parallel performance to the load level of the nodes, in terms of the number of simultaneously used cores. This performance degradation is related to memory bottlenecks during the modal analysis required during each FE analysis.

Keywords: structural reliability; stochastic structural mechanics; parallel computing; Monte Carlo simulation.

1. Introduction

Structural reliability is defined as the probability that a structure will maintain a certain performance standard during operation. In terms of the structural response, this implies that threshold levels of the stress, deformation, acceleration etc. will not be exceeded. The definition of reliability in a probabilistic framework is consistent with the generally acknowledged observation that significant random uncertainty remains associated with the parameters of numerical models (see e.g., Schuëller 2007), which are used to forecast the structural performance and which are nowadays indispensable tools in design and analysis.

While the evaluation of the reliability, or its complement, the failure probability, is conceptually straightforward - it merely involves the solution of a multi-dimensional integral -, its application to numerical models of large-scale structures is usually far from trivial, mainly because of the enormous computational efforts required. Various methods have been developed, such as the First

† E-mail: mechanik@uibk.ac.at

and Second Order Reliability Methods (FORM, SORM) (Ditlevsen and Madsen 1996) and Importance Sampling (Rubinstein 1981), as well as advanced simulation-based methods proposed more recently (Au and Beck 2001, Schuëller *et al*. 2004, Koutsourelakis *et al*. 2004). The latter have been specifically developed for analyzing the reliability of structural models which involve a very large number of uncertain parameters.

All of the above methods have in common that the computational cost of performing a reliability analysis is a multiple (often by orders of magnitude) of the computational cost of a deterministic analysis performed with a given numerical (finite element) model. This is because instead of running the analysis code only once, the reliability analysis involves its repeated execution. The large computational efforts associated with a full-scale reliability analysis have inhibited a deeper penetration in engineering practice. Significant efforts for reducing these efforts by adopting more efficient and parsimonious algorithms, as mentioned above, have been made. However, even with a greatly improved computational efficiency on the algorithmic side, the computational effort of reliability analysis remains massive, in particular in the context of high-fidelity numerical models of large-scale structures.

An additional remedy for reducing the wall-clock time, i.e., the time between the submission of a reliability analysis job and its completion, consists in the distribution of the computations on a number of distinct processors, which then work on the solution of the reliability problem simultaneously. The use of parallel processing for structural reliability analysis is nowadays a potentially viable option for most analysts in engineering practice, thanks to the affordability of computer hardware.

The efficiency of parallel processing in general depends on the degree of parallelism of the analysis task, i.e., on the relative amount of computations that can be performed in parallel. This parallelism clearly depends on the characteristics of the algorithm itself: for instance, direct MCS enjoys a very high degree of parallelism, since different samples can be computed completely independently of each other; on the other hand, algorithms which are inherently sequential, such as Markov chains, have a relatively low degree of parallelism.

Background and existing literature The computational mechanics community has been heavily using parallel processing for decades (see e.g., Sotelino 2003, Alonso *et al.* 2007). In the past, particular emphasis has been given to domain decomposition and sub-structuring (see e.g., Farhat and Lesoinne 1993), which govern the work distribution and are hence central issues of parallel structural analysis. In addition, the development of parallel solvers (see e.g., Bitzarakis *et al.* 1997, Wriggers and Boersma 1998) and parallel time stepping algorithms (see e.g., Krysl and Bittnar 2001, Farhat *et al*. 2006) has been pursued with impetus. The latter class of algorithms are relevant in structural dynamics, as is the parallel solution of eigenvalue problems (Mackay and Law 1996, Saleh and Adeli 1996).

Naturally, parallel computing is particularly suitable in situations that require repeated execution of analysis tasks, such as structural optimization (Papadrakakis *et al.* 2003, Umesha *et al*. 2005). Despite of this, quite paradoxically, parallel computing has not been emphasized so far in stochastic structural analysis and reliability analysis, although the importance of parallel computing in this field has long been recognized (see e.g., Ghanem and Kruger 1996, Schuëller (Ed.) 1997). In the field of structural dynamics, Johnson *et al.* (1997) investigated the performance of massively parallel platforms for the MCS of stochastic dynamics problems, in particular for the evolution of the transition PDF governed by the Fokker-Planck equation. This work was extended in (Johnson *et al.* 2003), which included also the topic of the parallel random eigenvalue problem, which had been

introduced in (Székely *et al.* 1998, Székely and Schuëller 2001). In the context of dynamics, the parallel MCS-synthesis of seismic ground motion has been addressed in (Shinozuka and Deodatis 1997). Considering stochastic structural mechanics in general, parallel computing has been used with the stochastic finite element method (SFEM). The parallelization of MCS-and weighted integral-based SFEM is described in (Papadrakakis and Kotsopulos 1999, Charmpis and Papadrakakis 2005), whereas in (Keese and Matthies 2005) the parallelization of the Spectral SFEM equations is addressed. Parallel computing is also instrumental for the relatively recent field of reliability based optimization(Valdebenito and Schuëller 2008), in which the reliability is to be estimated for each iteration of the outer loop, represented by the optimization process.

Purpose and outline of the present contribution In the present contribution, the use of parallel processing in the context of structural reliability analysis is investigated, with special emphasis on advanced simulation methods for the reliability estimation. More specifically, the Line Sampling method and the Subset Simulation method are parallelized. It is envisioned that by combining these efficient simulation methods with the power of parallel processing, the key objective of making structural reliability analysis of large-scale structures an affordable undertaking can be met.

Two case studies have been performed and are presented in this manuscript: the first consists in a multi-story building modeled with about 24, 000 DOF's, to which the parallel versions of both the Line Sampling and the Subset Simulation algorithm have been applied. The second case study involves a much simpler numerical model of a shear beam, with the purpose to study the parallel algorithmic efficiency of advanced reliability algorithms. In particular, the potential of the parallelization approach based on speculative computing is investigated in the context of Subset Simulation.

2. Methods of analysis

2.1 Structural reliability analysis

2.1.1 General remarks

The assessment of the reliability of structures requires a quantitative definition of failure. For this purpose it is common practice to define a so-called performance function $g(\mathbf{X})$, which characterizes the state of the structure and which is therefore a function of the vector of the uncertain parameters $\mathbf{X}$

$$g(\mathbf{X}) \quad \text{such that} \quad \begin{cases} g(\mathbf{X}) > 0 \;\Leftrightarrow\; \mathbf{X} \in S \\ g(\mathbf{X}) < 0 \;\Leftrightarrow\; \mathbf{X} \in F \\ g(\mathbf{X}) = 0 \;\Leftrightarrow\; \text{limit state} \end{cases} \tag{1}$$

where S and F denote the safe set and the failure set, respectively. The reliability of a structure can then be quantified by its complementary quantity, the probability of failure p_F

$$p_F = P[\mathbf{X} \in F] = \int_F f_{\mathbf{X}}(\mathbf{x})\,d\mathbf{x} = \int 1_F(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x})\,d\mathbf{x}$$

$$\text{where} \quad 1_F(\mathbf{x}) = \begin{cases} 0 \;\Leftrightarrow\; \mathbf{X} \in S \\ 1 \;\Leftrightarrow\; \mathbf{X} \in F \end{cases} \tag{2}$$

In the above equation, $f_{\mathbf{X}}(\mathbf{x})$ denotes the multi-dimensional probability density function (PDF) of the random vector $\mathbf{X}$. Since in most cases of interest to structural reliability it is possible to expediently represent the vector of (possibly non-Gaussian) uncertain parameters in terms of a vector of standard normal variables (zero mean, unit variance, mutually uncorrelated) by means of a suitable transformation - cf. for instance (Schuëller *et al.* 2004) - it will be assumed in the sequel without loss of generality that $\mathbf{X}$ is a vector of standard normal random variables.

While conceptually simple, the evaluation of the integral in Eq. (2) is usually a complex task, particularly when applied to large-scale structural systems. This is mainly because the evaluation of the indicator function $1_F(\mathbf{X})$ is usually time consuming, since it is necessary to perform a full FE analysis in order to determine whether a realization of the input parameters leads to failure. In the present work Monte Carlo Simulation based methods (Rubinstein 1981, Schuëller *et al.* 2004) have been adopted, which are robust and generally applicable methods for estimating p_F.

2.1.2 Monte Carlo Simulation (MCS)

The Monte Carlo estimator $\hat{p}_F$ for the probability of failure $p_F = P[F]$ has the form

$$\hat{p}_F = \frac{1}{N}\sum_{k=1}^{N} 1_F(\mathbf{X}^{(k)}), \quad \mathrm{Var}[\hat{p}_F] = \frac{1-p_F}{p_F N} \tag{3}$$

where $\mathrm{Var}[\hat{p}_F]$ is the variance of the estimator, N is the number of samples and $\mathbf{X}^{(k)}$ denotes the k-th realization of the set of input variables. The resulting coefficient of variation (C.o.V.) is a measure of the accuracy of the estimate

$$CoV_{\hat{p}_F} = \sqrt{\mathrm{Var}[\hat{p}_F]}/p_F = \sqrt{(1-p_F)/(N_{p_F})} \tag{4}$$

It should be noted that $CoV_{\hat{p}_F}$ is independent of the dimensionality of the random vector $\mathbf{X}$. From Eq. (4) it is clear that for the estimation of small failure probabilities p_F a very large number (proportional to $1/p_F$) of samples is needed for an accurate estimate, i.e., an estimate with a moderate value of $CoV_{\hat{p}_F}$. Consequently, for complex structural systems with high target reliability levels, direct MCS is usually not a viable option. For this purpose advanced MCS techniques (Au and Beck 2001, Schuëller *et al.* 2004) have been developed, which aim at reducing the variance of the estimator of p_F, with the effect that a smaller number of samples suffices. Two methods that are particularly robust with respect to dimensionality, i.e., to the number of uncertain parameters, are presented in the next sections.

2.1.3 Line sampling

The Line Sampling method is a robust sampling technique particularly suitable for high-dimensional reliability problems, in which the performance function $g(\mathbf{X})$ exhibits moderate non-linearity with respect to the uncertain parameters $\mathbf{X}$. The key step consists in the identification of a direction in the high-dimensional input parameter space, pointing to regions which strongly contribute to the overall failure probability. This important direction is denoted by $\mathbf{e}_\alpha$ in Fig. 1. The multi-dimensional space that is orthogonal to $\mathbf{e}_\alpha$ is denoted by $\mathbf{y}^\perp$.

Once an important direction $\mathbf{e}_\alpha$ has been identified, the Line Sampling algorithm starts with the generation of random samples $\mathbf{y}^{\perp(j)}$ in the space $\mathbf{y}^\perp$. To each sample j there corresponds a line $l^{(j)}$, which has the same direction as $\mathbf{e}_\alpha$. The intersection point of each line $l^{(j)}$ with the limit state ($g_y(\mathbf{y})$

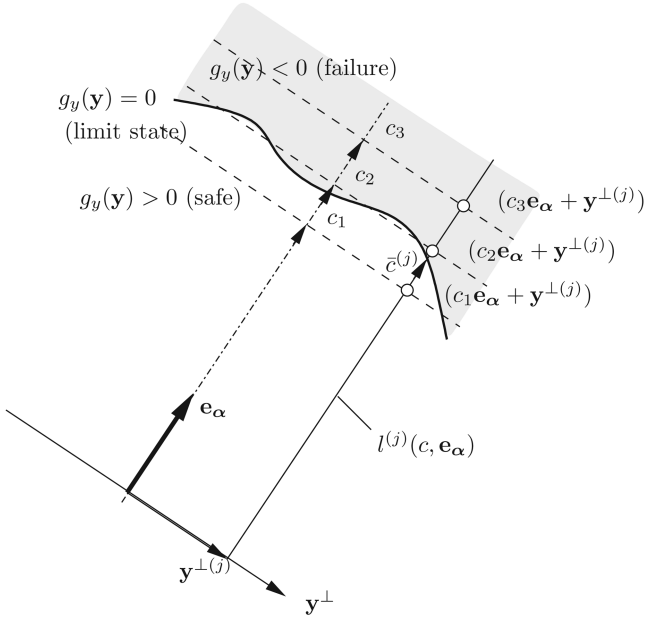

Fig. 1 Line Sampling method

= 0) can be represented as $\bar{c}^{(j)}\mathbf{e}_\alpha + \mathbf{y}^{\perp(j)}$ and hence defines the distance $c^{(j)}$ of this point from the space $\mathbf{y}^{\perp}$. Both the limit state ($g_y(\mathbf{y}) = 0$) and consequently the intersection point are not defined explicitly, but implicitly. Consequently, for each line $l^{(j)}$ the intersection point has to be found by evaluating the limit state function for several samples along that line, i.e., by computing $g_y(c_i\mathbf{e}_\alpha + \mathbf{y}^{\perp(j)})$, $i = 1, \ldots, N_c$. In most cases of practical interest, N_c is around 5.

For each sample j, the value $\bar{c}^{(j)}$ supplies a sample for the corresponding sample failure probability

$$p_F^{(j)} = \Phi(-\bar{c}^{(j)}) \tag{5}$$

where Φ denotes the Gaussian cumulative distribution function.

Repeating this procedure for a number N_L of lines, the estimator $\hat{p}_F$ of the probability of failure and the associated sample variance are then

$$\hat{p}_F = \frac{1}{N_L}\sum_{j=1}^{N_L} p_F^{(j)}, \quad \mathrm{var}[\hat{p}_F] = \frac{1}{N_L - 1}\sum_{j=1}^{N_L}(p_F^{(j)} - \hat{p}_F)^2 \tag{6}$$

With the above approach the variance of the estimator of the probability of failure $\hat{p}_F$ can be considerably reduced. Generally, a relatively low number N_L of lines have to be sampled to obtain a sufficiently accurate estimate.

Line Sampling - identification of the important direction with statistical gradient estimation The identification of a suitable important direction is critical for the efficiency of Line Sampling. For problems in which the dependence of the performance function is only moderately non-linear with respect to the parameters modeled as random variables, the direction corresponding to the gradient vector of the performance function in the underlying standard normal space leads to highly efficient Line Sampling, as demonstrated in a number of publications. For high-dimensional

problems it is possible to approximate the gradient vector by an efficient iterative procedure based on Monte Carlo sampling (Pradlwarter 2007, Pellissetti *et al.* 2007).

The main steps of the algorithm are briefly reviewed next, for more detailed information the reader is referred to the above references: An initial set of N_0 samples of the random input parameters of the model is generated with Direct MCS, however with a greatly reduced coefficient of variation; this ensures that the limit state function is still sufficiently linear with respect to the input parameters. The corresponding response samples are computed and the correlations between the input parameters and the sample deviation of the limit state function from the nominal value are estimated. For the N_{FD} input parameters with the largest absolute values of the correlation the corresponding components of the gradient are determined accurately, with a separate finite difference calculation. The contribution of these input parameters can then be removed from the sample deviation; with the updated sample deviation the iteration cycle is restarted and the next set of important parameters is extracted, based on the correlations. As the algorithm converges, i.e., the convergence measure ε approaches the target value $\varepsilon_{\min}$ (typically 0.9 or 0.95), the magnitude of the updated sample deviations will decrease steadily.

It is a characteristic of this algorithm that after a certain number of gradient components has been identified, the convergence rate decreases, because the ranking of the relative importance based on a limited number of samples deviates from the true ranking and hence the algorithm starts to determine the gradient components of spurious important parameters (so-called failures in the terminology of the algorithm). In order to speed up convergence, it is hence beneficial to enrich the initial sample pool by additional ΔN samples, whenever the accumulated number of failures m_f exceeds a user-defined threshold m_{cr}.

2.1.4 Subset simulation

The Subset Simulation method is a Markov Chain Monte Carlo (MCMC) method introduced for structural reliability problems by Au and Beck (2001). Subset Simulation overcomes the inefficiency of direct MCS in estimating small probabilities, by expressing the failure probability p_F as a product of larger, conditional probabilities. This is achieved by defining a decreasing sequence of failure events (subsets) $\{F_i\}_{i=1}^{m}$ such that $F_m = F$ and $F_1 \supset F_2 \supset \ldots \supset F_m = F$, implying that $\cap_{i=1}^{k} F_i = F_k \forall k \le m$. In Fig. 2 the subsets $F_1, \ldots, F_4$ are delimited by curved lines; the actual failure domain ("region of interest") consists in the shaded area. The total probability of failure p_F is then

$$p_F = P(F_m) = P(F_1)\prod_{i=1}^{m-1} P(F_{i+1}/F_i) \tag{7}$$

Through the appropriate definition of the intermediate failure domains, the probabilities $P(F_1)$ and $P(F_{i+1}/F_i)$, $\forall i \ge 1$ can be made sufficiently large so that their estimation can be performed with a rather small number of samples, say 100.

The algorithm of Subset Simulation (SS) starts with a set of N_1 samples generated by direct (or "plain") MCS, i.e., N_1 independent samples of the uncertain parameter vector $\mathbf{X}$ are generated. In Fig. 2 these initial samples are represented by the eight smaller dots clustered around the origin of the two-dimensional space, in the lower left portion of the figure, as well as by the two larger dots.

The boundary of the first subset F_1, indicated by the dashed line, is then defined implicitly, such that $P(F_1) \times 100\%$ of the samples fall within F_1. A suitable value for $P(F_1)$ could be 0.2, hence 20% of the initial samples would be in F_1. It should be noted that the implicit definition of the failure domain is accomplished in terms of the limit state function, cf. Eq. (1). More specifically

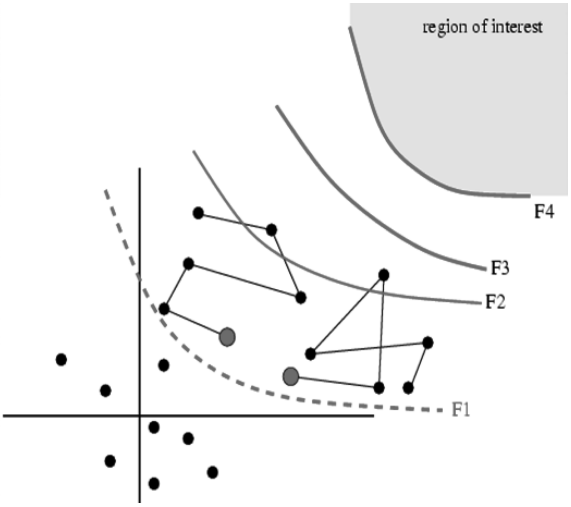

Fig. 2 Subset Simulation method

$$\mathbf{X} \in F_1 \Leftrightarrow g(\mathbf{X}) < g_1 \tag{8}$$

where $g_1 \geq 0$ is the threshold value defining the intermediate failure domain F_1. Clearly, in Fig. 2 the two larger dots correspond to those samples of the initial MCS, which belong to F_1.

From here on it is necessary to perform conditional sampling; more specifically, in order to estimate $P(F_{i+1}/F_i)$ it is necessary that the samples used for the estimate be within F_i. As described in (Au and Beck 2001), Markov Chain Monte Carlo sampling (MCMC) (Metropolis and Ulam 1949) can be used to *efficiently* obtain the conditional samples. Fig. 2 exemplifies the procedure for $i = 1$, i.e., for conditional sampling in F_1. The initial samples of the new level, associated with F_1, are represented by the larger dots. The subsequent states of the Markov chains, indicated by the dots connected by lines in Fig. 2, are obtained by generating new samples according to some probability density ("proposal density"), centered around the previous state. Before being accepted as a new actual state, it is verified whether the "candidate state" is located in F_1 by virtue of Eq. (8). If the result of this check is negative, the candidate vector is discarded and the "new" state of the Markov chain is identical to the previous one. It should be noted that in Fig. 2 only accepted candidates are shown, since all chain samples are inside F_1.

Once the pre-defined number of steps of the chains has been reached, the algorithm proceeds with the definition of yet another intermediate failure domain. For instance in Fig. 2 two samples of the two chains are seen to be in F_2. These samples are the initial states of the chains of the next level, i.e., the one associated with F_2. This procedure is repeated until the intermediate failure domains have converged to the actual failure domain, i.e., until $g_i \leq 0$.

2.2 Parallel reliability analysis

2.2.1 Parallelism in simulation-based stochastic structural mechanics

In stochastic structural mechanics there are mainly two levels of parallelism, a high level, coarse-grain parallelism associated with the stochastic algorithms and a low level parallelism related to the deterministic problem.

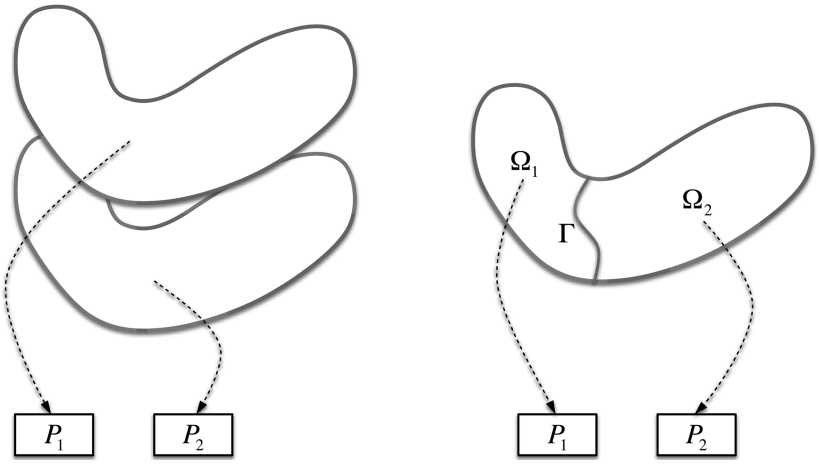

Fig. 3 Task decomposition and mapping; left: sample set decomposition, right: domain decomposition

High level parallelism This level of parallelism is associated with the stochastic analysis method used to propagate the uncertainty to the response quantities of interest. In the case of Monte Carlo simulation-based algorithms, the computationally expensive part consists in the repeated execution of computations with system matrices which differ in some way from the nominal ones. With these algorithms, a significant number of repeated executions can be carried out independently from each other, before synchronization is required.

The high level parallelism is shown conceptually in the left portion of Fig. 3, where two different realizations of the entire structural domain are mapped to two different processing units, P_1 and P_2.

Low level parallelism of the deterministic analysis A second, *lower level of parallelism* is associated with the deterministic problem: in order to exploit the parallelism at this level, the most time consuming computational tasks required for a single solution of the deterministic problem are re-defined in a way that permits a parallel execution for as great a portion of the overall work as possible. This may be done by performing the iterative solution of systems of linear equations in parallel. In context with FE models domain decomposition methods (Farhat and Lesoinne 1993) can be applied for this purpose. The right portion of Fig. 3 sketches the basic idea of domain decomposition based parallelization, where the physical domain is divided into subdomains Ω_1 and Ω_2, which are separated by the boundary Γ.

The goal of parallel efficiency is typically more difficult to accomplish at this lower level of parallelism, since two of the main factors, load balancing and communication, are more difficult to control, compared to the high level parallelism inherent in most stochastic algorithms. However, the potential benefits of parallelizing the deterministic analysis increase with the size of the deterministic problem. Therefore, the question of which level of parallelism to exploit first depends strongly on the size of the underlying FE model. The present paper does not provide answers to this question; however, the importance of dealing with this aspect in future research is emphasized.

2.2.2 Parallel line sampling

Parallelization of statistical gradient estimation The statistical gradient estimation algorithm for approximating the gradient of the performance function in the space of the uncertain parameters can provide the important direction for Line Sampling and is amenable to parallel processing. In

particular, the time-consuming steps of the algorithm, namely i.) those in which MC samples are generated, and ii.) those in which individual gradient components are approximated by finite difference, may be parallelized. In the present study, the parallelization is implemented following the *master-slave paradigm*. A continuous numbering of the algorithmic steps has been introduced along the right margin in the presentation of all parallel algorithms of this manuscript.

Algorithm 1: Parallel statistical gradient estimation algorithm

Master initializes algorithm parameters ($\epsilon = 0\,,\ m_f = 0$)	*1*
Master generates input data of N_0 initial Monte Carlo (MC) samples	*2*
for all i=1:N_0 **parallel do**	
Master sends input data of MC sample i to worker	*3*
Worker performs **independent** FE analysis for MC sample i	*4*
Worker returns output data of sample i to master	*5*
barrier	
Master evaluates ϵ	*6*
while $\epsilon < \epsilon_{\min}$	
if $m_f \geq m_{\mathrm{cr}}$ **then**	
Master generates input data of ΔN new MC samples (upgrade sample pool)	*7*
for all i=1:ΔN **parallel do**	
Master sends input data of MC sample i to worker	*8*
Worker performs **independent** FE analysis for MC sample i	*9*
Worker returns output data of sample i to master	*10*
barrier	
Master ranks RV's	*11*
Master creates N_{FD} finite difference (FD) input data sets $\Rightarrow$ gradient components	*12*
for all i=1:N_{FD} **parallel do**	
Master sends FD input data set i to worker	*13*
Worker performs **independent** FE analysis for FD input data set i	*14*
Worker returns output data of sample i to master	*15*
barrier	
Master increments m_f by no. of less important components ("failures") $\Rightarrow m_f$	*16*
Master updates convergence measure ϵ	*17*
end	

In this algorithm, as highlighted in the above pseudo code, the FE analysis performed by the workers for each MC sample and for each input data set -corresponding to one gradient component - is *independent* and requires no communication between the worker and the master or among the workers. The associated portions of the algorithm, i.e., the FE analyses, account *almost entirely* for the execution time of the algorithm. In comparison, the generation and the transfer of the input data samples and the output data requires a negligible amount of time.

Two aspects related to the parallel loops in Algorithm 1 (steps 3 through 5; 8 through 10; 13 through 15) are noteworthy and apply also to all the other algorithms presented in the sequel (i.e., Algorithms 2 through 5):

1. each worker corresponds to a hardware resource; in the case studies presented here each core represents a worker.
2. The number of iterations executed in the parallel for-loops may exceed the number of workers. In this case, at least some workers process more than one iteration.
3. Each worker processes one iteration at a time. The higher granularity that would be achieved by batching several iterations is not essential in structural reliability analysis with even the most basic FE models, since the overhead associated with the dispatch of each batch of iterations is negligible compared to the computational cost of the FE analysis. Hence, the reduction of the overall overhead via reduction of the number of batches (and the consequent increase in the granularity) is not justified, especially since with higher granularity the efficiency of the dynamic load balancing inherent in the master-slave paradigm deteriorates.

Besides the input data sets sent by the master to the workers, there is no additional need for shared *variable* data between master and workers. Furthermore, workers do not need to share any *variable* data among themselves. Typically, there is a substantial amount of *invariable* data that all workers need, in particular related to the definition of the FE model.

It should be noted that in the sequential version the parameter ΔN takes on rather moderate values, of the order of 20-30, and and even more so N_{FD}, especially for a low specified value of m_{cr}. For the above parallel version of the algorithm, however, it is advantageous to tune the parameters N_0, ΔN and N_{FD} of the statistical gradient estimation algorithm such that each available worker evaluates at least one sample. This is the case if these parameters are greater than or equal to the number of workers *nw*

$$N_0 \geq nw$$
$$\Delta N \geq nw$$
$$N_{FD} \geq nw$$

Line-wise parallelization The Line Sampling method, previously described and illustrated in Fig. 1, has significant inherent parallelism, since the individual lines, along which samples of the limit state function are evaluated, can be processed independently. The consideration of this fact leads to an embarrassingly parallel Line Sampling algorithm with the coarsest possible partition; with reference to Eq. (6), each of the sample failure probabilities, $p_F^{(j)}$, associated with a given batch of N_{BL} lines, is evaluated in parallel.

Algorithm 2: Line Sampling using line-wise parallelization

Master initializes important direction (e.g. based on gradient estimation), $CoV_{\hat{p}_F}$	*1*
while $CoV_{\hat{p}_F} = \sigma_{\hat{p}_F}/\hat{p}_F \geq CoV_{\text{tol}}$	
Master generates N_{BL} line seeds in input parameter space (vectors $\mathbf{y}^{\perp(j)}$ in Fig. 1)	*2*
for all j=1:N_{BL} **parallel do**	
Master sends line seed of line j to worker	*3*
Worker evaluates **independent** sample failure probability $p_F^{(j)}$	*4*
Worker returns sample failure probability $p_F^{(j)}$ to master	*5*
barrier	
Master updates estimates $\hat{p}_F$ and $CoV_{\hat{p}_F}$ based on Eq. (6)	*6*
end	

For a given target accuracy, let the total number of lines required to achieve convergence of the estimator in Eq. (6) be denoted by N_L. Then the degree-of-parallelism (DOP) and the speedup of the above *line-wise parallelization* can be quantified as follows

$$\text{DOP} = N_L \quad \Rightarrow \quad S(nw) \leq N_L \tag{9}$$

The bound on $S(nw)$ expresses that once the number of workers nw reaches the number of lines N_L, no additional speedup can be materialized by adding more workers. Clearly, this affects the efficiency, which experiences a steady deterioration for increasing nw

$$E(nw) \leq N_L/nw, \quad \text{for} \quad nw \geq N_L \tag{10}$$

It should be noted that in view of the fact that each worker corresponds to one hardware unit, these expressions may be viewed as relating to hardware resources.

Sample-wise parallelization A finer partition -and hence a higher degree-of-parallelism can be achieved by considering that in the application to real engineering problems the sample failure probability $p_F^{(j)}$ associated with each line in Fig. 1 is approximated by evaluating the performance function on a grid of points, indicated by circled dots in the figure, and subsequent interpolation of the performance function between the evaluated points.

Denoting the number of grid points of each line by N_c, for a given batch of N_{BL} lines, a total of $N_c \cdot N_{BL}$ samples are to be evaluated for each batch, a task that is clearly embarrassingly parallel.

Algorithm 3: Line Sampling using sample-wise parallelization

Master initializes important direction (e.g. based on gradient estimation), $CoV_{\hat{p}_F}$	*1*
while $CoV_{\hat{p}_F} = \sigma_{\hat{p}_F}/\hat{p}_F \geq CoV_{\text{tol}}$	
Master generates N_{BL} line seeds in input parameter space (vectors $\mathbf{y}^{\perp(j)}$ in Fig. 1)	*2*
for j=1:N_{BL} **do**	
Master generates input data sets corresponding to N_c grid points (vectors $c_k\mathbf{e}_\alpha + \mathbf{y}^{\perp(j)}$, $k = 1, 2, 3$ in Fig. 1)	*3*
for all i=1:N_cN_{BL} **parallel do**	
Master sends input data of sample i to worker	*4*
Worker performs **independent** FE analysis for sample i	*5*
Worker returns output data of sample i to master	*6*
barrier	
for j=1:N_{BL} **do**	
Master recovers sample failure probabilities $p_F^{(j)}$	*7*
Master updates estimates $\hat{p}_F$ and $CoV_{\hat{p}_F}$ based on Eq. (6)	*8*
end	

Compared to the line-wise parallel Line Sampling, cf. Eqs. (9) and 10, the above sample-wise parallelization exhibits clearly a higher degree-of-parallelism and hence a less restrictive bound on the speedup and the efficiency,

$$\text{DOP} = N_c \cdot N_L \Rightarrow \; S(nw) \leq N_c \cdot N_L \;\Rightarrow\; E(nw) \leq N_c \cdot N_L / nw, \quad \text{for} \quad nw \geq N_c \cdot N_L \tag{11}$$

As will be exemplified in section 3.3, this advantage of algorithm 3 over algorithm 2 is particuarly relevant for problems in which line sampling performs very well, in which case the number of lines N_L required for convergence can be quite low and hence the bound on the parallelism of algorithm 2 becomes particularly severe.

2.2.3 Subset simulation

Chain-wise parallelization As mentioned previously, the Subset Simulation method consists in advancing Markov chains with the objective to approach the failure domain gradually.

The parallelism of Subset Simulation stems from the fact that at each level a number of Markov chains are advanced concurrently and independently. This is indicated in Fig. 2, where the two chains are clearly independent of each other. The mutual independence of the Markov chains at a given level gives rise to the most straightforward approach for parallelizing the Subset Simulation method, in which the individual chains are mapped to different processors.

Algorithm 4: Subset Simulation using parallel chain advancement

Master initializes algorithm parameters	*1*
Master generates input data of N_0 initial Monte Carlo (MC) samples	*2*
for all j=1:N_0 **parallel do**	
Master sends input data of MC sample j to worker	*3*
Worker performs **independent** FE analysis for MC sample j	*4*
Worker returns output data of sample j to master	*5*
barrier	
Master extracts N_C samples closest to failure, sets intermediate failure threshold g_i	*6*
while $g_i > 0$	
Master initializes Markov chains w/ samples inside failure domain of previous level	*7*
for k=2:N_K (steps in the Markov chains) **do**	
for j=1:N_C (individual Markov chains) **do**	
Master generates input data set for one new candidate state of chain j	*8*
for all j=1:N_C **parallel do**	
Master sends input data of candidate state of chain j to worker	*9*
Worker performs **independent** FE analysis	*10*
Worker returns corresponding output data to master	*11*
barrier	
for j=1:N_C **do**	
Master determines if candidate state of chain j is accepted	*12*
Master extracts N_C samples closest to failure, set intermediate failure threshold g_i	*13*
end	

In the above algorithm, the number of concurrent chains is equal to $N_C = F_i N_0$, where F_i is the intermediate failure probability. The degree-of-parallelism of this *chain-wise parallelization* is equal to the number of chains, which then bounds the speedup that can be achieved

$$\text{DOP} = N_C \quad \Rightarrow \quad S(nw) \leq N_C \tag{12}$$

From the above it follows that once the number of workers nw reaches the number of chains N_C, additional workers will be useless, as the wall-clock time and hence the speedup will stagnate. This lacking benefit of using more workers results in a steadily decreasing efficienc

$$E(nw) \leq N_C/nw, \quad \text{for} \quad nw \geq N_C \tag{13}$$

It should be noted that the above equations exclusively address the parallelism of the MCMC part. Including the initial direct MCS in the analysis of the parallelism, leads to a slightly more advantageous situation, since direct MCS has a higher degree of parallelism, namely equal to the number of initial samples N_0, which is usually between 10^2 and 10^3.

Parallelization by speculative computing As discussed in the previous section, the parallelism of Subset Simulation is exhausted once the number of workers reaches the number of Markov chains advanced at a given level. In the present section an approach is discussed, through which the wall-clock time of the Subset Simulation algorithm can be further reduced.

The presented approach consists in evaluating the limit state function of potential future states of the Markov chains in advance. This type of approach is referred to as "speculative computing" in the literature (cf. e.g., Leite and Topping 1999).

In order to apply speculative computing to Subset Simulation it is necessary to construct a so-called speculative tree, the root of which corresponds to the current state of the chain. Fig. 4 schematically shows a speculative tree of degree three, i.e., including all the possible states of the Markov chain after three future steps. The current state of the chain is indicated by the cross ("Step 0"). The potential states after one step are indicated in the right portion of the figure by the small circles ("Step 1"). While one of the potential states corresponds to a new state (in Fig. 4 this new state is located to the lower right of the current state), the other potential state after one step is identical to the current state. This case materializes if the candidate state is rejected, i.e., if it lies outside the subset corresponding to the current level.

Based on the above, the following parallel algorithm for Subset Simulation can be formulated, where N_{Steps} is the number of steps simultaneously advanced with the speculative tree and N_{off} is the total number of off-springs, i.e., potential *new* states, for each chain. For these offsprings the performance function needs to be evaluated.

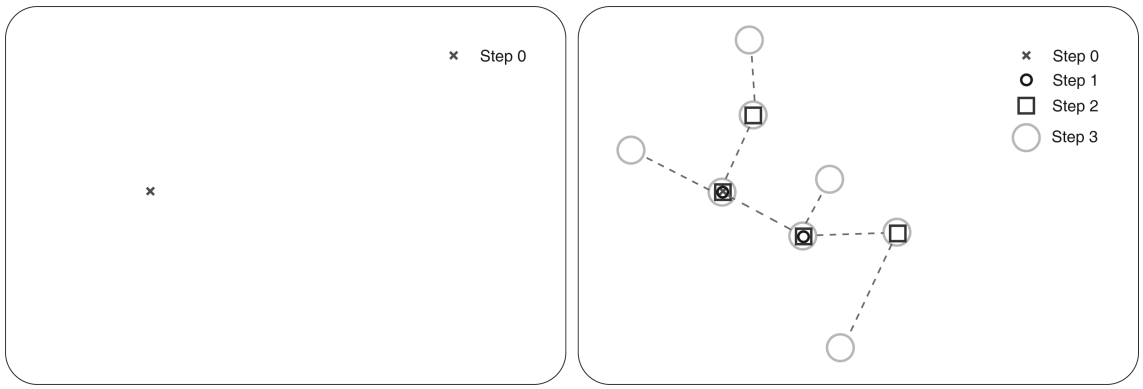

Fig. 4 Schematic sketch for speculative tree associated with a given state ("Step 0") in the Markov chain

Algorithm 5: Subset Simulation using multi-step parallel chain advancement

Master initializes algorithm parameters	*1*
Master generates input data of N_0 initial Monte Carlo (MC) samples	*2*
for all j=1:N_0 **parallel do**	
Master sends input data of MC sample j to worker	*3*
Worker performs **independent** FE analysis for MC sample j	*4*
Worker returns output data of sample j to master	*5*
barrier	
Master extracts N_C samples closest to failure, sets intermediate failure threshold g_i	*6*
while $g_i > 0$	
Master initializes Markov chains w/ samples inside failure domain of previous level	*7*
Master initializes $k = 2$	*8*
while $k < N_K$ (steps in the Markov chains) **do**	
for j=1:N_C (individual Markov chains) **do**	
Master generates input data sets for speculative tree (N_{Steps}) of chain j	*9*
for all j=1:$N_C N_{\text{off}}$ **parallel do**	
Master sends input data of candidate state j to worker	*10*
Worker performs **independent** FE analysis	*11*
Worker returns corresponding output data to master	*12*
barrier	
for j=1:N_C **do**	
Master identifies realized branch of speculative tree	*13*
Master advances chain j by N_{Steps} steps	*14*
Master increments k by N_{Steps}	*15*
Master extracts N_C samples closest to failure, set intermediate failure threshold g_i	*16*
end	

In order to analyze the potential reduction of the wall-clock-time afforded by this approach, it is necessary to quantify the total number of off-springs N_{off} in the speculative tree, for a given S_{teps}, i.e., a given number of steps to be advanced at once. For a single chain the following holds

$$N_{\text{off}} = \sum_{k=1}^{N_{\text{Steps}}} N_{\text{off}}^{(k)} = \sum_{k=1}^{N_{\text{Steps}}} 2^{k-1} = 2^{N_{\text{Steps}}} - 1 \tag{14}$$

Clearly, in a parallel program with nw workers, the number of samples that can be processed in parallel at once, $N^{(1)}$, is bounded by nw, i.e., $N^{(1)} \leq nw$. Since N_C chains are advanced concurrently, the number of simultaneous samples has to be distributed among the N_C chains and the maximum number of off-springs that can be computed simultaneously for each Markov chain is then

$$N_{\text{off}}^{(1)} = N^{(1)}/N_C \leq nw/N_C \tag{15}$$

In view of Eq. (14)

$$N_{\text{Steps}}^{(1)} \leq \log_2\left(\frac{nw}{N_C} + 1\right) \tag{16}$$

If only speculative trees of full degree are considered, the number of steps that the Markov chain can be advanced at once is obtained by rounding down the right-hand side of the previous inequality

$$N_{\text{Steps}}^{(1)} = \begin{cases} \text{floor}\left[\log_2\left(\frac{nw}{N_C}+1\right)\right] & \text{if} \quad nw \geq N_C \\ 1 & \text{otherwise} \end{cases} \tag{17}$$

Assuming that $nw > N_C$, the wall-clock time for subset simulation on nw workers, using speculative trees is then as follows,

$$T(nw) = \text{ceil}\left[\frac{N_1 \cdot T(1)}{nw}\right] + N_L \cdot \text{ceil}\left[\frac{N_K}{N_{\text{Steps}}^{(1)}}\right] \cdot T(1) \tag{18}$$

where N_1 is the number of samples of the initial plain MCS step, $T(1)$ is the sequential execution time of a single evaluation of the performance function, N_L is the number of intermediate failure domains in the Subset Simulation algorithm and N_K is the number of states of a single chain.

Clearly, speculative computing is merely concerned with reducing the wall-clock time. The efficiency $E(nw)$ of the associated parallel algorithm will be rather poor, in particular as the degree of the speculative tree increases, because the number of discarded samples rapidly increases.

2.2.4 Lower level parallelisms in line sampling and subset simulation

The parallel algorithms presented in the previous two subsections have in common that exclusively the high level parallelism, associated with the *complete independence* of individual FE analyses, is exploited. With reference to section 2.2.1 it is noted that lower levels of parallelism do exist. The specifics of the lower level parallelisms clearly depend on the type of FE analysis, such as linear static analysis, non-linear quasi-static analysis, modal analysis (eigenvalue extraction), transient linear dynamic analysis, non-linear dynamic analysis, etc.

While the use of this lower level parallelism is not within the scope of this work, it can play a role in further reducing the wall-clock time of both Line Sampling and Subset Simulation. This is because for both algorithms there is a limit on the parallelism associated with the independence of individual FE analyses, since the algorithms involve synchronization steps, such as the evaluation of convergence parameters.

As the case study presented next demonstrates, the number of samples that can be evaluated simultaneously is frequently in the range of 30 to 50 or even below. Since the number of workers available in modern computing infrastructure is often well above that number, it is worthwhile to pursue further improvement of the wall-clock time of Line Sampling and Subset Simulation by proceeding to the next lower level of parallelism. More specific options in this regard are discussed in section 3.5, based on observations related to the first case study.

3. Case study - Multi-story building modeled with finite elements

3.1 General remarks

The present section addresses the reliability analysis of a multi-story building with uncertain

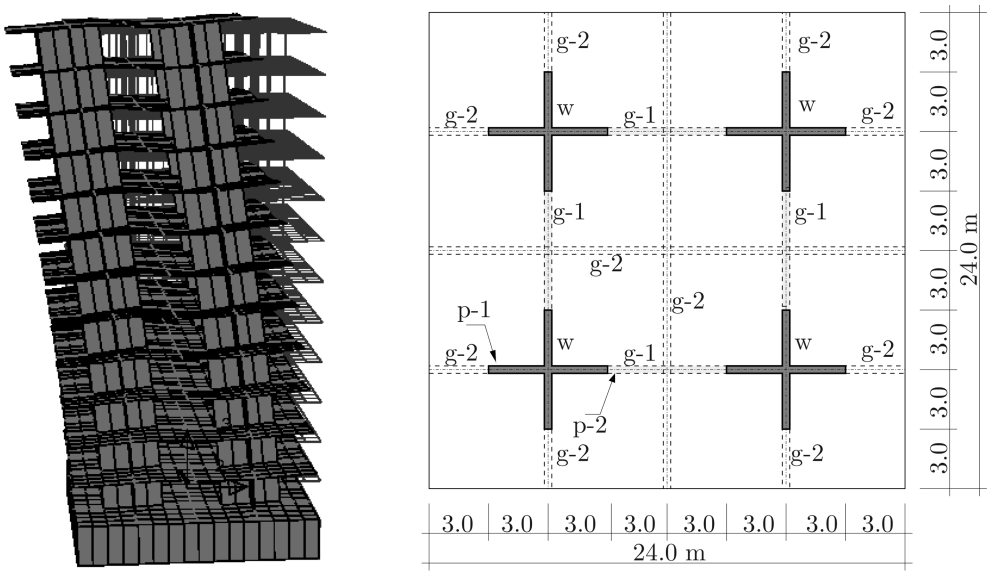

Fig. 5 FE model of multi-story building

loading and uncertain structural properties, recently investigated by (Pradlwarter and Schuëller 2008, submitted). The building is depicted in Fig. 5; more specifically, the left portion shows the first eigenmode of the structure, while on the right the floor plan is shown. The finite element model of the structure has been constructed and solved with the software code FEAP (Zienkiewicz and Taylor 2000); the model consists of 4,046 nodes, 5,972 elements, for a total of 24,276 DOFs.

In this example uncertainties both in the structural parameters and in the loading are accounted for. The uncertain structural parameters are denoted by Θ and the uncertain loading parameters are denoted by Ξ; their respective probability density functions are denoted by $p_{\Theta}(\theta)$ and $q_{\Xi}(\xi)$. It is convenient to approach the reliability problem involving these uncertainties with a two-level strategy, accounting at a first level for the uncertainties in the loading and at a second level for the uncertainties of the structural parameters.

The analysis performed in this work is restricted to the linear domain, hence the equation of motion resulting from the FE discretization has the form

$$\mathbf{M}(\Theta)\ddot{\mathbf{u}}(t;\Theta,\Xi)+\mathbf{C}(\Theta)\dot{\mathbf{u}}(t;\Theta,\Xi)+\mathbf{K}(\Theta)\mathbf{u}(t;\Theta,\Xi)=\mathbf{f}(t;\Xi) \tag{19}$$

where the dependence of the transient response on the uncertain structural and loading parameters is made explicit by the notation $\mathbf{u}(t;\Theta,\Xi)$.

Based on the investigations by (Pradlwarter and Schuëller 2008, submitted), who identified the curvature at the position p-2 of the girder g-1 (cf. right portion of Fig. 5) of floor 5 as the most critical response quantity, the limit state function (cf. Eq. (1)) is defined as follows

$$\begin{aligned} g_{\theta}(\Xi)\geq 0 \quad &\Leftrightarrow \quad \kappa_{\theta}(\Xi)\in[-0.004,0.004] \\ g_{\theta}(\Xi)<0 \quad &\Leftrightarrow \quad \kappa_{\theta}(\Xi)\notin[-0.004,0.004] \end{aligned} \tag{20}$$

where $\kappa_{\theta}(\Xi)$ is the total curvature (sum of curvature due to static load and ground acceleration) at

position p-2 of girder g-1 of floor 5. The subscript θ of κ and g indicates the dependence on the value of θ.

For a given value θ of the uncertain structural parameters Θ the conditional failure probability has the form,

$$p_F(\theta) = \int 1_F(\theta,\xi) q_\Xi(\xi) d\xi \tag{21}$$

where $1_F(\theta,\xi)$ is evaluated according to Eqs. (2) and (20). For the analysis pursued in the sequel it is convenient to utilize the β-value corresponding to $p_F(\theta)$

$$\beta(\theta) = -\Phi^{-1}(p_F(\theta)) \tag{22}$$

where $\Phi^{-1}(\cdot)$ denotes the inverse standard Gaussian CDF. The calculation of the above conditional p_F and of the corresponding β for a specific value of the structural parameters θ can be performed quite efficiently by evaluating the Karhunen-Loève expansion of the response, as detailed in (Pradlwarter and Schuëller 2008, submitted). The computationally most demanding step of this procedure is the solution of the *generalized eigenvalue problem* associated with the structural model and its parameters θ.

Since the structural parameters corresponding to θ are random, the conditional p_F is a random variable. The second-level reliability problem, now involving the uncertainties in the structural parameters is then formulated by defining the following limit state function,

$$\begin{aligned} &g(\Theta) \geq 0 \;\Leftrightarrow\; p_F(\Theta) \leq 2.275\% \;\Leftrightarrow\; \beta(\Theta) \geq 2 \\ &g(\Theta) < 0 \quad \text{otherwise} \end{aligned} \tag{23}$$

In view of this definition the total p_F obtained by evaluating the integral

$$p_F = \int 1_F(\theta) p_\Theta(\theta) d\theta \tag{24}$$

corresponds to the probability that for a random sample of the structural parameters the conditional failure probability due to uncertain loading exceeds 2.275%. In the present manuscript the estimation of the above integral is performed using Parallel Line Sampling (section 3.3) and Parallel Subset Simulation (section 3.4).

Computational aspects of the limit state function evaluation The application of the advanced reliability methods described in the following sections requires the repeated evaluation of the limit state function defined in Eq. (23). Each of these evaluations involves the following analysis steps (in the parenthesis at the end the average execution time on the Xeon cluster - introduced in section 3.3 - is given):

1. *Assemby of the FE model* with the parameters θ and export of mass and stiffness matrix to file; this step is executed using the finite element program FEAP, around which a Perl script is wrapped, in order to first modify the input file records corresponding to the uncertain structural parameters and to trigger the FEAP run. (*4 seconds*)
2. *Eigenvalue and -vector extraction* and construction of the Karhunen-Loève vectors of the response. The eigensolution is the most time consuming task and is performed using the function eigs of Matlab, which utilizes the iterative solver Arpack (Lehoucq *et al.* 1998). Since

the parallel versions of the advanced MCS algorithms are implemented in C, the Matlab function performing this analysis step is compiled into a C shared library utilizing the Matlab compiler mcc, against which the C program handling the parallel sample evaluation (cf. the following section 3.2) is linked at run-time. (*18 seconds*)

3. *Evaluation of the first excursion probability.* Again, this step is implemented in a Matlab function which is in turn compiled into a C shared library. (*2 seconds*)

For a detailed explanation and the formulation of the above analysis steps it is referred to (Pradlwarter and Schuëller 2008, submitted).

3.2 Computing environment and implementation of the parallel algorithm

Hardware The parallel reliability analysis of the multi-story building has been conducted using two different compute clusters. The first one (hereafter referred to as *Xeon cluster*) consists of rack server nodes with two Intel Xeon quad-core processors (2.33 GHz) per node, whereas the second one (hereafter referred to as *Opteron cluster*) consists of rack server nodes with four AMD Opteron dual-core processors (2.2 GHz) per node. Consequently, in both clusters each of the used nodes has a total of eight available cores. As for the memory, the Xeon cluster has 6 GByte SDRAM per node, whereas the Opteron cluster nodes have access to 16 GByte ECC RAM each. Finally, with respect to the cache size, the Xeon cluster features 2 MB L2-Cache per core, versus 1 MB of the Opteron cluster.

Software The adopted parallel algorithms have been implemented utilizing Matlab for the high-level control tasks of the advanced MCS algorithms, such as the generation of the input parameter samples and the processing of the output of the limit state function evaluations. The Matlab-based control program requests the evaluation of batches of samples from a C program called taskpooldriver, which performs this evaluation in parallel with automatic load balancing achieved by using the *master-slave paradigm*: the master hands out samples to the slaves one-at-a-time and upon completion of a sample evaluation the corresponding slave obtains a new sample. This ensures that faster slaves process more samples and consequently the cumulative idle time of all the slaves is minimized. This master-slave paradigm has been implemented using the MPICH implementation of the MPI library.

On both utilized clusters the execution of taskpooldriver is handled by the Grid Engine, an open source job management system sponsored by Sun Microsystems. The implementation avoids delays related to the overheads associated with the job management system, such as queuing time or library initialization. This is done by designing taskpooldriver so as to live for the entire duration of the parallel reliability analysis: this way it is ready to distribute samples for parallel evaluation immediately, whenever the algorithm requires it (parallel for loops in Algorithms 1 through 5). In view of this, and considering that the timing started after start-up of taskpooldriver and ended prior to its completion, the overheads introduced by the job management system can be neglected.

The evaluation of the limit state function conducted by the slaves for each sample, consists in the three analysis steps described in the last paragraph of the previous section. The Perl script driving the first step is triggered using the function system available in C, whereas the compiled Matlab functions implementing steps 2 and 3 are called directly from inside the C code.

3.3 Parallel line sampling

Convergence behavior of the sequential algorithm The analysis presented in this section has been conducted implementing the algorithm described in section 2.1.3, i.e., first the efficient gradient estimation method has been utilized to identify the most important parameters, which leads to the important direction required by the Line Sampling method, and then Line Sampling is performed until convergence is achieved with respect to the specified CoV of the estimator. The summary of the algorithm parameters and the convergence history of the algorithm in the sequential mode are reported in Table 1.

Fig. 6 depicts the performance of the Line Sampling method if the above parameters are used. In the left portion the value of the performance function -in this case the conditional β - is plotted for different lines. Selecting a value of $\beta = 2$ as the limit state - as indicated in the left part of Fig. 6 - leads to an ensemble of intersection points, the c-coordinate of which is utilized to compute the Line Sampling estimate of p_F, based on Eqs. (5) and (6). In the right portion of Fig. 6 estimates of p_f are shown for different values of the limit state of β, denoted by $\beta_{\text{cond,LS}}$. The error bars superimposed on the estimates delimit the interval obtained by offsetting the estimate by the square root of its variance. The value of p_F corresponding to the limit state indicated by the dashed line in the right portion of the figure is given by the bar at $\beta_{\text{cond,LS}} = 2$. The CoV corresponding to this

Table 1 Line Sampling - Algorithm parameters and convergence history

Algorithm parameters		
Gradient estimation		
Initial Monte Carlo sample pool, N_0		32
Number of gradient components per stage, N_{FD}		8
Number of failures per stage triggering upgrade, m_{cr}		2
Size of Monte Carlo sample pool upgrade, ΔN		8
Line Sampling		
Number of support points per line, N_c		4
Convergence criteria		
Gradient estimation, $\varepsilon_{\min}$		> 90%
Line Sampling, CoV (p_F)		< 30%
Convergence history	**No. FE runs**	**Convergence measure in %**
Gradient estimation		$\varepsilon_{\min}$
Initial Monte Carlo sampling, N_0	32	
Gradient component evaluation - stage 1, $N_{FD}^{(1)}$	8	84.6
Monte Carlo sample pool upgrade, ΔN	8	
Gradient component evaluation - stage 2, $N_{FD}^{(2)}$	8	93.9
Line Sampling		CoV (p_F)
Evaluation of limit state function at line support points	13 × 4	0, 32, 59, 42, 44, 38, 33, 34, 35, 36, 35, 31, 27
Total number of FE runs	108	

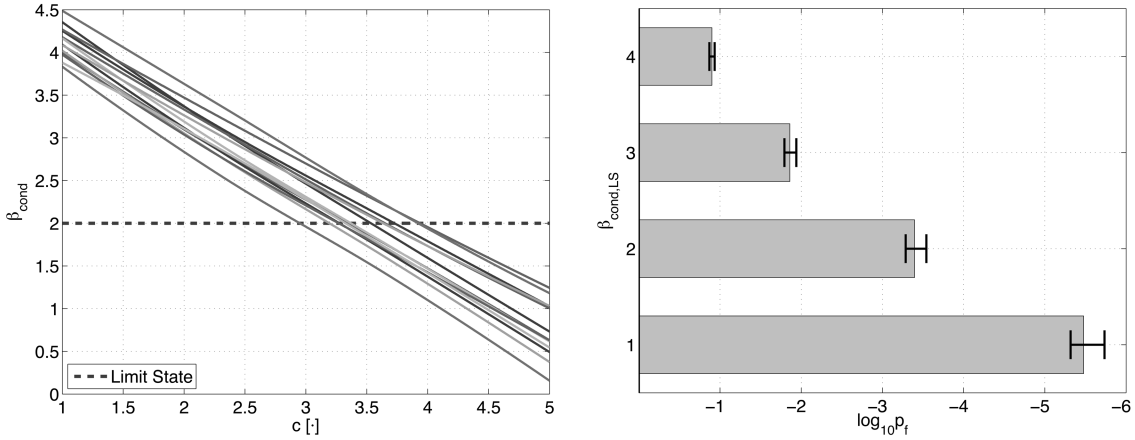

Fig. 6 Line Sampling based reliability analysis of multi-story building

estimate, defined as the ratio of the square root of the variance over the estimate itself, amounts to 28.5% and hence satisfies the stopping criterion adopted in this case, namely that the CoV of the estimate be below 30%.

Ideal speedup On the basis of the convergence history of the considered problem, listed in Table 1, the *ideal* parallel performance of Line Sampling is analyzed. The performance of parallel gradient estimation (Algorithm 1 in section 2.2.2) and that of Line Sampling (Algorithms 2 and 3) is considered separately.

This analysis assumes a constant execution time of each FE analysis and neglects overheads, such as the time required for transferring the input data sets from the master to the workers. The first assumption, i.e., constant execution time, represents a stronger deviation from the actual behavior, since the input data set transfer time is indeed negligible.

Under these circumstances and for the convergence history of this example the ideal execution time of Algorithms 1-3 is given as

$$T_1(nw) = \mathrm{ceil}\left(\frac{32}{nw}\right) + 2 \cdot \mathrm{ceil}\left(\frac{8}{nw}\right) + \mathrm{ceil}\left(\frac{8}{nw}\right)$$

$$T_2(nw) = \mathrm{ceil}\left(\frac{13}{nw}\right)$$

$$T_3(nw) = \mathrm{ceil}\left(\frac{13 \cdot 4}{nw}\right)$$

Figs. 7 and 8 show the corresponding ideal speedup for different numbers of workers.

The speedup of parallel gradient estimation is linear for $nw \leq 8$. This corresponds to the minimum size of sample batches used by the algorithm, namely the number of gradient components evaluated per stage N_{FD} and the number of Monte Carlo samples used for a sample pool upgrade ΔN. After nw reaches the maximum size of sample batches arising in the algorithm, i.e. the sample size N_0 of the initial MC sampling, the speedup remains constant, as the execution time cannot be further reduced (without tackling lower level parallelism).

A qualitatively similar behavior applies to the Line Sampling, where the speedup is close to linear

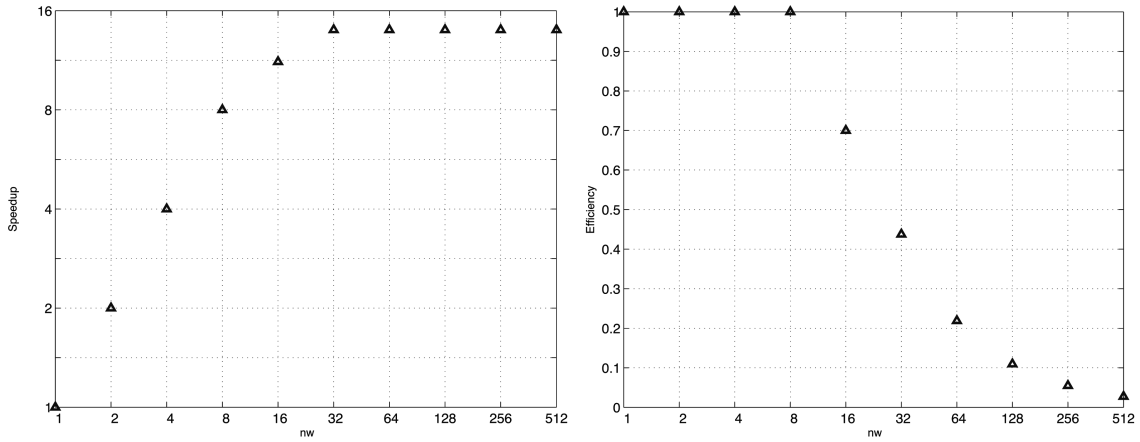

Fig. 7 Ideal performance of parallel gradient estimation for multi-story building

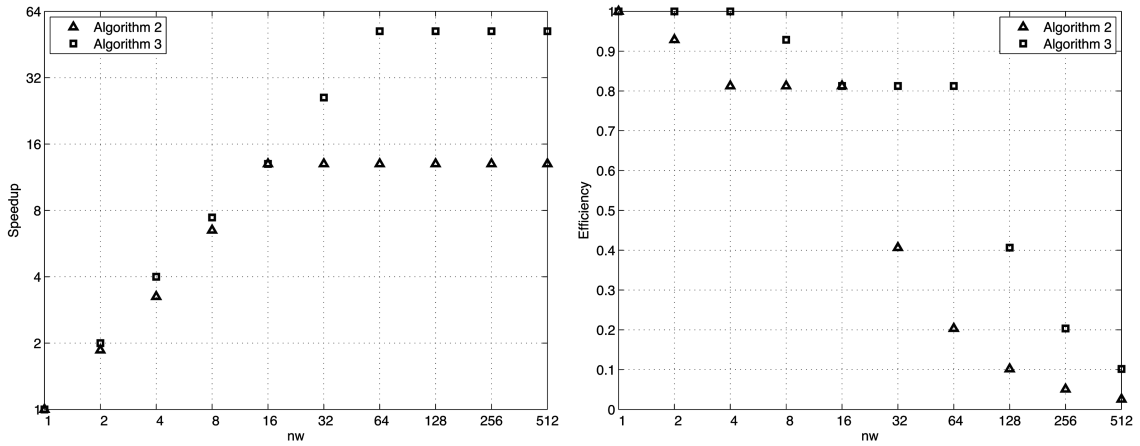

Fig. 8 Ideal performance of parallel Line Sampling for multi-story building

for $nw \leq 16$, in the case of Algorithm 2, and for $nw \leq 64$, in case of Algorithm 3.

Clearly, the latter algorithm can utilize effectively a higher number of workers due to its finer granularity.

While these results apply quantitatively only for this case study, qualitatively the above statements carry over to most cases of practical interest. Based on experience with the presented gradient estimation and Line Sampling algorithm, similar convergence histories apply in numerous cases. This suggests the general statement that the presented parallel gradient estimation and Line Sampling algorithms are efficient for *nw* up to about 100.

Speedup measurements In this section results based on wall-clock measurements are reported for the parallel Line Sampling of the multi-story building model. The more fine-grain parallelization scheme represented by Algorithm 3 (sample-wise parallelization) has been used, even though in the present case with a maximum number of workers of $nw = 16$ the ideal performance of the Algorithm 2 (line-wise parallelization) would have been also satisfactory.

With reference to Table 1 the algorithm parameters of the corresponding parallel execution of

Table 2 Parallel Line Sampling - Algorithm parameters

Algorithm parameters	***nw***				
	1	**2**	**4**	**8**	**16**
Gradient estimation					
Initial Monte Carlo sample pool, N_0	32	32	32	32	32
Gradient component evaluation - stage 1, $N_{FD}^{(1)}$	———		8	———	16
Monte Carlo sample pool upgrade, ΔN	———		8	———	16
Gradient component evaluation - stage 2, $N_{FD}^{(2)}$	———		8	———	16
Line Sampling					
Number of line batches	13	13	13	7	4
Lines per batch, N_{BL}	1	1	1	2	4
Total number of lines, N_L	13	13	13	14	16
Samples per line, N_c	———		4	———	
Samples per batch	4	4	4	8	16

Line Sampling are presented in Table 2. Two items are particularly noteworthy: firstly, the gradient estimation performed with $nw = 16$ did not converge after the first stage, even though the number of evaluated gradient components, 16, is equal to the number of components that sufficed for the other cases of *nw*. This shows that with the initial sample pool of 32 samples the correlation-based statistical importance measures are not sufficiently accurate to reveal all the most important parameters, needed to achieve convergence. As for the other cases of *nw*, a sample pool upgrade is needed to identify also the remaining important components. For *nw* the upgrade size ΔN is set equal to *nw*, to avoid idle workers. Secondly, the total number of lines, N_L, needed to achieve convergence of the Line Sampling algorithm is larger for $nw = 8$ and 16 ($N_L = 14$ and 16, respectively), than for the other cases of *nw* ($N_L = 13$). This is due to the fact that N_L of the sequential algorithm ($nw = 1$) is not a multiple of the number of lines per batch for $nw = 8$ and 16 ($N_{BL} = 2$ and 4, respectively). Therefore, some lines of the last batch are evaluated in excess.

Fig. 9 shows results based on speedup measurements of the parallel gradient estimation and Line Sampling algorithm, with a convergence history according to Table 2. The results on the left relate to the Xeon cluster, those on the right to the Opteron cluster. Speedup results are presented for different cases of node loading, in terms of the number of cores simultaneously used on each node (*cpn*, cores per node). For instance, for the triangular markers only one core (out of eight) per node has been used, while the remaining seven cores remained idle. The number of utilized cluster nodes corresponding to a given datum of the speedup plot is equal to the ratio *nw*/*cpn*. (This applies to Fig. 12 as well.)

Two types of results are shown in Fig. 9: the bold markers ("measured") are based on the measurement of the wall-clock times for the entire algorithm. The remaining markers ("estimated") are based on the measured wall-clock time of the sample evaluation progress of a reference batch, as shown in Fig. 10. From this time-profile the execution time of a batch of *N* samples can be estimated for any given combination of the number of workers *nw* and the number of used cores per node cpn. The estimation has been conducted under the condition that *nw* is a multiple of *cpn*

$$nw = nn \cdot cpn, \qquad \text{where} \qquad nn \ldots \textit{no.} \text{ of used nodes} \tag{25}$$

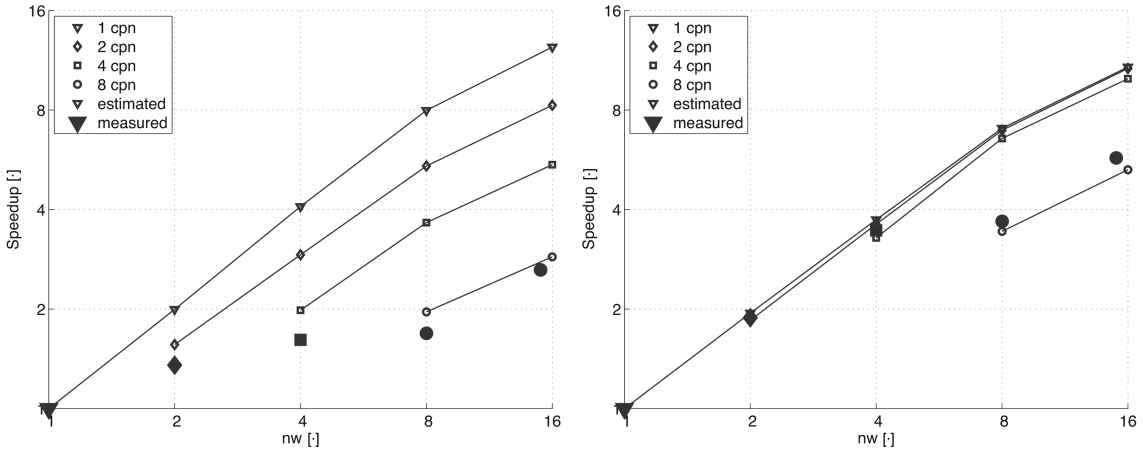

Fig. 9 Speedup of parallel gradient estimation and Line Sampling for multi-story building, based on wall-clock time measurements; left: Xeon, right: Opteron

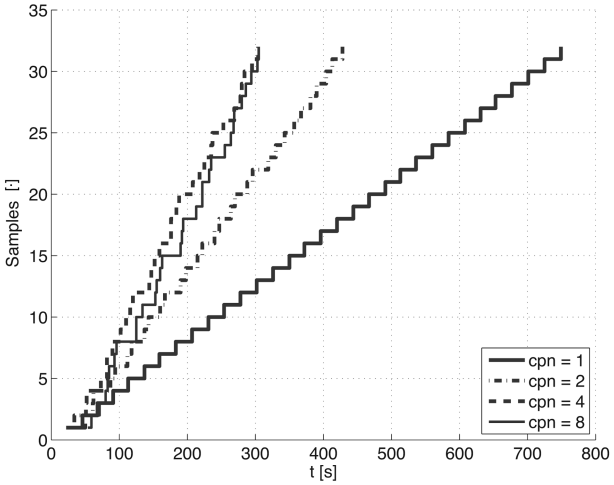

Fig. 10 Time profile of sample evaluation progress of reference batch (Xeon) on a single node

and under the assumption that the same time profile applies to each node. In this case the number of samples to be processed at most by each of the nodes is

$$Npn = \text{ceil}(N/nn) = \text{ceil}\left(\frac{N \cdot cpn}{nw}\right) \tag{26}$$

and the execution time of N is then simply obtained by intersecting the time profile of the selected value of *cpn* with the horizontal line corresponding to a number of samples equal to *Npn*. The reason for resorting to this estimation is that on the investigated clusters only two nodes each were available, with exactly the same hardware configuration (Xeon) or for which exclusive use by the author was granted, thus avoiding potential performance loss due to other simultaneously running jobs on the same node (Opteron).

As seen from Fig. 9 the agreement between the measured speedup of the entire algorithm and the speedup estimated from measurements of reference batches is reasonable enough to use the estimated speedup for an assessment of the parallel performance on clusters of multi-core processors. In particular, Fig. 9 shows that with increasing node loading, i.e. as *cpn* approaches eight, the speedup and the efficiency deteriorate quite dramatically, compared to the ideal performance. More specifically, the Xeon cluster (left) exhibits a greater sensitivity to the number of busy cores on a given node, *cpn*.

3.4 Parallel subset simulation

Convergence behavior of the sequential algorithm The Subset Simulation algorithm applied in the present case study corresponds to the description provided in section 2.1.4. The details on the parameters convergence behavior of the algorithm in the sequential mode are reported in Table 3. The number of samples evaluated at each level, $N_0 = 23 \times 9 = 207$, has been specified so as to obtain a CoV of the intermediate p_{F_i} of about 20%. Indeed, for $N_0 = 207$, $CoV(p_{F_i}) = \sqrt{1 - p_{F_i}/p_{F_i} N_0} = 20.9\%$.

Ideal Parallel Performance Applying the same assumption as in the case of Line Sampling, the ideal parallel performance of parallel Subset Simulation, using Algorithm 4 in section 2.2.2, can be computed.

For the convergence history reported in Table 3, the ideal wall-clock time amounts to

$$T_4(nw) = \text{ceil}\left(\frac{207}{nw}\right) + 3 \cdot 9 \cdot \text{ceil}\left(\frac{23}{nw}\right)$$

Table 3 Subset Simulation - Algorithm parameters and convergence history

Algorithm parameters	
Direct Monte Carlo Simulation	
Initial Direct MCS (subset simulation level 1), N_0	207
Subset Simulation	
Number of concurrent chains per level, N_C	23
Number of steps per chain, N_K	9
Intermediate failure probability, F_i	0.1
Proposal PDF for offspring generation	uniform
Window size for offspring generation, $\Delta\xi$	0.8

Convergence history	**No. FE runs**	**Intermediate Limit State g_i**
Direct Monte Carlo Simulation		
Initial Direct MCS (subset simulation level 1)	207	0.0227
Subset Simulation		
Markov chain sampling - subset simulation level 2	23 × 9	0.0210
Markov chain sampling - subset simulation level 3	23 × 9	0.001488
Markov chain sampling - subset simulation level 4	23 × 9	0.0
Total	828	

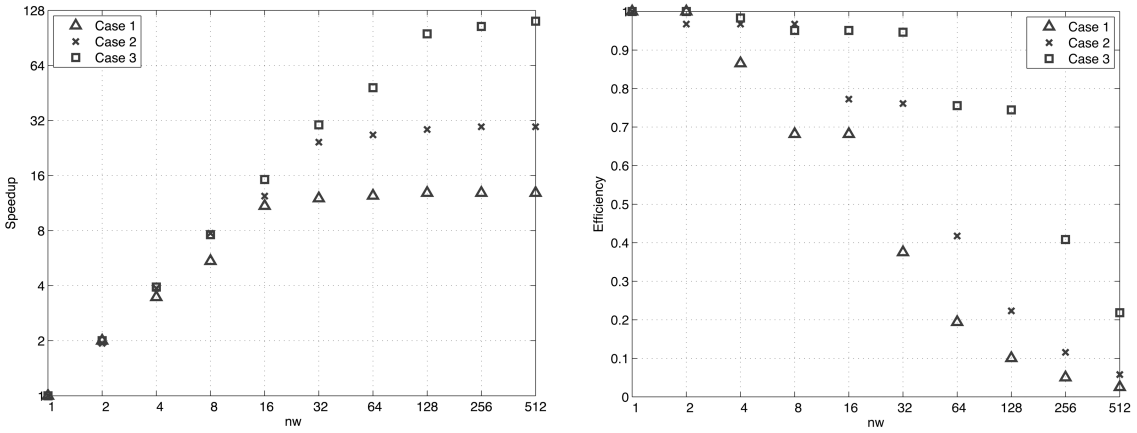

Fig. 11 Ideal performance of parallel Subset Simulation for multi-story building

It is recalled that the algorithm parameters N_C (number of simultaneously advanced Markov chains) and N_K (number of steps in each chain) depend on the required accuracy, expressed by $CoV(p_{F_i})$. In the actual performance analysis, the parameters have been set such that $CoV(p_{F_i}) \approx 20\%$. For a more comprehensive discussion, two additional cases are studied here, as summarized in the following table:

Case	p_{F_i}	$CoV(p_{F_i})$	N_K	N_C	N_0
1	11%	30%	9	10	90
2	11%	20%	9	23	207
3	11%	10%	9	90	810

For cases 1 and 3 the above expression for $T_4(nw)$ has to be modified accordingly.

Fig. 11 shows the ideal speedup for different numbers of workers. In general, the speedup of parallel Subset Simulation (using Algorithm 4), similar to that of Line Sampling, is initially almost linear with respect to *nw*. Depending on the specified accuracy (in terms of $CoV(p_{F_i})$), the speedup remains nearly constant above a certain number of workers *nw*. This value of *nw*, from which on the speedup stagnates, is larger if a higher accuracy is specified (i.e. $CoV(p_{F_i})$ is smaller), since in this case a greater number of Markov chains, N_C, is advanced and since the degree of parallelism of Algorithm 4 is equal to N_C (except in the initial phase).

On the basis of these results, the presented algorithm for parallel Subset Simulation (Algorithm 4) can - in fairly general terms -be considered as efficient for *nw* up to about 50 workers, if the required accuracy is low ($CoV(p_{F_i}) \geq 20\%$) and up to 100 workers, if the required accuracy is high.

Speedup measurements The results presented in this section are based on wall-clock measurements of the multi-story building model and on the use of algorithm 4 in section 2.2.3. The parameters of the Subset Simulation algorithm used in the parallel computation (i.e. the number of Markov chains and of the steps at each level) are identical to those in the sequential version of the algorithm, listed in Table 3.

Fig. 12 depicts the dependence of the speedup on the number of workers *nw* and on the number

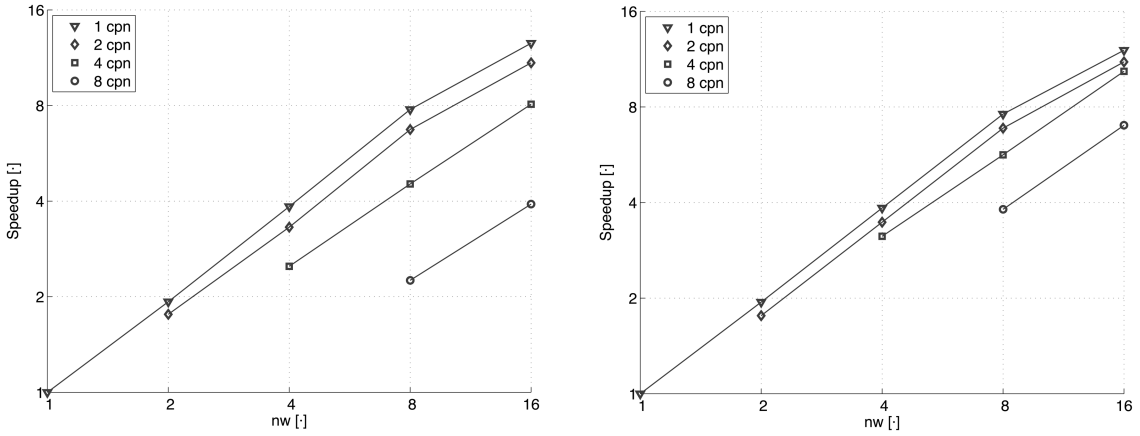

Fig. 12 Speedup of parallel Subset Simulation for multi-story building, based on wall-clock time measurements; left: Xeon, right: Opteron

of cores used on each node *cpn*, both for the Xeon (left) and the Opteron (right) cluster. The shown speedup results have been estimated as described in section 3.3, based on wall-clock time measurements leading to a time profile for a reference batch of 207 samples, as well as on the convergence history listed in Table 3.

The observations that may be gathered from these plots are qualitatively similar to those emerging from the corresponding figure for Line Sampling, i.e. Fig. 9. As a general trend, the efficiency of parallel Subset Simulation is slightly higher than that of parallel Line Sampling. This is related to the fact that in Subset Simulation the number of samples contained in each batch of samples is significantly larger than in Line Sampling. Indeed, in the initial Subset Simulation step a total of 207 samples have to be evaluated in parallel and also in the later steps, each sample batch comprises 23 samples. In contrast, in Line Sampling the maximum number of samples processed in parallel is 32 and at later stages mostly batches of only 8 are evaluated. Larger sample batches have a beneficial impact on the speedup, because the load-balancing of the parallel sample evaluation based on the master-slave paradigm becomes active only for a somewhat larger number of samples.

It should however not be overlooked that the absolute execution time required for Subset Simulation is significantly larger (about one order of magnitude) in this case study.

Subset simulation including speculative computing The purpose of this example is to analyze the potential for further reduction of the wall-clock time of parallel subset simulation by resorting to speculative computing, as introduced with Algorithm 5 in section 2.2.3. For this purpose Fig. 13 compares the ideal speedup of parallel subset simulation based on parallel chain advancement (Algorithm 4) with the parallel subset simulation algorithm including speculative computing (Algorithm 5). It should be noted that these ideal speedup curves apply to the Subset Simulation of the multi-story building, as presented in this paper, and to all other cases in which the number of Markov chains is $N_C = 23$, the number of steps in each chain $N_K = 9$ and the subset simulation converges after three levels.

The left portion of Fig. 13 (logarithmic scale) indicates that by using speculative computing the wall-clock time can be further reduced, even after the number of workers *nw* exceeds the number of chain, $N_C = 23$. For instance, with $nw = 128$ and $nw = 256$ the wall-clock time is reduced to about

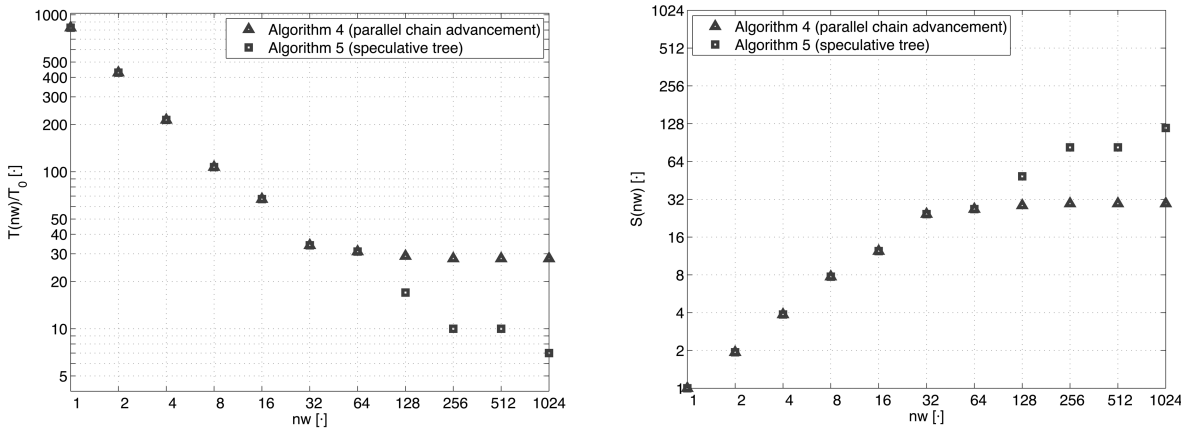

Fig. 13 Ideal wall-clock time (left) and speedup (right) of parallel subset simulation for multi-story building, including speculative computing; T_0 . . . execution time of one FE analysis on one worker

one half and one third, respectively, of what can be achieved by chain-wise parallelization. This is accomplished thanks to the use of a speculative tree of order two and three, respectively.

Given that compute clusters of multi-core processors, in which the number of available workers totals 200 or more, are rather common nowadays, it can be concluded that speculative trees of order two and three can be effectively used to reduce the wall-clock time of subset simulation. In order to achieve an additional reduction of the execution time a speculative tree of order 4 is needed; in Fig. 13 the case of $nw = 1,024$ corresponds to this order. In terms of efficiency the latter case is already quite unfavorable, since a very large number of discarded samples is required at this point.

For a more comprehensive assessment of its merits, the measured speedup of speculative computing is to be compared with the corresponding speedup of alternative parallel subset simulation algorithms, in particular those in which parallelisms at a lower level are exploited. While the algorithmic efficiency will be inferior in most situations, due to the large number of discarded samples, it is anticipated that from a practical point of view parallel subset simulation with speculative trees of order two or three will prove useful if time-to-solution has highest priority. Indeed, this approach permits the utilization of an existing FE model without modification and hence in a black box fashion. This may not be the case if the parallelism at a lower level is to be exploited.

3.5 Speedup degradation on multi-processor multi-core server nodes

The results in the previous sections indicate, both for Line Sampling and Subset Simulation, that if the number of busy cores on the utilized nodes increases, then the observed speedup experiences significant degradation compared to the algorithmic speedup. Obviously, for the analyzed multi-story building the resources shared by the individual cores an a given node are on the critical path.

The memory requirements due to the size of the mass and stiffness matrix (in total approximately 15 MB) are not nearly approaching the size of the available RAM, even if eight concurrent sample evaluations are conducted on a single node. Consequently, the cache size constitutes a bottleneck, as it is inferior to the memory required to store mass and stiffness matrix. The problem of cache efficiency is recognized as a pressing problem in computational mechanics (Hartmann *et al.* 2008,

Coutinho *et al.* 2006), especially due to the fact that the performance growth of processing power and memory is highly asymmetric to the disadvantage of the latter. Future investigations are planned to address this issue in the context of advanced Monte Carlo based structural reliability.

For this specific example, the following options for improving the performance on clusters of multi-core nodes are currently being investigated.

- **Solver optimization:**

By far the greatest portion of the execution time is accounted for by the extraction of a small subset of the eigenvalues of the structural matrices, using a solver performing Arnoldi iterations, namely Arpack. In the present case study the solver has been used on directly through the corresponding Matlab interface. Chances are that by optimizing the performance the observed speedup degradation can be alleviated; alternatively, the performance of alternative eigensolvers may be evaluated.

- **Use of a parallel eigensolver:**

The Arnoldi iterations used to extract some of the eigensolutions can be parallelized. The efficiency of parallelizing these iterations on multi-core nodes is then to be investigated in the context of Line Sampling and Subset Simulation and the overall efficiency to be compared with the efficiency of the high level parallel algorithms presented here.

- **Substructuring:**

In order to reduce the size of the matrices, on which every worker (and hence core) operates, substructuring techniques may be applied and each core may be assigned to handle the matrices associated with only one substructure rather than the full size matrices of the complete system. Again, the overall efficiency of parallel reliability analysis is to be measured; it is expected that the efficiency of this approach increases with the size of the underlying FE model.

3.6 Comparison parallel line sampling and subset simulation

This final subsection of the case study is meant to provide an overview of the efficiency of the parallel advanced Monte Carlo simulation procedures for reliability analysis.

Fig. 14 depicts the execution time (in logarithmic scale) vs. the number of cores vs. the failure

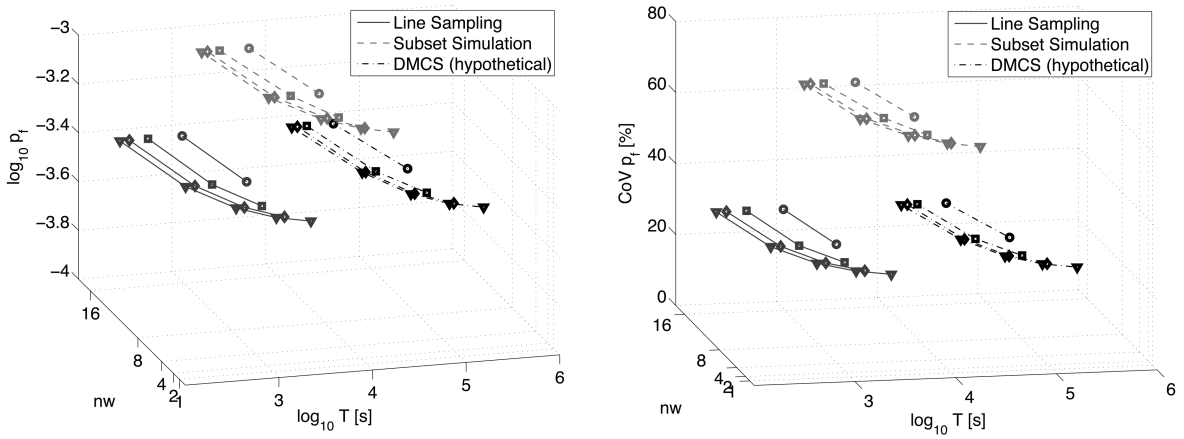

Fig. 14 Comparison of Parallel Line Sampling and Subset Simulation for multi-story building on Xeon cluster; left: estimate of failure probability p_f; right: variability of the estimate (CoV)

probability estimate (left portion of the figure) and vs. the CoV of the estimate (right portion of the figure). The shown results relate to Line Sampling (continuous) and Subset Simulation (dashed), as well as for Direct Monte Carlo (DMCS, dash-dotted) simulation. For the latter method, the results are an estimate, based on the requirement that the number of samples used in DMCS lead to a CoV of the estimate $\hat{p}_F$ of 30%, based on Eq. (3). The meaning of the markers corresponds to that in previous figures, e.g., Fig. 12, i.e., circular markers denote the case in which $cpn = 8$.

The figure indicates that in the present example the Line Sampling method involves the smallest wall-clock time. This is due to the only mildly non-linear dependence of the limit state function on the scaling value c of the Line Sampling scheme (cf. Fig. 6), which implies that a small number of samples is needed. For the present example the Subset Simulation method has a wall-clock time that is almost an order of magnitude higher than Line Sampling. Furthermore, as indicated by the right portion of the figure, the accuracy of the Subset Simulation algorithm run with the parameters defined according to Table 3 is somewhat smaller in this case (60% CoV vs. $\approx$ 30% of Line Sampling). It should be noted that obviously this observation applies to this particular example and cannot be expected to hold in general. Clearly, Direct MCS involves a wall-clock time that is another order of magnitude higher and would further grow, if failure probabilities smaller than the one estimated here would be interest, such as 10^{-6}.

4. Conclusions

The present paper analyzed the parallel performance of advanced, simulation-based algorithms for reliability analysis and justifies the following conclusions:

- Both Line Sampling and Subset Simulation are suitable for parallel processing in that the inherent degree of parallelism of these algorithms is significant and corresponds to the number of samples that can be computed simultaneously in the course of these algorithms. This number ranges from around 10 to about 100 in most applications, including the presented case study.
- The limits on the exploited parallelism, i.e., the possibility to perform independent FE analyses, are due to the need to update convergence measures in the case of Line Sampling and due to the limited number of simultaneously advanced Markov chains in Subset Simulation. For the latter method the wall-clock time can be further reduced by utilizing speculative trees of order two or three. Speculative trees of order higher than that will in most cases be of low interest due to the rapidly growing amount of discarded samples.
- Alternative parallel reliability analysis algorithms can be formulated by exploiting lower levels of parallelism, such as those associated with the solution of linear equation systems or eigenvalue problems. Current efforts are directed towards implementing and comparing alternative algorithms, in particular those which make use of parallel eigensolvers and/or substructuring methods.
- The FE model of a multi-story building (24,000 DOFs) has served as a case study of the efficiency of the proposed parallel algorithms, both in terms of the ideal speedup and on the basis of measurements of the wall-clock time. The reduction of the time-to-solution, due to the advanced Monte Carlo simulation algorithms and due to parallelization is significant and ensures - for this a medium-size problem - improved compatibility with the current pace of the design workflow:

	Sequential	Parallel
Direct MCS	> 1 day	3 hours
Advanced MCS - Line Sampling	45 min	< 10 min
Advanced MCS - Subset Simulation	6 hours	< 1 hour

- However, this case study also revealed the sensitivity of the measured performance to the load level of the utilized nodes, in terms of the number of cores simultaneously performing FE analyses. This issue can be expected to appear whenever the FE model used in the reliability analysis exceeds a certain size. Since multi-core machines are becoming increasingly common, it is therefore imperative to investigate strategies for alleviating this memory-related bottlenecks, for instance by exploiting lower level parallelisms.

Acknowledgements

The author wishes to thank Professor Schuëller for his continued interest, encouragement and support. This work has been conducted with the support of the FWF-project L269-N13 "Quality assurance of mechanical systems and structures" (project leader: Professor Schuëller). The assistance and advise by Prof. Fahringer, Dr. Moritsch and Mr. Niederwieser, in the framework of the HPC (High Performance Computing) Consortium of the University of Innsbruck, is gratefully acknowledged. Finally, the helpful comments of the two anonymous reviewers have been appreciated.

References

Alonso, J., de Alfonso, C., García, G. and Hernández, V. (2007), "Grid technology for structural analysis", *Adv. Eng. Software*, **38**(11-12), 738-749.

Au, S.K. and Beck, J. (2001), "Estimation of small failure probabilities in high dimensions by subset simulation", *Probabilist. Eng. Mech.*, **16**(4), 263-277.

Bitzarakis, S., Papadrakakis, M. and Kotsopulos, A. (1997), "Parallel solution techniques in computational structural mechanics", *Comput. Meth. Appl. Mech. Eng.*, **148**(1-2), 75-104.

Charmpis, D. and Papadrakakis, M. (2005), "Improving the computational efficiency in finite element analysis of shells with uncertain properties", *Comput. Meth. Appl. Mech. Eng.*, **194**(12-16), 1447-1478.

Coutinho, A., Martins, M., Sydenstricker, R. and Elias, R. (2006), "Performance comparison of data-reordering algorithms for sparse matrix-vector multiplication in edge-based unstructured grid computations", *Int. J. Numer. Meth. Eng.*, **66**, 431-460.

Ditlevsen, O. and Madsen, H.O. (1996), *Structural Reliability Methods*, John Wiley & Sons, Chichester.

Farhat, C., Cortial, J., Dastillung, C. and Bavestrello, H. (2006), "Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses", *Int. J. Numer. Meth. Eng.*, **67**, 697-724.

Farhat, C. and Lesoinne, M. (1993), "Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics", *Int. J. Numer. Meth. Eng.*, **36**(5), 745-764.

Ghanem, R. and Kruger, R. (1996), "Numerical solution of spectral stochastic finite element systems", *Comput. Meth. Appl. Mech. Eng.*, **129**, 289-303.

Hartmann, J., Krahnke, A. and Zenger, C. (2008), "Cache efficient data structures and algorithms for adaptive multidimensional multilevel finite element solvers", *Appl. Numer. Math.*, **58**, 435-448.

Johnson, E.A., Wojtkiewicz, S.F., Bergman, L.A. and Jr., B.F.S. (1997), "Observations with regard to massively

parallel computation for monte carlo simulation of stochastic dynamical systems", *Int. J. Non-linear Mech.,* **32**(4), 721-734.
Johnson, E., Proppe, C., Spencer Jr, B., Bergman, L., Székely, G. and Schuëller, G.I. (2003), "Parallel processing in computational stochastic dynamics", *Probabilist. Eng. Mech.,* **18**(1), 37-60.
Keese, A. and Matthies, H. (2005), "Hierarchical parallelisation for the solution of stochastic finite element equations", *Comput. Struct.*, **83**, 1033-1047.
Koutsourelakis, P., Pradlwarter, H.J. and Schuëller, G.I. (2004), "Reliability of structures in high dimensions, part I: Algorithms and applications", *Probabilist. Eng. Mech.,* **19**(4), 409-417.
Krysl, P. and Bittnar, Z. (2001), "Parallel explicit finite element solid dynamics with domain decomposition and message passing: Dual partitioning scalability", *Comput. Struct*., **79**(3), 345-360.
Lehoucq, R., Sorensen, D. and Yang, C. (1998), *ARPACK Users Guide: Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*, SIAM: Philadelphia, PA.
Leite, J.P.B. and Topping, B.H.V. (1999), "Parallel simulated annealing for structural optimization", *Comput. Struct*., **73**, 545-564.
Mackay, D. and Law, K. (1996), "A parallel implementation of a generalized lanczos procedure for structural dynamic analysis", *Int. J. High Speed Comput*., **8**(2), 171-204.
Metropolis, N. and Ulam, S. (1949), "The monte carlo method", *J. Am. Stat. Assoc.,* **44**, 335-341.
Papadrakakis, M. and Kotsopulos, A. (1999), "Parallel solution methods for stochastic finite element analysis using monte carlo simulation", *Comput. Meth. Appl. Mech. Eng*., **168**(1-4), 305-320.
Papadrakakis, M., Lagaros, N. and Fragakis, Y. (2003), "Parallel computational strategies for structural optimization", *Int. J. Numer. Meth. Eng*., **58**(9), 1347-1380.
Pellissetti, M.F., Pradlwarter, H.J. and Schuëller, G.I. (2007), "Relative importance of uncertain structural parameters, Part II: Applications", *Comput. Mech.*, **40**(4), 637-649.
Pradlwarter, H.J. (2007), "Relative importance of uncertain structural parameters, Part I: Algorithm", *Comput. Mech*., **40**(4), 627-635.
Pradlwarter, H. and Schuëller, G. (2008, submitted), "Uncertain linear systems in dynamics, Part 2: Efficient reliability assessment", *Comput. Struct*.
Rubinstein, R. (1981), *Simulation and the Monte Carlo Method*, John Wiley & Sons, New York, Chichester, Brisbane, Toronto.
Saleh, A. and Adeli, H. (1996), "Parallel eigenvalue algorithms for large-scale control-optimization problems", *J. Aerospace Eng.,* **9**(3), 70-79.
Schuëller (Ed.), G.I. (1997), "A state-of-the-art report on computational stochastic mechanics", *Probabilist. Eng. Mech*., **12**(4), 197-321.
Schuëller, G.I. (2007), "On the Treatment of Uncertainties in Structural Mechanics and Analysis (based on a Plenary Keynote Lecture Presented at the Third M.I.T. Conference on Computational Fluids and Solids Mechanics), Boston, USA", *Comput. Struct*., **85**(5-6), 235-243.
Schuëller, G.I. and Pradlwarter, H.J. (2007), "Benchmark study on reliability estimation in higher dimensions of structural systems - an overview", *Struct. Safety*, **29**(3), 167-182.
Schuëller, G.I., Pradlwarter, H.J. and Koutsourelakis, P. (2004), "A critical appraisal of reliability estimation procedures for high dimensions", *Probabilist. Eng. Mech.*, **19**(4), 463-474.
Shinozuka, M. and Deodatis, G. (1997), "Parallel implementation of mcs — synthesis of seismic ground motion using a stochastic barrier model", *Probabilist. Eng. Mech., Special Issue on Computational Stochastic Mechanics,* **12**(4), 213-216.
Sotelino, E. (2003), "Parallel processing techniques in structural engineering applications", *Struct. Eng.,* ASCE, **129**(12), 1698-1706.
Székely, G., Pradlwarter, H.J. and Schuëller, G.I. (1998), Computational Stochastic Structural Analysis - Software Development, in S. Lydersen *et al*., ed., *Proceedings of the ESREL'98 -Safety and Reliability*, Vol. 2, A.A. Balkema, Rotterdam, Trondheim, Norway, 1013-1020.
Székely, G. and Schuëller, G.I. (2001), "Computational procedure for a fast calculation of eigenvectors and eigenvalues of structures with random properties", *Comput. Meth. Appl. Mech. Eng.,* **191**(8-10), 799-816.
Umesha, P.K., Venuraju, M.T., Hartmann, D. and Leimbach, K.R. (2005), "Parallel computing techniques for sensitivity analysis in optimum structural design", *J. Comput. Civil Eng.*, **21**(6), 463-477.

Valdebenito, M. and Schuëller, G. (2008), Quality Assurance of Uncertain Mechanical Systems Considering the effects of Fatigue and Fracture, in '8th World Congress on Computational Mechanics, (WCCM8)', Venice, Italy, EU.

Wriggers, P. and Boersma, A. (1998), "A parallel algebraic multigrid solver for problems in solid mechanics discretisized by finite elements", *Comput. Struct.*, **69**(1), 129-137.

Zienkiewicz, O. and Taylor, R. (2000), *The Finite Element Method*, 5th edn, Butterworth-Heinemann, Oxford, UK.