Computers and Structures 171 (2016) 46–53

Contents lists available at ScienceDirect

Computers and Structures

journal homepage: www.elsevier.com/locate/compstruc

GPU computing for accelerating the numerical Path Integration approach

P. Alevras [a], D. Yurchenko [b,*]

[a] Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, LE11 3TU, UK
[b] Institute of Mechanical, Process & Energy Engineering, Heriot-Watt University, Edinburgh EH14 4AS, UK

ARTICLE INFO

Article history:
Received 7 June 2015
Accepted 5 May 2016

Keywords:
Path Integration method
Probability density function
GPU computing
Random vibrations
Stochastic modeling
Ship roll
Reliability

ABSTRACT

The paper discusses a novel approach of accelerating the numerical Path Integration method, used for generating a stationary joint response probability density function of a dynamic system subjected to a random excitation, by the GPU computing. The paper proposes the parallelization of nested loops technique and demonstrates the advantages of GPU computing. Two, three and four dimensional in space problems are investigated as a part of the pilot project and the achieved maximum accelerations are reported. Three degree-of-freedom system (6D) is approached by the Path Integration technique for the first time. The application of the proposed GPU methodology for problems of stochastic dynamics and reliability are discussed.

1. Introduction

Efficiency of any computer simulations depend on three factors: development of the theory describing the process, numerical methods used and hardware capabilities. Until recently the computational capabilities have been developing surprisingly well, doubling the single-core processor performance twice every 18–24 months, as predicted by Moore's law 50 years ago. In recent years the doubling rate has slowed down forcing one to revisit the Moore's law. Latest observations indicate that the performance doubling happens every 5–7 years with the potential of flattening out completely in 10–20 years from now. Graphics Processing Unit (GPU) computing is another alternative that has been attracting more and more interest in last 10 years.

Graphics Processing Unit computing can be viewed today as the most powerful computational hardware. Originally GPU has been developed and used as a tool to manipulate and accelerate displayed images. Hence GPU was designed to process relatively small amount of data (pixels) in parallel, which was very different in nature compared to a CPU. In last 10 years the GPU performance, measured in Floating-point Operations per Second (FLOPS), has been doubling every year reaching hundreds of TFLOPS (10^{12}). The GPU advantage is mostly related to the number of threads (a single smallest code sequence that can be executed independently) it can host. Single or multiple core CPU systems may host twice as many threads as cores, so a 32 core workstation is capable of hosting 64 threads. However a number of CPU cores that can be managed is limited compared to GPU that can host thousands of threads. These amazing capabilities have opened a new research direction known as GPU computing.

It should be stressed that it is not a very straightforward task to use the GPU architecture effectively due to its stream processing feature. Indeed, if one thinks of a GPU as a matrix, each element of which is capable of performing a small computational task, then the required communication between thousands of cores/threads or writing/reading information to/from a CPU memory would take enormous amount of time, substantially decreasing the computational effectiveness and even reducing the overall performance below the CPU one. Thus, to use the GPU capabilities in full it is essential to adapt existing codes by parallelizing them and minimizing the amount of intercommunication. Mesh-free numerical methods for instance [1–3], used intensively in CFD, compatible of GPU computing in contrast to the majority of standard mesh-based numerical methods, where intercommunication is required.

Unfortunately, GPU computing was not much exploited in the area of nonlinear and stochastic dynamics in particular. The fact that parallel processing is a very powerful tool needed in stochastic mechanics was reported back in [4], when the multiprocessing computing had just been developing and GPU computing was about to germinate. Apparently, GPU computing can be used for a number of purposes such as generating various parametric maps, basins of attractions, and bifurcation diagrams. Some results of GPU computing can be found in solid mechanics, see for instance [5] and references therein. In this paper, the focus is on a more fundamental

* Corresponding author.
E-mail address: d.yurchenko@hw.ac.uk (D. Yurchenko).

http://dx.doi.org/10.1016/j.compstruc.2016.05.002
0045-7949/

concept used for describing a response of a stochastic system – a probability density function (PDF). Knowledge of a joint response PDF not only provides the information on the system statistical characteristics, but also helps in evaluating the system reliability. To find a stationary response PDF of a dynamical system one may have to find a solution to the corresponding Fokker–Plank–Kolmogorov (FPK) partial differential equation of parabolic type. It can be handled by conventional methods such as cell mapping technique [6] or Path Integration (PI) approach. Despite some advantage of one method over another, finding a joint response PDF of a generally nonlinear multi-degree-of-freedom (MDOF) system is computationally challenging and expensive, therefore mostly low dimensional 1–2 DOF systems resulting in 2–4D FPK equation (not counting time) have been analyzed, although the amount of real time required was tremendous [7–10]. The fact that the higher the order of a system the heavier the computational costs generated a term *the curse of dimensionality*. In some special cases techniques like FFT, decoupling or decomposition [11–13] can be adapted to reduce the amount of computational efforts.

In this paper the curse of dimensionality of the PI approach will be addressed through the GPU computing. The PI methods, in the sense we understand it here, was introduced by Feynman in quantum mechanics in an effort to generalize quantum mechanics connecting it to the classical one through time–space trajectories [14,15]. The suggested generalization substituted the classical action principle by the integration over all possible properly weighted paths/trajectories. The PI methodology in some way helped to explain the controversial and entangled result of the two- slit experiment [15]. Historical development of the PI method, starting from a formula derived by Onsager and Machlup [16] for the Ornstein–Uhlenbeck process to the numerical implementation of the PI method [7,17] can be found in [18]. Despite some lack in rigorous mathematical substantiation, unlike the well defined Wiener path integral [19], the PI method has been widely used in various areas of engineering, physics and finance [20–24]. In stochastic dynamics the PI method has been successfully adapted for Markov processes along with the Chapman–Kolmogorov equation for finding a response PDF, as well as reliability characteristics of a system [25–27].

The intent of this paper to fill the existing gap between the GPU computing and its application to the problems of analyzing high dimensional dynamical systems. It will be demonstrated that utilizing the simplest optimization procedure in the code through the parallelization of nested loops substantially accelerates the PI approach in finding a multidimensional joint response PDF. Many finds Cuda language, built to optimize and control the performance of a GPU card, difficult to work with, therefore OpenACC framework is used for parallelization of the nested loops. It should emphasized that being the extremely powerful GPU cannot be considered as panacea to all the difficulties one has to overcome in computing a multidimensional PDF. GPU memory is rather limited compared to the CPU memory, therefore computing very high dimensional problems will require the communication between GPU and CPU, thereby reducing the GPU efficiency. Nevertheless, the existing modern GPU cards (Tesla K80 for instance) allow stepping up to the next level and dealing with 4D–6D problems within a reasonable time frame, which was not possible 10 years before. Moreover, some clusters of GPUs can work in parallel and thus can handle significantly larger amount of data, generated by high dimensional dynamical systems.

2. Path Integration and its numerical implementation with GPU

The Path Integration method is based on an iterative approach for calculating numerically the response PDF of a system when the stochastic unknown process follows the Markov property. This method was used as an alternative for solving the FPK equation, which in many cases presents extreme difficulties in obtaining a solution for the transition PDF. Consider, then, an n-dimensional Ito process $\mathbf{X}$ for which the following SDE can be written in the general case:

$$\dot{\mathbf{X}} = \alpha(\mathbf{X}, t) + b(\mathbf{X}, t)\mathbf{Z}(t) \tag{2.1}$$

where α is the drift matrix and b the diffusion one, $\mathbf{Z}(t)$ is an m-dimensional vector of independent Gaussian white noise stochastic processes, for the components of which it holds $\langle \xi_i \xi_i \rangle = \delta(t)$. Without any loss of generality, and since the systems presented in this paper satisfy the following simplification, let us assume that $m = 1$ and that the noise process enters the system's equations only through the last term. Thus b becomes a vector for which $b^T = [0 \dots \sigma]$. The central part of the PI method is based on the total probability law, which after recalling that for a Markov process the Chapman–Kolmogorov equation is true it reads:

$$p(\mathbf{x}, t) = \int_{-\infty}^{\infty} p(\mathbf{x}, t|\mathbf{x}', t')p(\mathbf{x}', t')d\mathbf{x}'. \tag{2.2}$$

with the prime denoting a time t' before t. Based on Eq. (2.2), the PDF of the response process at any time t can be calculated by the integral over the system's state space, subject to knowledge of the TPD from t' to t and an initial PDF at time t'.

From a numerical point of view, one would need a discretized state space for which the PDF at time t' is known at all the mesh points as well as a time discretization $t = t' + \Delta t$. Furthermore, an expression for the TPD is required for the calculation to be possible. For a sufficiently small time step, it has been proven [8] that the TPD is a degenerate multivariate Gaussian distribution. Keeping in mind the assumptions made on b, which fulfill the scope of this paper, the TPD reads:

$$p(\mathbf{x}, t|\mathbf{x}, t) = \prod_{i=1}^{n-1} \delta\left(x_{i,} - x_i' - r_i(\mathbf{x}', t')\Delta t\right) \cdot \tilde{p}(x_n|\mathbf{x}') \tag{2.3}$$

where

$$\tilde{p}(x_n, t|\mathbf{x}', t') = \frac{1}{\sqrt{2\pi\sigma^2\Delta t}} exp\left\{ -\frac{\left[x_n - x_n' - r_n(\mathbf{x}', \Delta t)\right]^2}{2\sigma^2\Delta t} \right\} \tag{2.4}$$

In Eqs. (2.3) and (2.4) the propagation of $\mathbf{x}$ forward in time is required and for that a 4th-order Runge–Kutta scheme is used denoted by r_j, $j = 1 \dots n$, in order to achieve better accuracy in the deterministic path, which previous studies [8,28] have shown to be very important for the application of PI.

Assuming $t' = 0$ one could calculate the PDF at $t = \Delta t$ and by successively applying this procedure the PDF of the response could computed at any time t, based on the previous time step and provided that Δt is small enough. A discussion about the choice of the time step could be found in [28]. Also, note that this method requires the computation of $p(\mathbf{x}', t')$ at points that do not necessarily coincide with the mesh points used to discretize the state space. To overcome this, an interpolation technique is used, namely cubic B-splines, so that the required values could be computed.

A summary of the PI procedure may be formulated as following:

(1) The PDF at time t' is inserted as input and interpolated by use of cubic B-splines.
(2) For each grid point the value of the new PDF is calculated as follows:
   (a) Find the points along the q axis where the noise is induced, at which the TPD has significant contribution.
   (b) Map these points backwards with a Δt time step by the Runge–Kutta method. Now we know all the possible paths that could significantly influence the new PDF value at each grid point.

(c) Through the aforementioned interpolation, calculate the old PDF value at the backwards-mapped points.
(d) Calculate the new PDF value at the grid point from the integral of Eq. (2.2) substituting $p(\mathbf{x},t|\mathbf{x}',t')$ from Eq. (2.3).

(3) Check convergence to steady state of the algorithm through the following scheme:

$$\int_{\mathbf{x}} |p(\mathbf{x},t) - p(\mathbf{x}',t')| d\mathbf{x} < \epsilon \tag{2.5}$$

where ϵ is chosen properly.

Previous studies have shown that the PI method is a reliable and extremely accurate course of action, especially when the tails of the PDF are concerned. This is of special importance for problems where rare events associated with the tails, play a key role such as in reliability analysis. However, the curse of dimensionality results in the heavily increasing computational cost of mesh-based methods. The goal of the PI method is to calculate the elements of an n-dimensional matrix approximating the continuous response PDF, $p(i_1, i_2, i_3, \ldots, i_n)$ where $i_j = 1, 2, \ldots, k_j$ with k_j the size of the grid at the jth dimension. It is evident that adding a new dimension $n+1$ to the problem increases the necessary computations by a factor of at least k^{n+1}. This a serious constraint to the applicability of this method since contemporary problems in stochastic modeling extend to at least some DOFs, leading to a high number of dimensions in Eq. (2.1). Thus, it is reasonable that most of the cases reported in the literature and treated with the PI method are SDOF systems.

Given this restrictive drawback, it is paramount to explore any potential of reducing the execution time required for a computer code implementing the PI method. The intention of the new GPU computing trend is not to fully substitute computing using CPUs, but to introduce a cooperative framework where the compute intensive parts of the computer code, such as matrix calculations, are offloaded to the GPU, in order to exploit its parallel capabilities. Besides, the pioneering development of CUDA enables users to use GPU cards for scientific computing in full, however it requires explicit programming from the user and in the majority of cases it means the code has to be completely rewritten in CUDA language. Several frameworks have been developed to alleviate this need; one of which is OpenACC – a directive-based framework that allows offloading part of the computations to the GPU. The advantage of this framework is that the user is only required to identify the parts of the code that should be compiled for parallel execution in the GPU. This is performed through a range of available directives that the user inserts within the code. In that way, the selected regions are offloaded to the GPU, which typically contains thousands of compute cores, resulting in a more timely execution of the code.

The key feature of the PI method regarding the introduction of the GPU in the necessary calculation is that the desired values of $p(i_1, i_2, i_3, \ldots, i_n)$ are computed based on information available from the previous time step and constants. This means that each of the points of the mesh can be treated independently as long as there is synchronization between different time steps. Considering this, the parallelization of the code for execution in the GPU is performed according to the following logic. The code needs to perform a number of iterations for propagating the PDF in time, with the exact number of iterations being unknown in the general case. However, the mesh points used to discretize the state space are fixed throughout the whole code. Thus, it is possible to parallelize the part of the code that is concerned with the point-by-point calculation of the PDF.

The GPU algorithm is depicted in Fig. 1(a), the code is split into three regions. The first part, which includes initialization of the code, assigning values to parameters, discretization of the state space, storing a user report, etc., is executed serially in the CPU. This part includes the initiation of the iterative loop that propagates the system forward in time in a step-by-step manner, as well as some preliminary calculations that are necessary for each time step, such as the B-splines coefficients (when allowed, see Fig. 1 (b)). The second part consists of n nested loops that access each point of the mesh, calculates and stores the value of the PDF at the particular time step. This is the part that is instructed to be compiled for execution in the GPU. Depending on the system's dimension, the size of the grid in each dimension is constrained by the capabilities of the available GPU(s), several threads (and blocks of threads) are created that perform the computations in parallel. Note that this part of the code is terminated before the master iterative loop is finished. This means in turn that the calculation of the PDF at time t is completed before the code moves on to the next time step, thus avoiding using the old values of the PDF at $t - \Delta t$ for the calculations at $t + \Delta t$ which would lead to erroneous results. The third, serially executed part includes the convergence checks, termination of the master loop, as well as some final operations mostly related to storing data.

3. Numerical results

In this section, the approach described earlier is applied to selected cases of stochastic systems. The purpose is to demonstrate the achieved acceleration by recording and comparing the execution times of the PI codes exclusively in the CPU with the time needed for the execution when the GPU is utilized too. To that end, all the factors that influence the performance such as the mesh size, the time step and the system's parameters were identical for each system to facilitate a fair comparison. The numerical results that follow for 2, 3 and 4D problems were obtained in a machine with two Intel® Xeon E5620 processors and an NVIDIA® Quadro K5000 with 1536 CUDA cores. The results for 6D problem were generated by 4xAMD Opteron 6376 and Quadro K6000 with 2880 CUDA cores. At the end of this section, a table is presented concentrating the times needed for the execution of the PI codes for all the cases that follow.

3.1. Linear system

The first system to be concerned is a linear SDOF oscillator excited by a Gaussian white noise for which $\langle \xi(t)\xi(t) \rangle = D\delta(t)$ with $D = \sigma^2$. The equation of motion of this basic system reads:

$$\ddot{x} + 2\alpha\dot{x} + \Omega^2 x = \xi(t) \tag{3.1}$$

where x is the displacement of the oscillator, Ω – its natural frequency and α the viscous damping coefficient. Transforming Eq. (3.1) to a system of first order differential equations with $x_1 = x$ and $x_2 = \dot{x}$ we get the following 2D system:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -2\alpha x_2 - \Omega^2 x_1 + \xi(t) \end{aligned} \tag{3.2}$$

This is a very basic stochastic system for which an analytical solution to the FPK equation has been obtained. In fact, the response PDF is a multivariate Gaussian one with $\langle x_1 x_1 \rangle = D/4\alpha$ and $\langle x_2 x_2 \rangle = D/4\alpha\Omega^2$. Given this knowledge, Eq. (3.2) is often used for benchmark studies involving an initial assessment of other methods. Then, the problem is solved with the numerical PI method as described in Section 2 by sole use of the CPU, which for brevity will be referred to as CPU computing, as well as

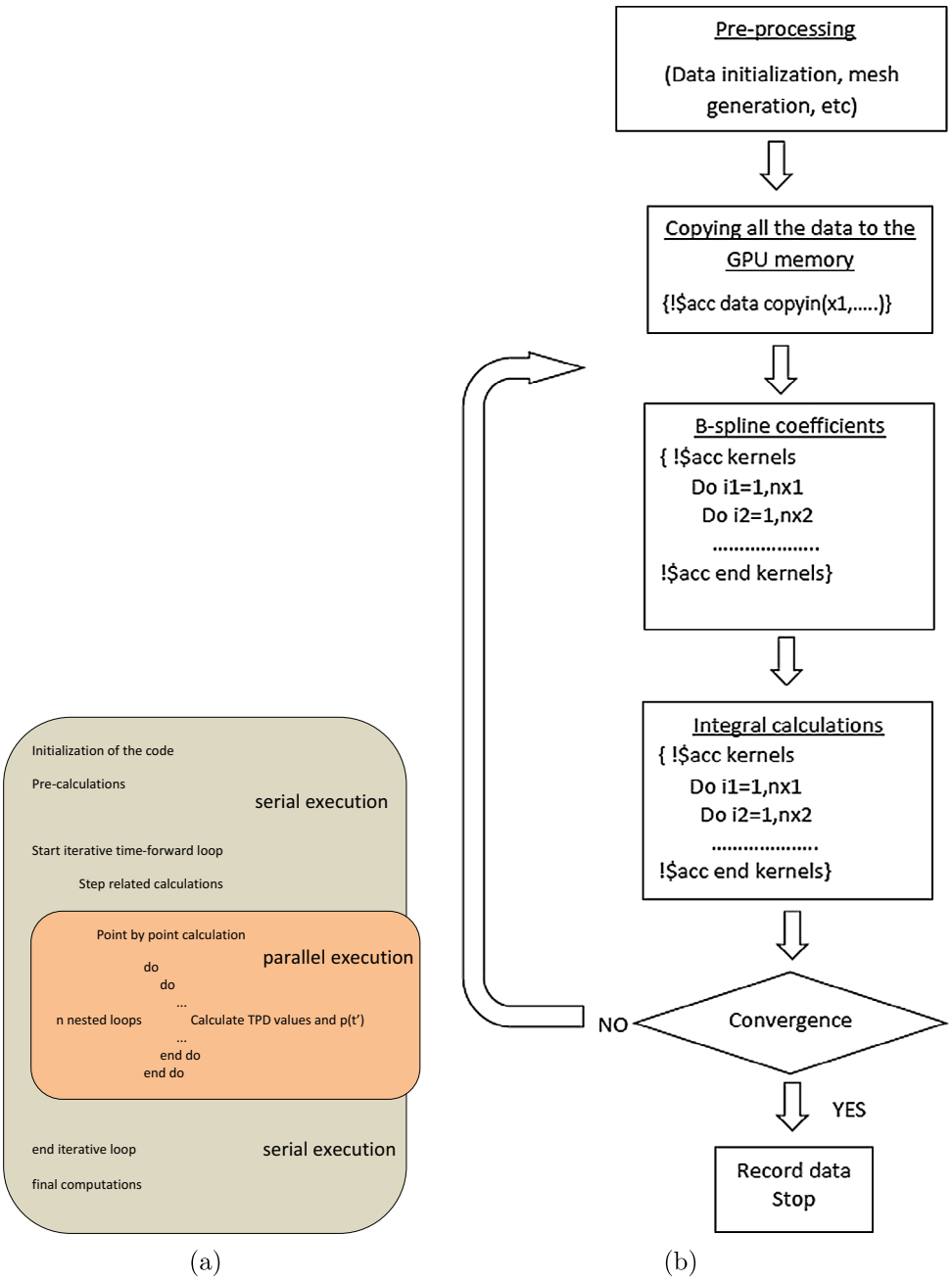

Fig. 1. Sketch (a) and algorithm (b) of the code marking the regions of the code executed serially in the CPU and parts offloaded to the GPU memory for parallel execution.

offloading part of the computations to the GPU. The execution time for both approaches is recorded seeking to quantify the acceleration of the code's execution. The necessary discretization is performed with a 101×101 mesh that spans along proper ranges estimated by MC simulations and the time step is chosen to be $\Delta t = 0.0035$ s, according to the scheme found in [7].

Fig. 2 shows the joint response PDF of Eq. (3.2) calculated with the PI method and implementing the GPU approach for $\alpha = 0.15$, $\Omega = 1.0$ and $D = 0.5$. Comparing the results with the regular computation in the CPU, which can be seen in Fig. 3 for the PDF of x_1 shows an almost ideal corroboration, indicating that any errors in the parallelization structuring have been avoided. When compared with the analytical solution, the second moment $\langle x_1 x_1 \rangle$ was found based on Fig. 2 to be $\langle x_1 x_1 \rangle_{num} = \langle x_1 x_1 \rangle_{an} = 0.8333$, while for x_2, it was found $\langle x_2 x_2 \rangle_{num} = 0.8342$ and $\langle x_2 x_2 \rangle_{an} = 0.8333$. The achieved acceleration of the execution time will be discussed later along with the presentation of the timings of all the examined cases.

3.2. Duffing nonlinearity

For the next case, a cubic nonlinear term is added to Eq. (3.1) resulting in a Duffing type nonlinearity. Applying the same standard transformation as in the linear system, leads to the following 2D system:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -2\alpha x_2 - \Omega^2 x_1 + \lambda x_1^3 + \xi(t) \end{aligned} \tag{3.3}$$

This is another well-known system for which an analytical solution to the FPK equation has been found and bares the interesting characteristic of having a double-well potential when λ is positive.

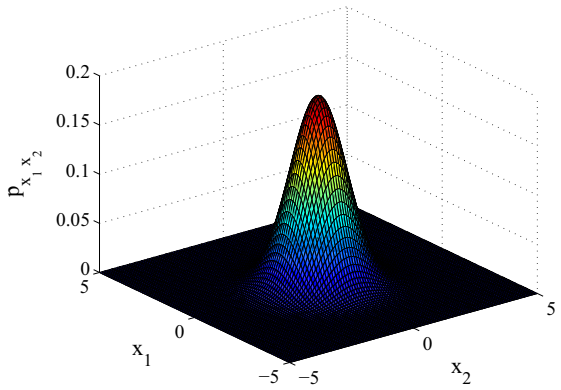

Fig. 2. Joint response PDF $p_{x_1x_2}$ for Eq. (3.2) calculated with the PI using the GPU computing approach for $\alpha = 0.15,\ \Omega = 1.0$ and $D = 0.5$.

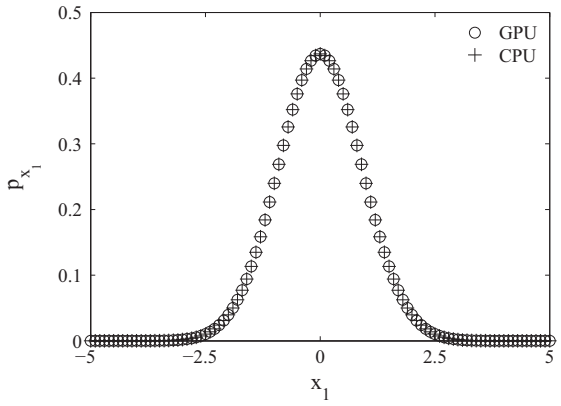

Fig. 3. Response PDF p_{x_1} for Eq. (3.2) calculated with the PI using both the GPU computing approach (○) and the CPU computing (+), for $\alpha = 0.15,\ \Omega = 1.0$ and $D = 0.5$.

Again, the system is solved via the PI method applying both CPU and GPU approaches and the recorded execution times are compared. The analytical solution is used here too to verify the numerical results. The discretization mesh is again chosen to be 101×101 and the time step $\Delta t = 0.005$ s.

Fig. 4 shows the joint response PDF of Eq. (3.3) calculated with the PI method and implementing the GPU approach for $\alpha = 0.15,\ \Omega = 1.0,\ \lambda = 0.25$ and $D = 0.5$, where the characteristic double-well structure of the PDF is obvious. In order to verify this computation, p_{x_1} is plotted against its counterpart stemming from the regular CPU computing application of the PI method in Fig. 5. Again, the agreement between the results of the two approaches is more than sufficient.

3.3. External imperfect periodic excitation

A step towards increasing requirements is taken by investigating the acceleration of the code for a SDOF oscillator subject to an external imperfect periodic excitation with random phase modulations. This system leads to the following 3D system of equations [29]:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -2\alpha x_2 - \Omega^2 x_1 + A\cos x_3 \\ \dot{x_3} &= \omega + \xi(t) \end{aligned} \tag{3.4}$$

The necessary introduction of a third dimension has a strong impact on the required time for the execution of the code even if the size of the 3rd dimension is relatively small, as it was taken

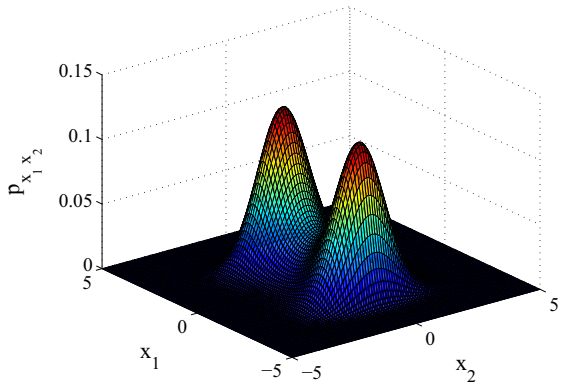

Fig. 4. Joint response PDF $p_{x_1x_2}$ for Eq. (3.3) calculated with the PI using the GPU computing approach for $\alpha = 0.15,\ \Omega = 1.0,\ \lambda = 0.25$ and $D = 0.5$. The characteristic double well structure is noticed.

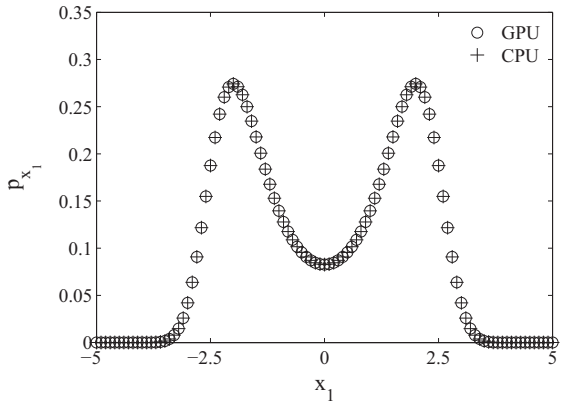

Fig. 5. Response PDF p_{x_1} for Eq. (3.3) calculated with the PI using both the GPU computing approach (○) and the CPU computing (+), for $\alpha = 0.15,\ \Omega = 1.0,\ \lambda = 0.25$ and $D = 0.5$.

to be herein. In fact, the space of x_3 is discretized at 22 points for $[-\pi, \pi)$, while the rest of the mesh is kept as before and the time step was $\Delta t = 0.03$ s.

For the sake of completeness, the joint PDF $p_{x_1x_2}$ is shown in Fig. 6 for $\alpha = 0.15,\ \Omega = 1.0$ and an excitation of $A = 1.0,\ \omega = 1.0$ and $D = 0.5$. The marginal PDFs p_{x_1} and p_{x_2} are plotted in Figs. 7 and 8 respectively and the results from the GPU and CPU approaches are again found to agree to a satisfactory extent.

3.4. Linear tuned mass damper

A 2-DOF system is considered posing the challenge of another 4th dimension in the numerical implementation of the PI method. To the best knowledge of the authors, the numerical solution of a 4D system calculated by the method described in Section 2 has not been reported before. We should mention though that the PDF of 4D systems has been computed in [11,30], applying a variation of the PI method where the Fourier transform is utilized to convert the integration in Eq. (2.2) to a convolution. This approach has been found to increase the efficiency of the PI code with respect to the execution time. Nevertheless, GPU computing can be integrated into the modified PI method in order to achieve better acceleration of the computation.

The equations of motion of a linear 2-DOF system modeling a tuned mass damper [31] can be found:

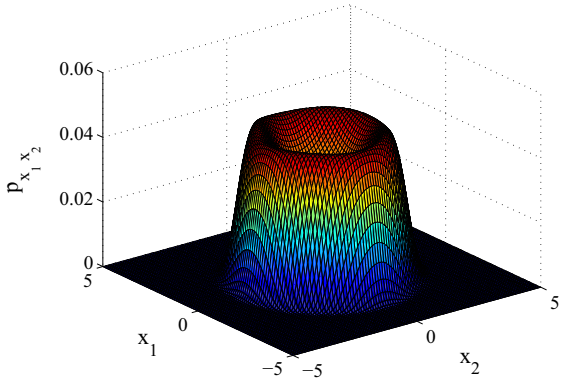

Fig. 6. Joint response PDF $p_{x_1x_2}$ for Eq. (3.4) calculated with the PI using the GPU computing approach for $\alpha = 0.15,\ \Omega = 1.0$ and an excitation of $A = 1.0,\ \omega = 1.0$ and $D = 0.5$.

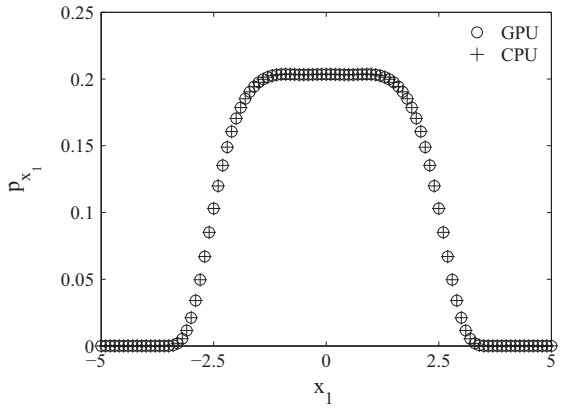

Fig. 7. Response PDF p_{x_1} for Eq. (3.4) calculated with the PI using both the GPU computing approach (∘) and the CPU computing (+), for the same parameters as in Fig. 6.

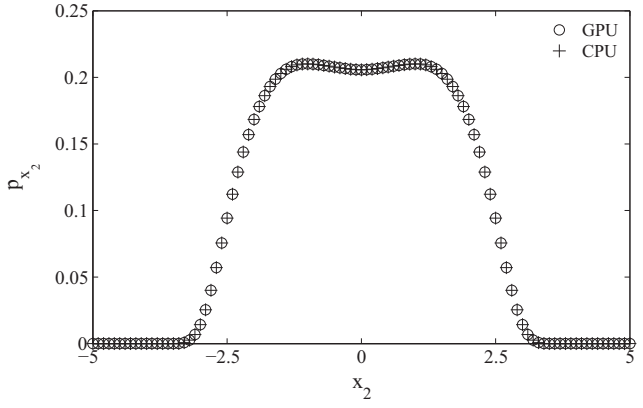

Fig. 8. Response PDF p_{x_2} for Eq. (3.4) calculated with the PI using both the GPU computing approach (∘) and the CPU computing (+), for the same parameters as in Fig. 6.

$$
\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= -2\zeta_2\Omega_2\mu(x_2 - x_4) - 2\zeta_1\Omega_1 x_2 - \left(\Omega_1^2 + \mu\Omega_2^2\right)x_1 + \Omega_2^2\mu x_3 + \xi(t) \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= -2\zeta_2\Omega_2(x_4 - x_2) - \Omega_2^2(x_3 - x_1)
\end{aligned}
\tag{3.5}
$$

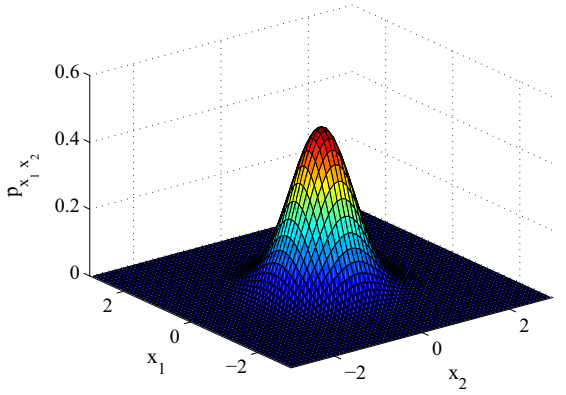

Fig. 9. Joint response PDF $p_{x_1x_2}$ for Eq. (3.5) calculated with the PI using the GPU computing approach for $\mu = 0.1,\ \zeta_1 = \sqrt{\mu}/2,\ \zeta_2 = 0,\ \Omega_1 = \Omega_2 = 1$ and $D = 0.1$.

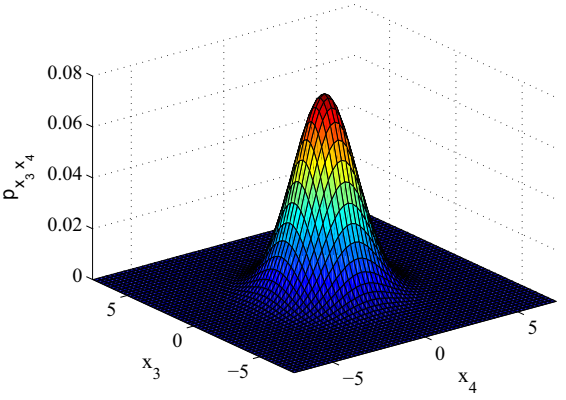

Fig. 10. Joint response PDF $p_{x_3x_4}$ for Eq. (3.5) calculated with the PI using the GPU computing approach for the same parameters as in Fig. 9.

where x_1 and x_3 are the displacements of the two masses M and m respectively, ζ_i the damping ratio of each mode, Ω_i their natural frequency and $\mu = m/M$. It is quite often in the analysis of tuned mass dampers to neglect the damping ratio of the primary mass, i.e. $\zeta_1 = 0$. The mass ratio is often chosen to be relatively small, and thus it is taken to $\mu = 0.1$. The natural frequencies are selected to be equal $\Omega_1 = \Omega_2 = 1.0$ while the noise intensity is $D = 0.1$. An interesting subject is the choice of an optimum value for the damping ratio of the secondary mass, designed to damp the motion of the primary one. Following [31], the latter is set to be $\zeta_2 = \sqrt{\mu}/2$.

Figs. 9 and 10 show the joint response PDFs $p_{x_1x_2}$ and $p_{x_3x_4}$ respectively, computed with the PI method and accelerated by the GPU approach. The focus is on the response PDF of the primary mass M, the motion of which is intended to be damped. Unfortunately, this result could not be extracted by the regular CPU computing due to the extreme computational cost that it would incur. Instead, standard MC simulations could be used to assess the response moments, although in this particular case an analytical solution for the second order moments is available. Based on the PDF shown in Fig. 9, the second moments are calculated as $\langle x_1x_1\rangle_{num} = 0.3489$ while the sampling showed $\langle x_1x_1\rangle_{MC} = 0.3479$. For the velocity, it is similarly found $\langle x_2x_2\rangle_{num} = 0.3153$ and $\langle x_2x_2\rangle_{MC} = 0.3193$. It could be seen that there is sufficient agreement between the PI results and the values extracted by MC simulations.

3.5. Timing

The core of this paper highlights the need to accelerate the numerical implementation of the PI method in order to more efficiently meet the demand for calculating the response PDF of

Table 1
Times required for the execution of the PI code for calculating the response PDF of the systems shown in Eqs. (3.2)–(3.5) (1st to 4th row respectively) using the CPU computing and the GPU computing approaches. The acceleration of the calculation achieved for each system is also shown in the last column.

System	Space dimen	GPU (s)	CPU (s)	Speed up
Linear	2D	37	711	×19
Duffing	2D	30	546	×18
External	3D	597	17,550	×29
TMD[a]	4D	3076	52,066	×17
TMD[a]	4D	1800	52,066	×29[b]

[a] Per 100 time steps.
[b] Calculated by NVIDIA K6000 card.

high-dimensional systems. Previously in this section, four different stochastic systems were solved with the PI method, applying the regular CPU and GPU computing approaches. Henceforth, the recorded execution times are presented in Table 1 along with the achieved acceleration.

First, the 2D systems, the linear one in Eq. (3.2) and the nonlinear one in Eq. (3.3) are found to demonstrate an acceleration of 19 times faster execution of the PI code and 18 times respectively. Even though the actual time needed to calculate the PDF via the CPU computing is not demanding since it spans to the scale of a few minutes, the much less time spent with the GPU computing is indicative of the potential acceleration. Furthermore, this achievement could be of advanced importance when high-throughput parametric studies of 2D systems are concerned allowing for either much better accuracy by increasing the size of the discretization mesh or a vast number of investigated cases in the unit of time. Better accuracy is of special interest in reliability problems where the influence of rare events, governed by the PDF's tail, can be better estimated.

Furthermore, when the 3D system in Eq. (3.4) is concerned, the acceleration is found to be even better, at 29 times faster execution, bringing the required time from the scale of a few hours down to a few minutes. This practically introduces the potential of parametric studies for 3D systems, since the computational cost of the CPU computing approach would inhibit any attempt to conduct such a bulky task. Note that the acceleration is 10 times more than the one for the 2D systems. This is a feature of the increased potential for parallelization, since the 3D system needs an extra nested loop to account for the 3rd dimension. This feature, however, is constrained by the utilized hardware and in fact, by the available GPU memory as well as the memory bandwidth.

The previous hardware constraint becomes evident when 2-DOF tuned mass damper in Eq. (3.5) or higher order systems are considered. The recorded acceleration on Quadro K5000 card was 17 times faster than that without it (CPU only) due to the fact that the 4D matrix required more memory than the GPU card had. Unfortunately OpenACC is not capable of working with many GPU cards, thus to cope with this problem the code was adapted to transfer data between GPU and CPU minimizing this exchange. It should be noted that the times presented in Table 1 for this case, are based on 100 iterations of the master loop, i.e. 100 time steps, since it was unfeasible to acquire a converging stationary PDF with the CPU approach due to the extreme computational cost. To conduct the experiment with enough GPU memory a K6000 NVIDIA card has been used. The acceleration of 29 times was recorded with that card, reducing the computational time to only 30 min per 100 iterations.

4. 6-D ship roll motion problem

The problem, reported in [32], describes the nonlinear roll motion of a ship due to Pierson–Moskowitz spectrum. The latter is obtained by using the measuring filters approach [33] and transmitting a white noise signal through two second order linear filters, rather than a single second order filter [34,35], so that the set of equations, including the ship dynamics, may be written as follows:

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -d_1 x_2 + d_2 x_2 |x_2| + d_3 x_1 - d_4 x_1^3 + x_3 \\
\dot{x}_3 &= x_4 - \lambda_1 x_3 \\
\dot{x}_4 &= x_5 - \lambda_2 x_3 + \xi(t) \\
\dot{x}_5 &= x_6 - \lambda_3 x_3 \\
\dot{x}_6 &= -\lambda_4 x_3,
\end{aligned} \tag{4.1}$$

where λ_i, d_i – are given constants and $\xi(t)$ – Gaussian zero mean white noise with intensity σ^2 [32]. This problem was treated by Monte-Carlo simulations and has never been treated by the PI approach.

It should be stressed that all previous calculations have been performed using a standard Microsoft Visual Studio compiler. It turns out that the time required to study the 6D problem, and we guess it is valid for any similar or higher order problems, depends substantially on the compiler and code architecture. For instance, a better compiler, like a PGI Parallel Fortran Compiler, which is used in the following 6D problem, can optimize the code performance for CPU computing, reducing the computational time. The code architecture can also be improved, but since Open ACC does not provide all the versatility available in CUDA language, one cannot expect the best of all performance from the code.

The code was executed on 4xAMD Opteron 6376 with Quadro K6000 and PGI FORTRAN compiler. Results of the numerical simulation have shown that GPU approach is 11 times faster than that of CPU for a single iteration. This acceleration may not seem as high as one we reported earlier, but since CPU takes about 38 h in real time (around 161M nodes in total) for the single iteration and on average it is required 300–500 iterations for convergence, reported acceleration is very much appreciated. Nevertheless, a single GPU card is yet not enough to speed up the calculations and further reduce the computational time. To improve this, a CUDA based PI code has to be created and run on a cluster of GPUs, however, this is a plan for future work, because OpenACC does not allow running the application on multiple GPU cards.

5. Conclusions

The paper presents an approach for accelerating the Path Integration method utilizing a GPU computing methodology. The proposed methodology resulted in a parallel execution of nested loops, which allowed significantly accelerating the calculations keeping the same accuracy of the results. It was reported that in the case of a SDOF system the achieved acceleration was 19 times, whereas for a TDOF system the maximum observed acceleration was 29 times. It is very well explained by the fact that for a SDOF system the GPU card was not fully loaded therefore the reported acceleration was lower than that for the TDOF system. Besides the proposed parallelization there are some other means of improving the existing code, which will be explored in the nearest future. The paper for the first time studies the application of the PI approach to a 6D system. Numerical results have shown that the computational time depends on the code architecture and compiler used. The particular example, studied in the paper, indicated 11 times acceleration using K6000 GPU card. High dimensional systems may appear as a result of a study of stochastic MDOF systems or lower order systems with non-Gaussian excitations [36]. The problems of reliability of large systems or networks, where the conventional Monte-Carlo simulation technique is not suitable

by a number of reasons [37], become a perfect candidate for the proposed GPU methodology as well. Moreover, mentioned techniques for external Gaussian excitation can be used along with GPU computing even further increasing the efficiency of the GPU computing.

Acknowledgments

The authors gratefully acknowledge the support of NVIDIA Corporation with the donation of a K6000 GPU card for this research.

The authors would like to thank Professor Scholz S.-B., Dr. Lee Y.C. and Dr. Viessmann H.-N., from Heriot-Watt University for fruitful discussions about GPU computing and use of GPU Robotarium.

References

[1] Panchatcharam M, Sundar S, Vetrivel V, Klar A, Tiwari S. GPU computing for meshfree particle method. Int J Numer Anal Model Ser B 2013;4(4):394–412.
[2] du Plessis O, Lee YC. Developing an accurate and efficient solution for modelling real and complex thin film and droplet flows. In: 27th Scottish fluids mechanics meeting. UK: University of St Andrews; 2014 [May].
[3] McKean A, Lee YC, Fruh W. Preliminary study of modeling ferromagnetic fluid using SPH. In: 27th Scottish fluids mechanics meeting. UK: University of St Andrews; 2014 [May].
[4] Johnson E, Proppe C, Spencer Jr B, Bergman L, Szkely G, Schuller G. Parallel processing in computational stochastic dynamics. Probabilist Eng Mech 2003;18(1):37–60.
[5] Liu W, Hong J-W. Discretized peridynamics for brittle and ductile solids. Int J Numer Methods Eng 2012;89(8):1028–46.
[6] Sun JQ. Control of nonlinear dynamic systems with the cell mapping method. Advances in intelligent systems and computing, vol. 175. Berlin Heidelberg: Springer; 2013. p. 3–18.
[7] Naess A, Johnsen J. Response statistics of nonlinear, compliant offshore structures by the path integral solution method. Probabilist Eng Mech 1993;8(2):91–106.
[8] Naess A, Moe V. Efficient path integration methods for nonlinear dynamic systems. Probabilist Eng Mech 2000;15(2):221–31.
[9] Wojtkiewicz S, Bergman L, Spencer B, Johnson E. Numerical solution of the four-dimensional nonstationary Fokker–Planck equation. In: Narayanan S, Iyengar R, editors. IUTAM symposium on nonlinearity and stochastic structural dynamics. Solid mechanics and its applications, vol. 85. Netherlands: Springer; 2001. p. 271–87.
[10] Yurchenko D, Mo E, Naess A. Response probability density functions of strongly non-linear systems by the path integration method. Int J Non-Linear Mech 2006;41(5):693–705.
[11] Mo E, Naess A. Efficient path integration by FFT. In: Proc 10th international conference on applications of statistics and probability in civil engineering (ICASP10), Tokyo, Japan, July 31–August 3.
[12] Beylkin G, Mohlenkamp MJ. Algorithms for numerical analysis in high dimensions. SIAM J Sci Comput 2005;26(6):2133–59.
[13] Sun Y, Kumar M. Numerical solution of high dimensional stationary Fokker–Planck equations via tensor decomposition and Chebyshev spectral differentiation. Comput Math Appl 2014;67(10):1960–77.
[14] Feynman R. Space–time approach to non-relativistic quantum mechanics. Rev Mod Phys 1948;20:367–87.
[15] Feynman RP, Hibbs AR. Quantum mechanics and path integrals. New York (USA): McGraw-Hill; 1993.
[16] Onsager L, Machlup S. Fluctuations and irreversible processes. Phys Rev 1953;91:1505–12.
[17] Wehner MF, Wolfer WG. Numerical evaluation of path-integral solutions to Fokker–Planck equations. Phys Rev A 1983;27:2663–70.
[18] Kougioumtzoglou I, Spanos P. An analytical Wiener path integral technique for non-stationary response determination of nonlinear oscillators. Probabilist Eng Mech 2012;28:125–31 [Computational Stochastic Mechanics {CSM6}].
[19] Wiener N. Generalized harmonic analysis. Acta Math 1930;55(1):117–258.
[20] Fuentes MA, Wio Horacio S, Toral Raul. Effective Markovian approximation for non-Gaussian noises: a path integral approach. Phys A: Stat Mech Appl 2002;303:91–104.
[21] Linetsky V. The path integral approach to financial modeling and options pricing. Comput Econ 1997;11:129–63.
[22] Skaug C, Naess A. Fast and accurate pricing of discretely monitored barrier options by numerical path integration. Comput Econ 2007;30:143–51.
[23] Jin H, Kellogg M, Mehrotra P. Using compiler directives for accelerating CFD applications on GPUs. In: Chapman B, Massaioli F, Mller M, Rorro M, editors. OpenMP in a heterogeneous world. Lecture notes in computer science, vol. 7312. Berlin Heidelberg: Springer; 2012. p. 154–68.
[24] Niemeyer KE, Sung C-J. Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. J Supercomput 2014;67(2).
[25] Naess A, Iourtchenko D, Batsevych O. Reliability of systems with randomly varying parameters by the path integration method. Probabilist Eng Mech 2011;26(1):5–9 [Special Issue: Stochastic Methods in Mechanics Status and Challenges].
[26] Kougioumtzoglou IA, Spanos PD. Response and first-passage statistics of nonlinear oscillators via a numerical path integral approach. J Eng Mech 2013;139(9):1207–17.
[27] Yurchenko D, Iwankiewicz R, Alevras P. Control and dynamics of a SDOF system with piecewise linear stiffness and combined external excitations. Probabilist Eng Mech 2014;35:118–24 [Stochastic Mechanics (SM12)].
[28] Mo E. Nonlinear stochastic dynamics and chaos by numerical path integration Ph.D. thesis. Norwegian University of Science and Technology; 2008.
[29] Yuchenko D, Naess A, Alevras P. Pendulum's rotational motion governed by a stochastic Mathieu equation. Probabilist Eng Mech 2013;31:12–8.
[30] Chai W, Naess A, Leira BJ. Stochastic dynamic analysis of nonlinear ship rolling in random beam seas. In: Proc 7th international conference on computational stochastic dynamics (CSM-7), Snatorini, Greece, 15–18 July.
[31] Yurchenko D. Tuned mass and parametric pendulum dampers under seismic vibrations. In: Beer M, Kougioumtzoglou IA, Patelli E, Au IS-K, editors. Encyclopedia of earthquake engineering. Berlin Heidelberg: Springer; 2015. p. 1–22.
[32] Chai W, Naess A, Leira BJ. Filter models for prediction of stochastic ship roll response. Probabilist Eng Mech 2015;41:104–14.
[33] Iourtchenko D. Response spectral density of linear systems with external and parametric non-Gaussian, delta-correlated excitation. Probabilist Eng Mech 2003;18(1):31–6.
[34] Alevras P, Yurchenko D. Stochastic rotational response of a parametric pendulum coupled with an {SDOF} system. Probabilist Eng Mech 2014;37:124–31.
[35] Yurchenko D, Alevras P. Stochastic dynamics of a parametrically base excited rotating pendulum. Proc {IUTAM} 2013;6:160–8.
[36] Alevras P, Yurchenko D. Stochastic rotational response of a parametric pendulum coupled with an SDOF system. Probabilist Eng Mech 2014;37:124–31.
[37] Zuev KM, Wu S, Beck JL. General network reliability problem and its efficient solution by subset simulation. Probabilist Eng Mech 2015;40:25–35.