Wind and Structures, *Vol. 21, No. 5 (2015) 461-487*
DOI: http://dx.doi.org/10.12989/was.2015.21.5.461

New GPU computing algorithm for wind load uncertainty analysis on high-rise systems

Wei Cui[a] and Luca Caracoglia[*]

Department of Civil and Environmental Engineering, Northeastern University, 360 Huntington Avenue, Boston, MA 02115, USA

(Received January 14, 2015, Revised July 2, 2015, Accepted July 24, 2015)

Abstract. In recent years, the Graphics Processing Unit (GPU) has become a competitive computing technology in comparison with the standard Central Processing Unit (CPU) technology due to reduced unit cost, energy and computing time. This paper describes the derivation and implementation of GPU-based algorithms for the analysis of wind loading uncertainty on high-rise systems, in line with the research field of *probability-based wind engineering*. The study begins by presenting an application of the GPU technology to basic linear algebra problems to demonstrate advantages and limitations. Subsequently, Monte-Carlo integration and synthetic generation of wind turbulence are examined. Finally, the GPU architecture is used for the dynamic analysis of three high-rise structural systems under uncertain wind loads. In the first example the fragility analysis of a single degree-of-freedom structure is illustrated. Since fragility analysis employs sampling-based Monte Carlo simulation, it is feasible to distribute the evaluation of different random parameters among different GPU threads and to compute the results in parallel. In the second case the fragility analysis is carried out on a continuum structure, i.e., a tall building, in which double integration is required to evaluate the generalized turbulent wind load and the dynamic response in the frequency domain. The third example examines the computation of the generalized coupled wind load and response on a tall building in both along-wind and cross-wind directions. It is concluded that the GPU can perform computational tasks on average 10 times faster than the CPU.

Keywords: wind engineering; parallel computing; GPU computing; uncertainty quantification; performance- based wind engineering; tall buildings

1. Introduction

1.1 Short review on computer architecture systems applied to engineering computations

In recent years, due to the non-negligible "quantum effect" (quantum tunneling), the Central Processing Unit (CPU) architecture, which has rapidly evolved for almost 40 years, has reached a "bottleneck" according to Moore's law (Moore 1998). The Moore's law is consequently near its end. In contrast, the Graphics Processing Unit technology (GPU), originally designed to efficiently

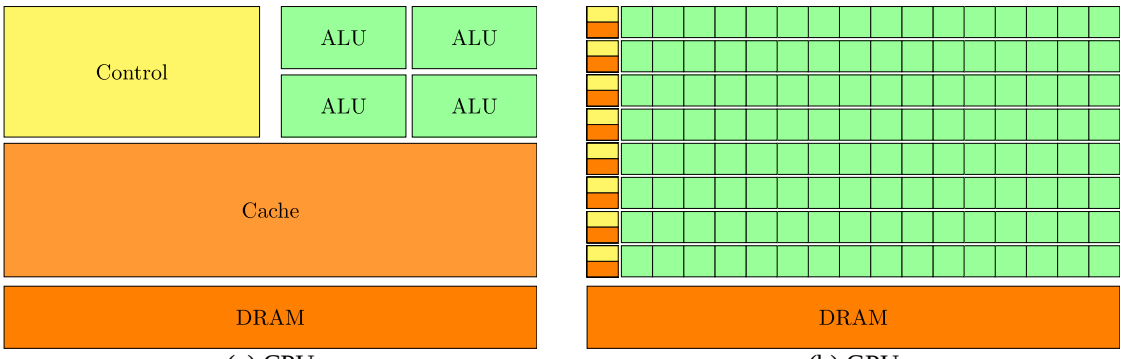
∗Corresponding author, Associate Professor, E-mail: lucac@coe.neu.edu
[a] Ph.D. Candidate, E-mail: cui.we@husky.neu.edu

http://www.techno-press.org/?journal=was&subpage=8 ISSN: 1226-6116 (Print), 1598-6225 (Online)

manipulate the creation and alteration of computer images, has drawn the attention of engineers and scientists in various fields for its potential aptitude to perform fast numerical operations.

The implementation of engineering computations on the GPU platform, using massive parallel processing, relies on the application and combination of basic numerical methods ("dwarfs"). In 2004, seven numerical dwarfs were identified, which have been important in science and engineering for at least one decade (Colella 2004, Asanovic *et al.* 2006). Later in 2006, six more dwarfs were added among the basic parallel computing methods (Colella 2004). Table 1 summarizes the total thirteen dwarfs. The thirteen parallel computing examples were initially developed for the multi-core computer and multi-node cluster technologies. It is, therefore, a natural and logical step to transfer the above-defined dwarfs from the multi-core or multi-node system to the GPU architecture.

The basic difference between GPU and CPU is the architecture design due to a distinct purpose of each computing method. The CPU, as the center of the whole computer, is designed to control and analyze the data flow for the entire system. Therefore, apart from the Arithmetic Logical Units (ALUs), the Control Unit (CU) and the Cache are also important components in the CPU architecture. In contrast, GPU, which is originally designed for rendering graphics on each pixel of the display, requires many more but "smaller" ALUs for performing the rendering work at the same time. Thus, in the GPU, the CU and Cache become less relevant since the ALUs are all doing similar work (i.e., rendering graphics). The schematic comparison between GPU and CPU architecture is illustrated in Fig. 1.



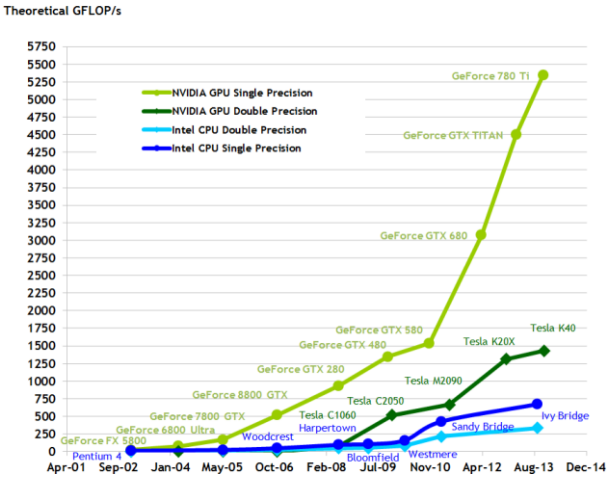
(a) CPU (b) GPU

Fig. 1 Schematic comparison between GPU and CPU architecture (Nvidia 2014)

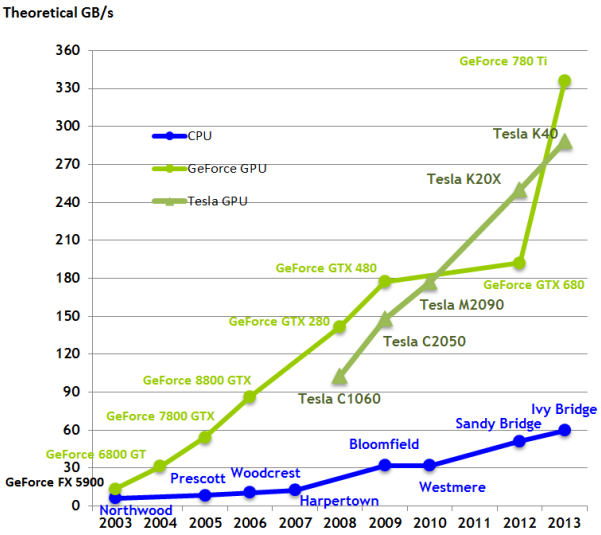
Table 1 "Thirteen Dwarfs" of parallel computing (Asanovic *et al.* 2006)

Dense linear algebra	Sparse linear algebra	Spectral methods (FFT)
N-body methods	Structured grids	Unstructured grids (FEM)
Monte Carlo methods	Combinational logic	Graph traversal
Dynamic programming	Backtrack and Branch+Bound	
Finite state machine	Construct graphic models	

The main reasons why the GPU architecture has attracted much interest, are better performance, anticipated evolution, reduced cost and energy efficiency (Gaurav and Wojtkiewicz 2011). For scientific computing more specifically, the GPU architecture is best suited for solving problems that can be divided into "smaller jobs", which can be computed in parallel. This means that the same program or routine can repeatedly be executed by using different data inputs and, consequently, generate different results. The current GPU architecture is always equipped with thousands of cores (ALU). For example, the NVIDIA® Tesla K40 has 2280 cores. As a result, its theoretical peak floating point performance and theoretical data transfer bandwidth would be much higher than the latest CPU architecture, as illustrated in Figs. 2(a) and 2(b).



(a) Floating-point operations per second



(b) Memory bandwidth

Fig. 2 Comparison of performance for the CPU and GPU, reproduced from Nvidia (2014)

Moreover, the GPU technology is also interesting in terms of cost efficiency and energy consumption. Comparison between the two main leading chip manufactures of CPU and GPU (Intel® and NVIDIA®, respectively) suggests that NVIDIA Tesla K40 GPU's specifications are 18.26 Gflop/s per watt and 1.03 Gflop/s per US dollar (Nvidia 2013b). In contrast, the specifications of Intel's Xeon E7-8893 v2 are 4.21 Gflop/s per watt and 0.095 Gflop/s per US dollar (Intel, 2014b).

Since GPUs are easily programmable and economically efficient, they have become a highly competitive alternative to CPU-based parallel computing (Gaurav and Wojtkiewicz 2011). For example, the GPU technology has been applied in many engineering disciplines, including computational fluid dynamics (Stantchev *et al.* 2009, Klöckner *et al.* 2009, Corrigan *et al.* 2011), computational structural mechanics (Krawezik and Poole, 2010; Georgescu *et al.*, 2013), finite element methods (Cecka *et al.* 2011, Dziekonski *et al*. 2013), molecular dynamics (Bauer *et al.* 2011), etc. Nevertheless, as a new computing technique, the GPU architecture still faces difficulties and challenges in four technological areas: (*i*) specialized tasks, (*ii*) difficulty in programming, (*iii*) bandwidth limitation and (*iv*) rapid evolution.

1.2 Motivation: use of GPU architecture for wind engineering computations

GPU is advantageous for programs that can be parallelized. For example, matrix addition and multiplication can be easily optimized for parallel computing because each element in the resultant matrix is independent of the computations for other elements. On the other hand, deriving the eigenvalues of a matrix is difficult for parallel computing since eigenvalues and eigenvectors are related to every element in the matrix.

In comparison with the programming languages for CPU architecture, the programming language for GPU is still premature. First, versatility of features in GPU programming is inferior to current major programming languages, such as *object oriented programming*, which is not supported on GPU. Second, many current scientific computing libraries are only compiled and optimized for CPU architecture (Foundation 2014). For instance, on MATLAB platform, only few functions are supported for GPU computing (Matworks 2013). Third, as the GPU architecture differs from CPU, some current computing algorithms are not available or need further evolution. Therefore, as a novel programming platform, GPU programming requires more effort to overcome these difficulties.

By exploiting GPU's thousands of cores, a much higher peak processing speed can be achieved. However, as with most GPU computing, the bottleneck is memory transfer bandwidth limitation.

Also, both GPU hardware and software technologies are in fast evolution. The side effect is that the programming may result in an incompatibility with other hardware platforms. The two current major programming tools are CUDA® (Compute Unified Device Architecture) and OpenCL®. The former is free but it is a proprietary software, which only works on NVIDIA® graphic chips. The latter is open source; it is supported by many companies and organizations (e.g., AMD, NVIDIA, Apple, Intel and IBM) and it also allows parallel programming on heterogeneous systems, which utilize many processor types (CPUs and GPUs).

Wind engineering, as an inter-disciplinary research field, involves structural dynamics, applied fluid mechanics, signal processing, stochastic analysis, data collection and reliability assessment. In particular, recent advances in the field of performance-based wind engineering for tall buildings, accounting for various sources of loading variability and estimation errors, require large computer resources (e.g., Smith and Caracoglia 2011, Spence and Gioffrè 2012, Barbato *et al.* 2013). Most

of these operations are suitable for parallel processing. In this paper, several case studies and algorithms, implemented for wind engineering, are demonstrated using both GPU and CPU architecture. Their computational speed is compared for various data sizes. The numerical computing examples are performed on CUDA® and MATLAB® since these two computing platforms are integrated. To the authors' knowledge, this is one of the first applications of this emerging computer technology in wind engineering. The aim of this study is to provide an efficient platform for computations in the context of performance-based wind engineering of tall buildings.

2. NVIDIA GPU architecture and CUDA platform

2.1 The NVIDIA GPU architecture

The NVIDIA® GPU architecture is composed of several streaming multi-processors (SMs), each having a number of streaming processor cores (SPs), which are also called CUDA® cores, on-chip memory, off-chip memory and one instruction unit. Each thread running on a single SP has its own local memory. This memory is often stored on-chip, but may be stored off-chip if the on-chip memory space has been already allocated. The shared memory, assigned to each block, can be accessed by many threads inside the block. Although each thread or SP has access to different memories (Fig. 3), the computing speed for on-chip and off-chip memory significantly differs. The off-chip global memory has a latency of 400 to 800 clock cycles. On the contrary, the on-chip register and shared memory's latency is typically 22 clock cycles in hardware devices of first and second generation, and about 11 clock cycles in hardware devices of third generation.

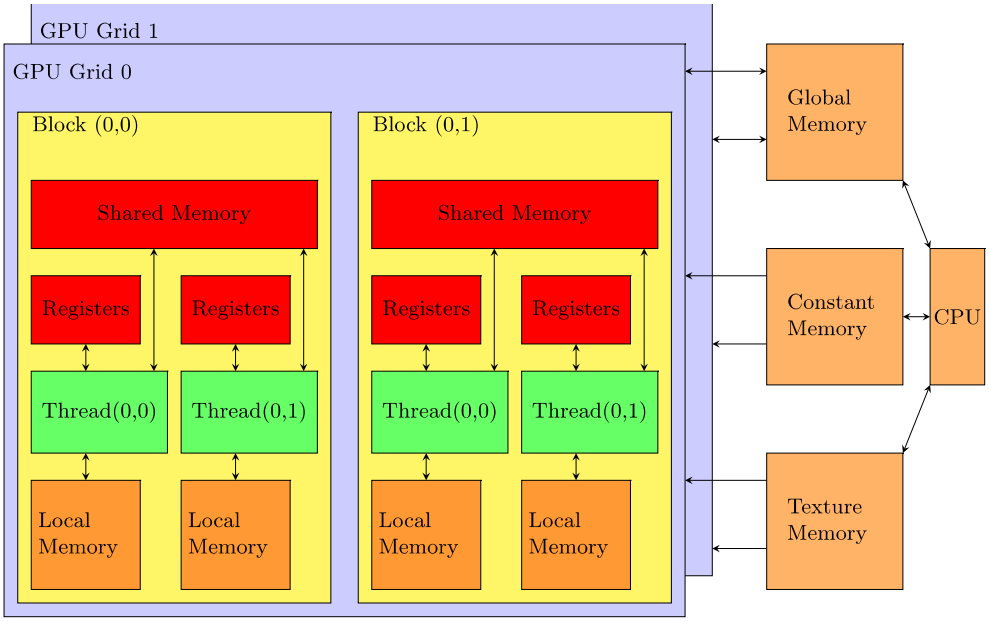


Fig. 3 Comparison of performance between CPU and GPU (Nvidia 2014)

The graphic card used in this article is the NVIDIA Tesla K20, which has 13 SMs consisting of 192 SPs each with a 706 MHz core clock. It has 5GB total board memory with a memory bandwidth of 208 GB/s. The CPU hardware information is composed of two Intel Xeon E5-2670, whose core clock is 2.60 GHz, and 128 GB RAM.

2.2 The CUDA platform

NVIDIA's CUDA parallel programming software provides a **C**/**C**++ language interface to control the hardware. Normally, a complete CUDA program consists of two parts: first there is a sequential *host* program running on CPU and, second, a *kernel* program controlled by GPU, which runs in parallel on massive GPU cores. The *host* program can use all **C**++ features, such as objective-oriented paradigm; on the contrary, the *kernel* programming methodology is exclusively restricted to **C** and the CUDA extension. A *kernel* is usually executed as Single Instruction Multiple Thread (SIMT) paradigm on a group of *threads*; this group is also called *block*. A *kernel* runs on a *grid* consisting of one or more *blocks*. Each *thread* has private but limited on-chip local memory. Different *threads* in the same *block* can communicate through on-chip Shared Memory. Finally, every *thread* has access to "read from" and "write to"' global off-chip memory. The whole CUDA memory hierarchical scheme is illustrated in Fig. 3. The Constant Memory and Texture Memory features are not used in this article.

3. Theoretical background: wind load and response analysis on tall slender systems

3.1 Along-wind spectrum of wind turbulence

Wind speed observation indicates that wind velocity varies very randomly with time; this randomness is due to the turbulence in the wind flow (Simiu and Scanlan 1996). One of most widely used turbulence one-sided power spectral density (PSD) expressions is Eq. (1), the Kaimal model

$$\frac{nS_{uu}(n,z)}{u_*^2} = \frac{200f}{(1+50f)^{5/3}} \tag{1}$$

in which $f = nz/\overline{U}(z)$ is the Monin coordinate, u_* is the friction velocity related to terrain roughness z_0; the mean wind speed $\overline{U}(z)$ varies with elevation due to boundary layer profile effects.

Performing time-domain structural analysis requires the synthetic generation of wind speed times series, unless real wind speed ambient records are available. A widely used method for generating time histories is described, for example, in Iannuzzi and Spinell (1987). This numerical method uses Fast Fourier Transform (FFT) to generate stationary random sequences that are compatible with a specified PSD function [Eq. (1)]. The basic idea is to generate a random complex number $Y = a\exp(\iota\phi)$, in which the modulus a is the square root of the PSD function (re-scaled), ϕ is a random phase (uniformly distributed between 0 and 2π) and $\iota^2 = -1$. The final step is performing inverse Fourier transformation on Y to obtain the desired realization of a random time series.

3.2 Monte Carlo simulation methods for performance-based wind engineering

The Monte Carlo method (e.g., Grigoriu 2002, Robert and Casella 2005) is a numerical algorithm, relying on the repeated random sampling to obtain numerical results for the analysis of stochastic systems, affected by various sources of uncertainty. One important application of the Monte Carlo method in mathematics is the Monte Carlo integration of a function $\mathrm{f}(\mathbf{x})$

$$I = \int_{\Omega} \mathrm{f}(\mathbf{x})\mathrm{d}\mathbf{x} \approx V\frac{1}{N}\sum_{i=1}^{N}\mathrm{f}(\mathbf{X}_i) \tag{2}$$

In the previous equation $\mathbf{X}_i$ is a realization of random data set, uniformly distributed on the domain Ω; V is the volume of Ω. Monte Carlo integration has been recently proposed for efficient numerical evaluation of the generalized modal wind loading for buffeting simulations on long-span bridges (Seo and Caracoglia 2012) and tall buildings (Smith and Caracoglia 2011).

One of the main benefits of this numerical method is the high computing speed compared to numerical quadrature, such as Newton-Cotes formulas (Faires and Burden 2012). If the size of the sample being evaluated is fixed, the computing time required by Newton-Cotes method will increase exponentially but the time used by Monte Carlo method is constant.

Another application of the Monte Carlo method is the "fragility analysis", employed to quantify structural reliability due to wind hazards. This method can account for various sources of uncertainty, including error-contaminated aerodynamic parameters. Unavoidable experimental errors in a wind tunnel test, used to determine the loads on the full-scale structure, are in fact a relevant uncertainty source for structural reliability (Smith and Caracoglia 2011, Seo and Caracoglia 2013, Bernardini *et al.* 2012).

Fragility analysis is based on the computation of the probability that a representative engineering parameter, corresponding to a specific feature of the dynamic response (maximum lateral drift, RMS acceleration, etc.) (Smith and Caracoglia 2011) can exceed a given limit-state threshold, conditional on the intensity of the hazard (i.e., reference mean wind speed and direction). More specifically, the Monte Carlo method involves the numerical evaluation of structural fragility (exceedance probability F_T) as

$$F_T \approx \frac{1}{N}\sum_{i=1}^{N}\delta(\mathbf{X}_i) \tag{3}$$

where $\mathbf{X}_i$ is the *i*-th realization of the random variable corresponding to a given engineering parameter, N is the total number of sets of random variables and $\delta(\mathbf{X}_i)$ is an "indicator function" which assumes the value $\delta = 1$ if the structural response exceeds the threshold and $\delta = 0$ otherwise.

In the Monte Carlo method, the evaluation of each random input is absolutely independent, such as with $f(\mathbf{X}_i)$ in Eq. (2) and $\delta(\mathbf{X}_i)$ in Eq. (3). Therefore, it is suitable to estimate each realization of the random input by parallelization, especially on a GPU architecture for many thousands of inputs. This paper will demonstrate this advantage by examining three numerical examples, employing the Monte Carlo method on GPU to evaluate fragility curves of two high-rise systems.

3.3 Single-degree-of-freedom high-rise system: description of the structure and aerodynamics

The first simplified example of a structure under wind loads is a single Degree-of-Freedom (DOF) structure, for example a vertical structure with a large mass at the top of the system (e.g., cantilever lighting system, advertising board, etc.) This system can be modeled as a single DOF using classical dynamics approach composed of a mass, spring and damping system, as shown in Fig. 4.

The dynamic equilibrium of the system in Fig. 4, for example illustrating the lateral vibration of the top node (Cui and Caracoglia 2015), can be written as

$$m\ddot{x} + c\dot{x} + kx = AC_d\overline{U}[u(t) - \dot{x}] \tag{4}$$

in which C_D is the drag coefficient, normalized to the "projected area" A of the wind load, $\overline{U}$ is the reference mean wind speed at the top of the structure, $u(t)$ is the time-dependent random turbulence and ρ is the air density. The power spectral density (PSD) of the structural response (displacement x at the top of the structure) can be estimated as (Simiu and Scanlan 1996)

$$S_{xx}(n) = \frac{[AC_D\overline{U}]^2 S_{uu}(n)}{16\pi^4 m^2\left\{\left(n_0^2-n^2\right)^2+[2nn_0(\zeta_s+\zeta_a)]^2\right\}} \tag{5}$$

in which n is a generic frequency, n_0 is the natural structural frequency, $S_{uu}(n)$ is the PSD of the along-wind turbulence [Eq. (1)], ζ_s is the structural damping ratio and ζ_a is the aero-dynamic damping. The standard derivation of the x response is the integral of the PSD (Simiu and Scanlan 1996)

$$\sigma_x = \sqrt{\int_0^\infty S_{xx}(n)\mathrm{d}n} \tag{6}$$

3.4 Continuum high-rise structure: description of the model and aerodynamics

In a continuum tall structure, if linear response is postulated or if the lateral deflection shape is assumed, an infinite DOF system can be simplified to a finite DOF generalized system. The tall building, illustrated in Fig. 5, is a typical continuum tall structure.

In this paper, the main objective is to demonstrate the advantage of GPU for wind engineering uncertainty analysis. Therefore, turbulent wind loading, which mainly consists of low-frequency quasi-steady forces, has been exclusively simulated.

Vortex-shedding effects, although very important for the structural dynamics, are not considered. In addition, for simplicity, only vibration occurring in the first fundamental mode [with shape $\Phi_i(z)$ and $i = x$ indicating along-wind direction] is used to analyze the building aerodynamics and dynamics. Even though the contribution of higher modes to the structural motion has been recognized (Kareem 1981, Melbourne and Cheung 1988, Feng *et al.* 2012, Aly 2013), it has not been included in the formulation. Under these assumptions, a continuum tall prismatic structure can be reduced to a single-DOF equivalent system in generalized coordinates, if the shape functions in the along-wind direction $i = x$ and cross-wind direction $i = y$ are planar, lying on the two orthogonal vertical principal planes of deflection and independent of each other.

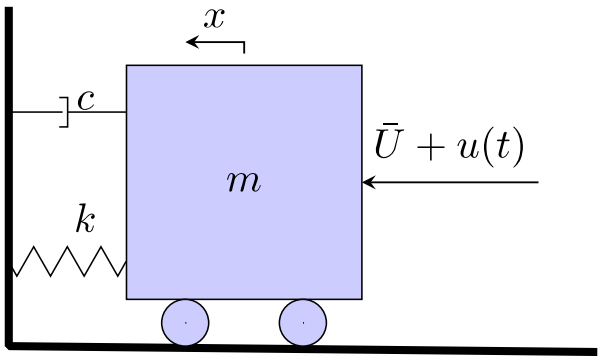
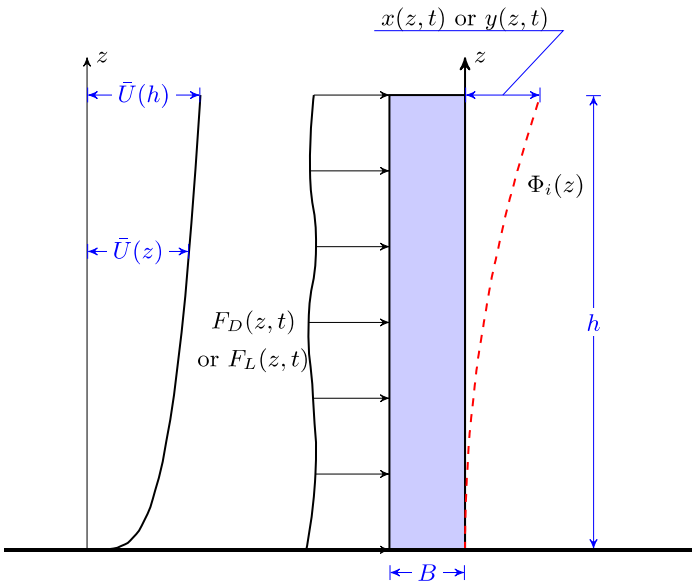
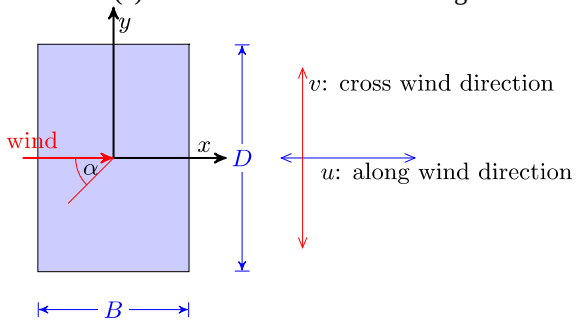


Fig. 4 Simplified single-DOF structure under wind load effect



(a) Lateral view of a tall building



(b) Horizontal-plane view with main wind load directions

Fig. 5 Schematics of a tall building under wind load effect (α: relative wind direction)

The PSD of the along-wind generalized force ($i = x$), taking into account the shape function and partial correlation of along-wind turbulence speed along the vertical coordinate z by means of the cross-PSD function $S_{uu}(z_1, z_2; n)$, is (Simiu and Scanlan 1996)

$$S_{Q_xQ_x}(n) = \iint_h C_D^2 \rho^2 D^2 \Phi_x(z_1)\Phi_x(z_2)\bar{U}(z_1)\bar{U}(z_2)\, S_{uu}(z_1, z_2; n)\mathrm{d}z_1\mathrm{d}z_2 \tag{7}$$

In the cross-wind direction ($i = y$), the generalized aerodynamic force (Simiu and Scanlan 1996; Piccardo and Solari 2000) is:

$$S_{Q_yQ_y}(n) = \iint_h \left[C_D^2 S_{vv}(z_1, z_2; n) + \left(0.25\,\partial C_L\,/\partial\alpha + C_D\right)^2 S_{uu}(z_1, z_2; n)\right] \times \rho^2 D^2 \Phi_y(z_1)\Phi_y(z_2)\bar{U}(z_1)\bar{U}(z_2)\mathrm{d}z_1\mathrm{d}z_2 \tag{8}$$

In the previous equation $S_{vv}(z_1, z_2; n)$ is the cross-PSD function of the cross-wind turbulence v. The drag C_D and lift C_L (transverse force) aerodynamic coefficients are evaluated per unit height and normalized with respect to dimension D in Fig. 5. The previous equations are derived for incident mean wind direction corresponding to the x direction in Fig. 5, as an example. These equations could be easily extended to a generic incident mean wind angle. The PSD of the generalized displacement ($i = x$ or $i = y$) and its standard derivation can also be derived, similar to Eqs. (5) and (6).

In some circumstances, the along-wind force and cross-wind force may affect each other, and the coupling effect should be considered into the analysis. For example, Smith (2009) provides a simplified simulation method to consider the coupling effect in a tall building, which neglects cross-correlation between turbulence velocity components u and v. For a more general formulation, the interested reader may refer to Piccardo and Solari (2000). In the simplified method, the cross-PSD matrix of the generalized response in the generalized coordinates ξ_x and ξ_y, $\mathbf{S}_{\xi\xi}(n)$, can be written as

$$\mathbf{S}_{\xi\xi}(n) = [\mathbf{E}(n)]^{-1}\begin{bmatrix} S_{Q_xQ_x}(n) & 0 \\ 0 & S_{Q_yQ_y}(n) \end{bmatrix}\{[\mathbf{E}(n)]^{-1}\}^{*,T} \tag{9a}$$

$$\mathbf{E}(n) = \begin{bmatrix} \bar{M}_x\left[\left(n_{0,x}^2 - n^2\right) + 2\imath n_{0,x} n\left(\zeta_{s,x} + \zeta_{a,x}\right)\right] & \imath n \int_0^h 2\rho\bar{U}(z) D\left(\frac{\partial C_L}{\partial\alpha} + C_D\right)\Phi_x(z)\Phi_y(z)\mathrm{d}z \\ \imath n \int_0^h 2\rho\bar{U}(z) D C_L \Phi_x(z)\Phi_y(z)\mathrm{d}z & \bar{M}_y\left[\left(n_{0,y}^2 - n^2\right) + 2\imath n_{0,y} n\left(\zeta_{s,y} + \zeta_{a,y}\right)\right] \end{bmatrix} \tag{9b}$$

In the previous equation $\bar{M}_x$ and $\bar{M}_y$ are generalized masses associated with mode shapes $\Phi_x(z)$ and $\Phi_y(z)$ respectively, $\zeta_{s,x}$, $\zeta_{a,x}$, $\zeta_{s,y}$ and $\zeta_{a,y}$ are generalized structural and aerodynamic damping ratios in the along-wind and cross-wind directions, $\bar{U}(z)$ is the mean wind speed at the elevation z due to boundary layer effects (Simiu and Scanlan 1996). The symbol $\{\cdot\}^{*,T}$ denotes complex conjugate transpose operator.

4. Parallelization paradigms in numerical computing

One of the difficulties of parallel computing design is how to distribute sub-tasks among various computing units (Kepner 2009). The two most popular forms of parallel computing are data-level parallelization and task-level parallelization (Culler *et al.* 1999). In data parallelization the same calculation procedure is performed on the different data sets. In contrast, the parallelization tasks perform entirely different calculation procedures on the same data set.

For parallel computing in wind engineering, the parallelization paradigms are more flexible and depend on the size of the parallel system. If the size of the system is small, for example the current Intel CPU with 4 to 12 cores and a computer cluster with several nodes, the structural response can be evaluated for different wind loads (mean wind velocities) concurrently on different cores or nodes. If the size of the system is very large, the sampling-based method, for example the Monte Carlo method described in previous sections, can be parallelized on the GPU with many cores and a supercomputer. Even more, the numerical computing procedure itself can be parallelized (Gaurav and Wojtkiewicz 2011). For example, in a matrix multiplication, the evaluation of each element is independent and can be easily parallelized. Other numerical analysis applications, such as FFT and numerical integration, can also make use of this parallelization. These two types of data parallelization paradigms will be considered in this study: small-scale data parallelization and large-scale data parallelization.

Besides data parallelization, the task parallelization strategy can also be employed and applied to the study of wind effects on structures, and even for other structural loads. For example, the wind effect in the along-wind direction, Eq. (7), and in the cross-wind direction, Eq. (8), can be evaluated simultaneously and combined later to obtain the resultant response. Theoretically, the entire structural analysis, analysis of dead loads, live loads, seismic loads and wind loads could be performed in parallel.

5. Numerical examples

5.1 GPU speed-up for basic mathematical computations

In the field of numerical simulation, complex algorithm and simulation methods are composed of either sequential or parallel basic mathematical operations, such as linear algebra, random number generation and numerical integration. The application of these basic calculations is an effective demonstration of the advantage of GPU computing.

Fig. 6 compares the computing time of four basic mathematical operations: dense matrix algebra (addition, multiplication, division) and FFT. Various computational sizes (number of operations to be performed) are considered. All the operations are carried out on MATLAB platform by both CPU and GPU. All the input data for matrix algebra and FFT are randomly generated. Machine times, needed for number generation and data transfer, are not included in the computing time. The GPU speed-up depends on data type, the size of processed data and the type of operation. For matrix multiplication with dimension 8192 by 8192 and single float data type, the computation time on the CPU is 3.34 s but computation time on the GPU is only 0.469 s, which is 7.2 times faster than CPU. For double float data type, the difference is similar to single float data type.

However, for matrix division, the GPU computing time is not as small as in the two previous examples. When the matrix size is 2^{13}, the GPU is only 4.34 times faster than the CPU. The reason is that matrix addition and multiplication are feasible for parallelization since the calculation for each element is independent. On the contrary, for matrix division using the current numerical methods (such as Gauss-Jordan Elimination or Gauss-Seidel Iteration), the computation result of the elements is inter-dependent. Therefore, most numerical operations must be sequential. For the double precision float data, recommended to ensure accuracy in the computation, the time difference is very small compared to single-precision float data.

Besides the comparison using elapsed time, another way to evaluate the efficiency of GPU computing vs. CPU is the computing performance based on hardware theoretical peak performance. From the Intel's specification (Intel, 2014a), the Xeon E5 2670 has maximum performance at 166.4 Gflop/s and maximum memory bandwidth 51.2 GB/s. For GPU Nvidia Tesla K20 (Nvidia, 2013a), the maximum performance is 3.52 Tflop/s for single precision floating point, 1.17 Tflop/s for double precision floating point and 208 GB/s memory bandwidth.

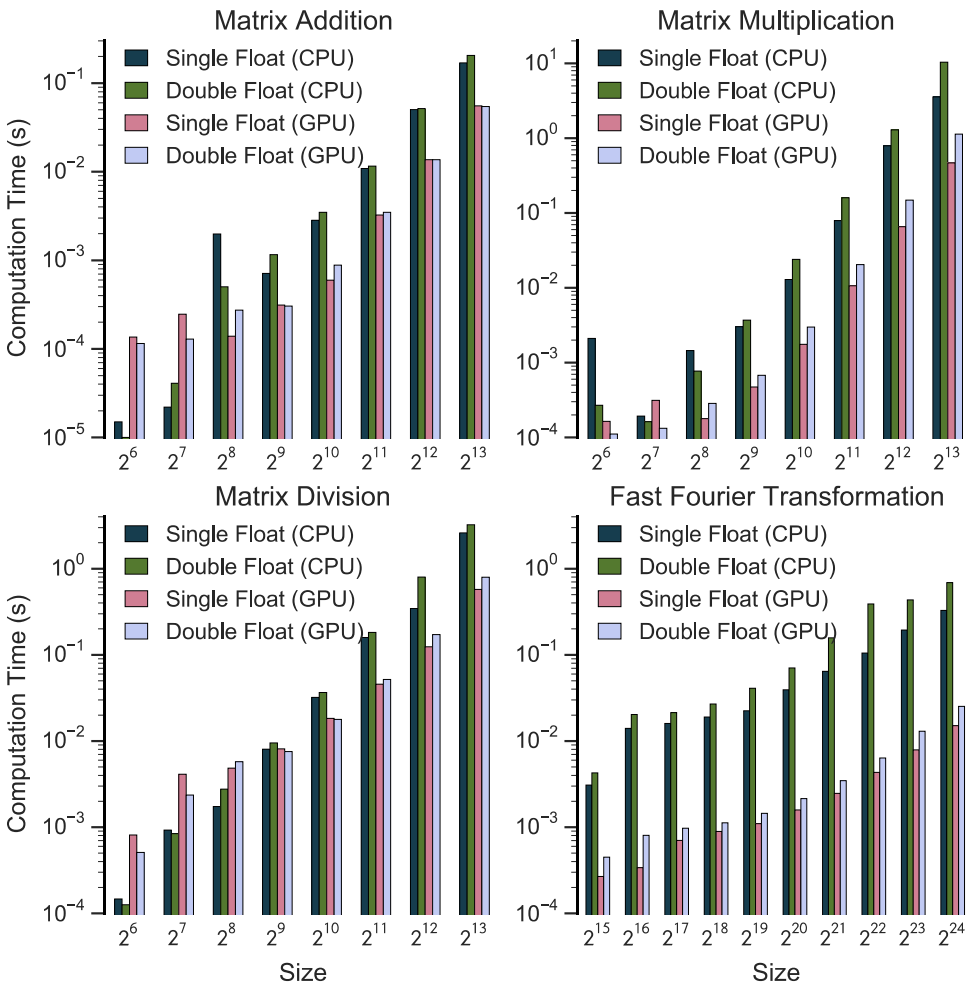


Fig. 6 Basic mathematics: computing time comparison between CPU and GPU

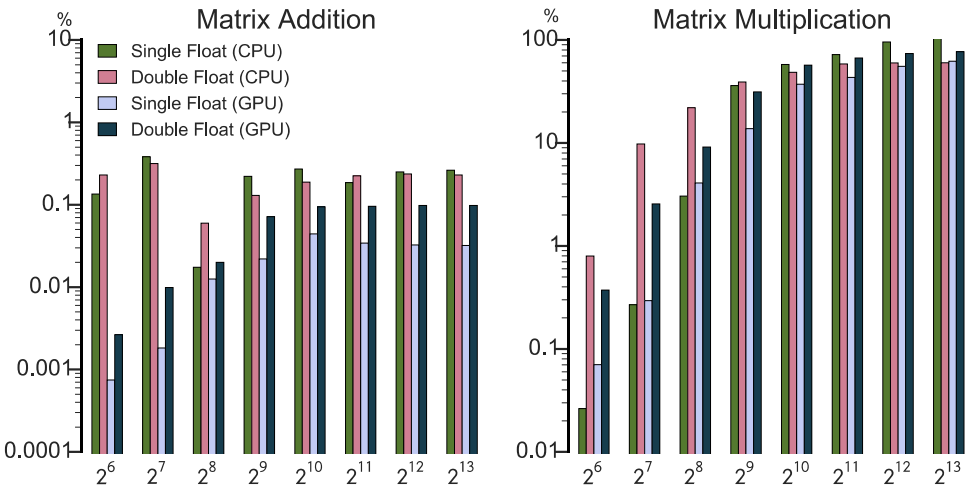


Fig. 7 Gflop/s in matrix operation compared to hardware peak performance

For the matrix addition, both CPU and GPU cannot make full use of their power on MATLAB platform, especially for GPU with only approximately 105 Gflop/s performance, which is only 0.3% of the hardware peak performance. However, for matrix multiplication, about 90% hardware peak performance can be achieved with both CPU and GPU when the matrix size is beyond 1024 (2^{10}) as in Fig. 7.

Therefore, it is plausible to conclude that both multiplication algorithms are well tuned and optimized for both CPU and GPU. Since the base GPU hardware performance is superior to CPU, computing speed on GPU is faster than CPU especially for a large size matrix.

FFT is another basic numerical procedure, widely used in signal processing, image processing and Fourier analysis in wind engineering. More importantly, the CUDA platform provides an FFT–GPU library (cuFFT). For FFT, the computational efficiency gain is 26.34 times, if GPU is compared to CPU when the data size is 2^{24}. In any case, it must be noted that in all the four operations analyzed, when data size or problem dimension is small, for example $2^6 = 64$, the GPU computation does not exhibit a reduction in time, or it is even slower than the CPU.

5.2 GPU speed-up for advanced mathematical computations

Commonly, the complete solution to a numerical problem consists of more than one sequential mathematical operation; therefore, the GPU computing efficiency for an advanced mathematical calculation depends on the slowest operation in the whole process (Liebig's barrel law). This section uses two examples to illustrate this phenomenon.

The first example shown in Fig. 8 is the application of Monte-Carlo integration of a function with singularity (but still integrable). The function is

$$I = \int_0^3 \frac{x^2}{\sqrt{\tan|0.5\pi x|}} \mathrm{d}x \tag{10}$$

This Monte-Carlo integration procedure was run on MATLAB. The random

uniformly-distributed points $0 \leq x_i \leq 3$ have been generated directly on the GPU global memory and the evaluation of $f(x_i)$ (with x_i being the *i*-th realization of the random set) has been run in parallel on various threads on the GPU. Finally, the summation operation (average) of all $f(x_i)$ is carried out using a binary summation algorithm, illustrated in Fig. 9.

Fig. 8 shows the computing time comparison between GPU and CPU with different float type. When the size of the sample xi is 221 and with double-precision float number, the computing time spent on the CPU is about 40 s and the time on GPU is only 0.002 s, which is approximately 20000 times faster. This results, which may appear surprising from the point of view of computer performance, could also be related to the software that is used for the comparisons. Even though the source codes (scripts) are almost identical, it is possible that the software platform may process information differently. Investigation on this particular result would require full access to the software core, which is not available. Therefore, more investigation, beyond the scope of this study, is needed in future research.

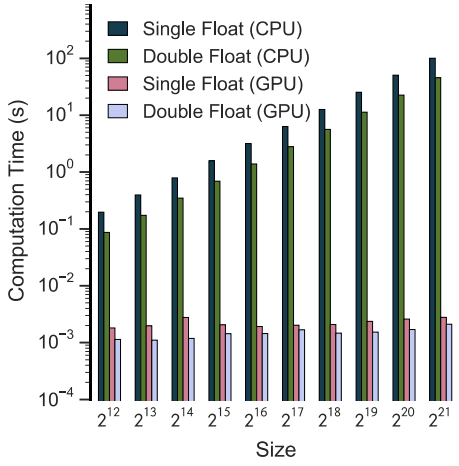


Fig. 8 Monte Carlo integral computation: time comparison between CPU and GPU platforms

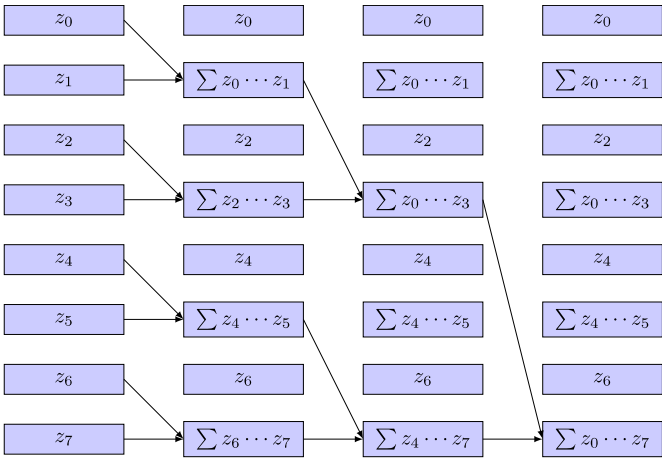
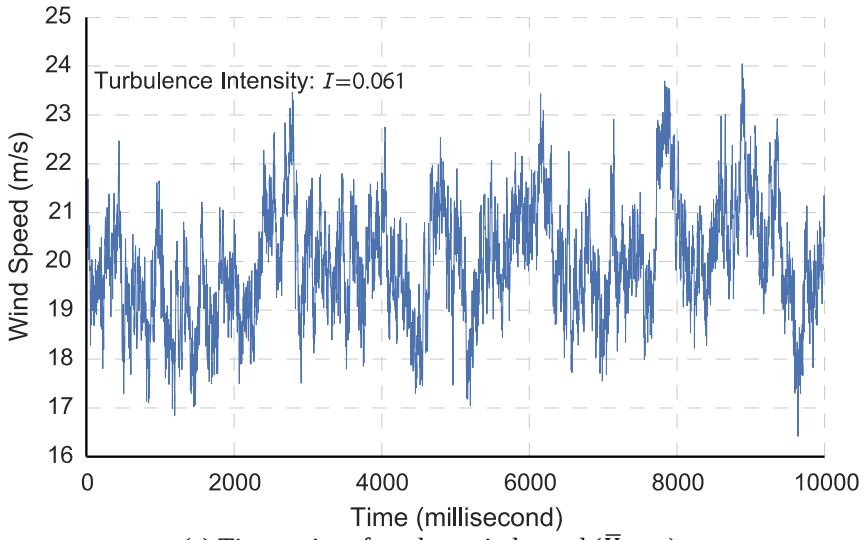


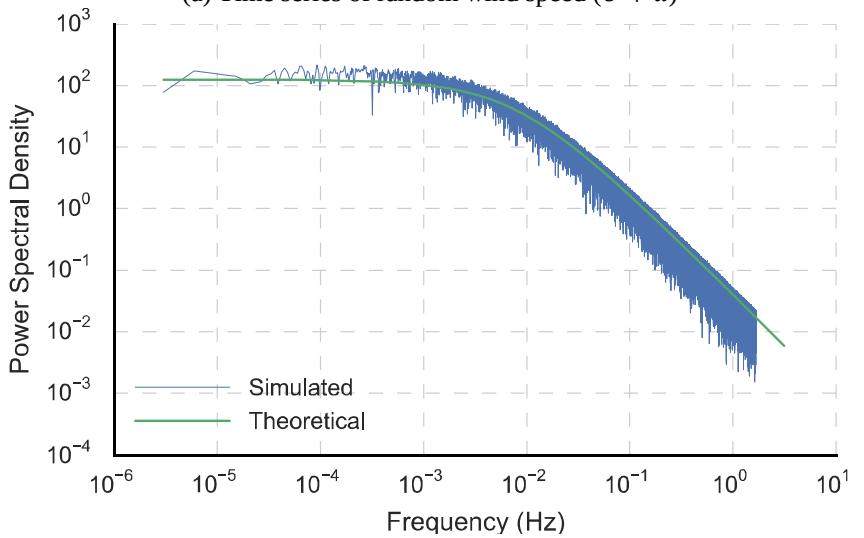
Fig. 9 Array summation algorithm on GPU platform

Random wind speed in time domain is also another application (Huang *et al*. 2010). The procedure is based on the synthetic generation of random wind speed records, according to a pre-selected PSD model and boundary layer profile. The main steps for simulating the random turbulence component u are derived from Iannuzzi and Spinelli (1987); they can be summarized as follows:

1) Computation of PSD $S_{uu}(n_i)$ at various equally-spaced frequencies n_i, with step Δn
2) Generation of independent random virtual phase realizations ϕ_i between 0 and 2π
3) Evaluation of $\left(\sqrt{S_{uu}(n_i)\Delta n}\right)\exp(\iota\phi_i)$,
4) Inverse FFT of all $\left(\sqrt{S_{uu}(n_i)\Delta n}\right)\exp(\iota\phi_i)$.



(a) Time series of random wind speed ($\bar{U} + u$)



(b) PSD of random turbulence u

Fig. 10 Synthetic generation of random wind speed ($\bar{U} + u$)

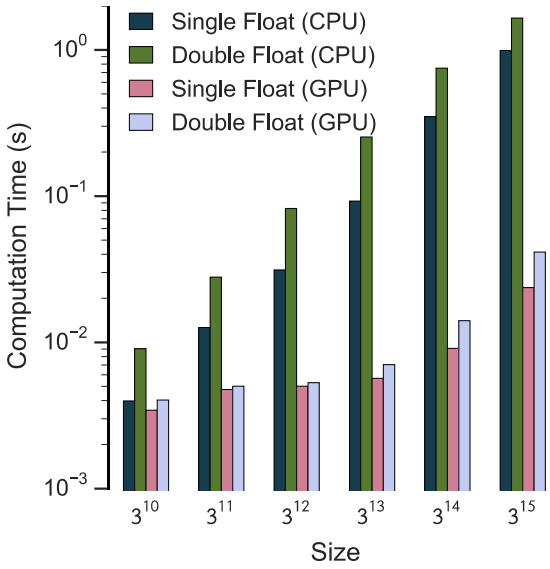


Fig. 11 Random wind speed time series generation: computing time comparison between CPU and GPU platforms

The step-by-step procedure, described above, is an approximate method for generating wind turbulence fluctuations, which is used to test the speed-up effect in FFT calculations by GPU. More accurate methods for the generation of turbulence time histories are, for example, discussed in Iannuzzi and Spinelli (1987).

An example of simulated wind speed record ($\overline{U}+u$, with mean speed $\overline{U}=20\,\mathrm{m/s}$ and random turbulence u) in both time domain and frequency domain is presented in Fig. 10.

From step 1 to step 4, the procedure can be fully parallelized. Nevertheless, the inverse FFT in the last step combines both a parallel process and a sequential process. This operation is much slower than the previous steps even though the cuFFT library has been optimized. Fig. 11 illustrates the computing time of the random wind speed time series generation on both CPU and GPU. When the generation size is 3^{15}, the speed-up of the GPU is about 40 times. The speed-up effect is considerable but it may also be affected by the choice of the MATLAB software platform. Even though similar speed-ups have been reported in the literature for similar problems in stochastic mechanics (Gaurav and Wojtkiewicz 2011), this results requires careful consideration and further investigation.

The comparison between the two examples above demonstrates that computing efficiency of advanced mathematical procedures is determined by the slowest operation, such as step 4 in the second case, which is similar to the "time complexity theory" (Sipser 2012) in computer science.

5.3 Fragility analysis of a single-DOF structure induced by uncertain wind load effect

The modeling of wind load effects on a single-DOF structure has been illustrated in Section 3.3; the governing equation for frequency domain analysis is Eq. (5). As also outlined in the previous sections, construction variability and experimental-error contamination may induce

uncertainty in the structural response PSD and variance. As a result, fragility analysis in Eq. (3) of the structure (probability of "failure") can be used to quantify the uncertainty.

In this section, a simple high-rise vertical "point-like" structure, i.e., a large advertising board (Fig. 12), is used as an example to demonstrate the numerical analysis process.

As shown in Fig. 12, the advertising board can be approximately modeled by splitting the system in two parts: board and support column (tower or mast). The mean wind direction is orthogonal to the surface of the board. Because most wind force is carried by the board and flexibility is concentrated in the support, the dynamics of the structure can be simplified as a single-DOF generalized structure as in Fig. 4. The structural properties and parameters of the equivalent SDOF system are summarized in Table 2. In Eq. (5), C_D can be determined indirectly by wind tunnel test or by CFD simulation.

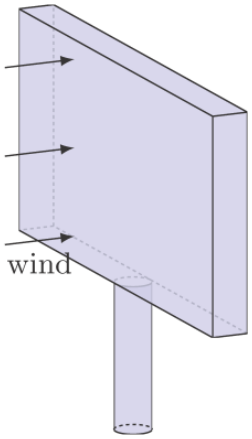


Fig. 12 Schematics of a large advertising board

Table 2 Main properties of the simulated SDOF structure – advertising board

Quantity	Value assigned
Area (A)	8 m^2
Elevation of board's centroid (h)	30 m
Mass (m)	20 kg
Stiffness (k)	1 kN/m
Structural damping ratio (ζ_s)	0.005
Air density (ρ)	1.0 kg/m^3
Drag coeff. (C_D, average value)	1.54
Terrain roughness (z_0)	0.5 m

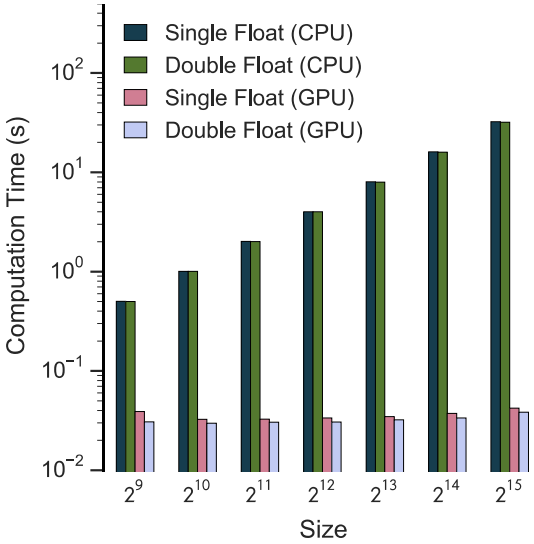


Fig. 13 Computing time gains in the fragility analysis of a single-DOF structure, induced by uncertain aerodynamic drag

Errors can therefore arise from measurement fidelity, variable laboratory and environmental conditions, test procedures, etc. The relevance of uncertainty, involved in Eq. (5), can be analyzed by fragility analysis in Eq. (3). The (scalar) random number x in Eq. (3) is the uncertain aerodynamic drag coefficient C_D. In this simulation C_D is assumed to be a Gamma random variable with mean of 1.54 and standard deviation 0.5 (Smith and Caracoglia 2011). As a consequence, the standard deviation of the along-wind dynamic response (σ_x, Root-Mean-Square or RMS) also becomes a random variable. Hence, the evaluation of σ_x and its variability can be carried out in a Monte-Carlo setting for the various uncertain parameters C_D. This operation can easily be distributed and performed on different GPU *threads* simultaneously. The computing speed can be greatly improved if a large set of uncertain parameters, such as C_D, is needed.

For demonstrating the efficiency of the GPU platform, the fragility analysis results are exclusively compared for wind mean velocity at the top of the structure $\overline{U} = \overline{U}(h) = 30$ m/s, measured at the centroid of the board, using both double-precision float number and single-precision float number. The same comparison will also be used in the last two numerical examples. The speed gains for different sample size is demonstrated in Fig. 13. The computing time on GPU with sample size from 2^9 to 2^{15} is roughly constant and equal to around 0.03 s. Most time is still spent by the algorithm to transfer data between GPU and CPU memory. The relative duration of the actual computations on GPU is relatively small. For computing time on CPU, on the contrary, the time increases linearly with the sample size. When the size of the input sequence is 32768, the CPU computing time on the selected software platform (MATLAB) is 33.6 s, whereas the GPU computing time is only 0.03 s. Both double float and single float computing times are very similar for the same input sample size. It is believed that this good performance may be influenced by the choice of the software platform and that, if a different platform is used or the problem to be solved becomes larger, the "scale" effect could significantly reduce the

speed-up. A reduction of speed-up effect has in fact been observed in the following case study, which is presented in the next sub-section.

It must be noted that, if the probability to be sought is of the order of 10^{-3}, at least 11000 samples would be needed to ensure that the coefficient of variation of the estimation error in Eq. (3) is less than or equal to 30%. If the two above procedures are repeated at various speeds from 5 m/s to 60 m/s, the fragility curve against unacceptable sway of the advertising board can be constructed, with the threshold defined as $\sigma_x = 0.5\,m$ at $z = h$. The fragility curve is shown in Fig. 14. The two curves in Fig. 14 are quantitatively very similar. This observation confirms that the GPU computing is fast and accurate compared to the reference CPU calculations.

5.4 Dynamic fragility analysis of a continuum high-rise building structure in the along-wind direction, induced by random wind load effect

For the continuum tall structure, such as the tall building shown in Fig. 15, a single-DOF simplified model is not adequate to simulate the structural response because the mean wind speed increases with the elevation z in the boundary layer (Simiu and Scanlan 1996). In addition, the correlation of wind turbulence at various z should be included into the simulation. In its simplest form, the response of a slender tall building with mean wind orthogonal to the face of dimension *D* in Fig. 15 can be simplified to a single-DOF model in generalized coordinates, using the prescribed shape function $\Phi_x(z)$ in Eq. (7). Therefore, Eq. (7), which requires a double integration to derive the generalized wind load, is used to compute the along-wind response of the structure. The evaluation of this double integral is usually the "bottleneck" of a dynamic buffeting analysis (Smith and Caracoglia 2011, Seo and Caracoglia 2012), since it must be repeated for all frequencies n. Therefore, it requires a large number of numerical computations.

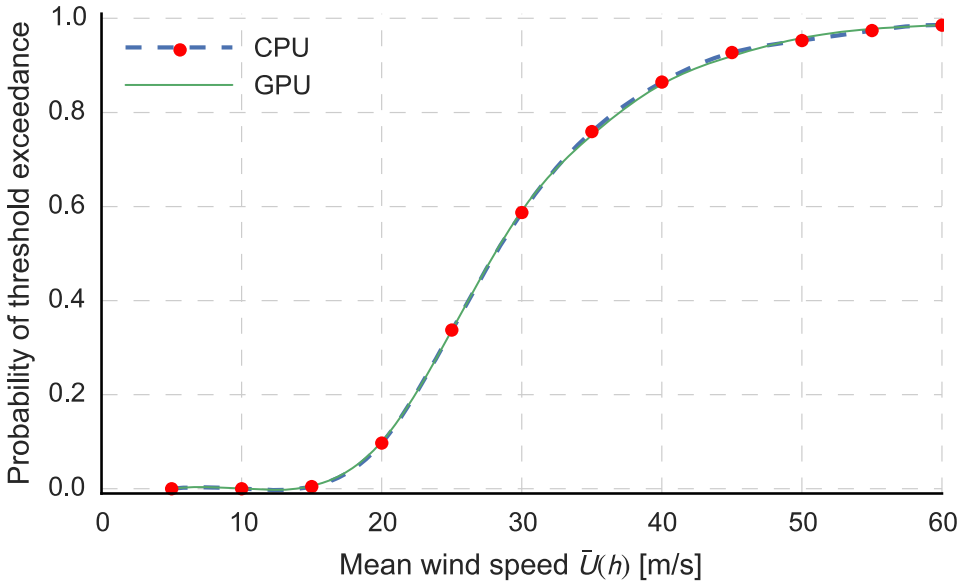


Fig. 14 Fragility curve for advertising board model in Fig. 12 (RMS vibration)

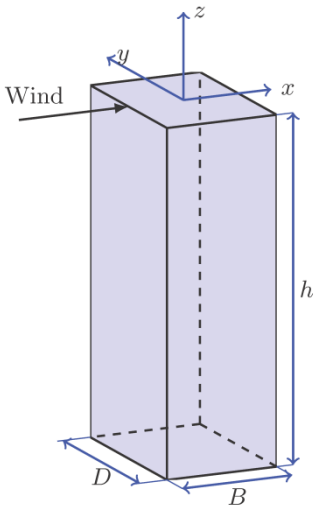


Fig. 15 Schematic view of the CAARC tall building

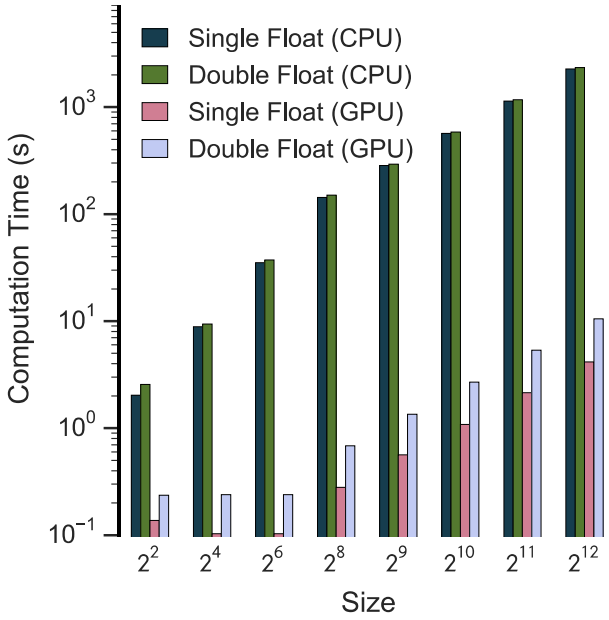


Fig. 16 Computing time gains in the fragility analysis of the CAARC tall building, examining the generalized response in the along-wind direction

In this paper, the evaluation of the double integral is carried out by "composite" Simpson's rule with 100 by 100 integration points. The same fragility analysis, employing the Commonwealth Advisory Aeronautical Research Council (CAARC) benchmark tall building (Melbourne 1980) in Fig. 15 is performed on both CPU and GPU platforms for comparison purposes.

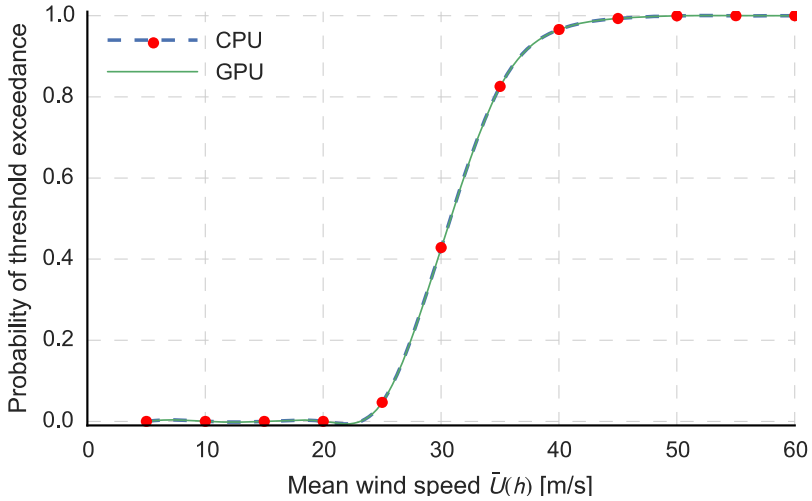


Fig. 17 Fragility curve of the CAARC tall building, corresponding to the threshold σ_x (h) = 0.5 m for along-wind response (RMS vibration at the roof-top)

Table 3 Structural and wind filed parameters of the CAARC tall building (Cui and Caracoglia 2015)

Quantity	Value assigned
Air density (ρ)	1.25 kg/m^3
Drag coeff. (C_D, per unit height)	1.54
Lift coeff. (C_L, per unit height)	0
$\partial C_L / \partial \alpha$ (per unit height)	-3.50
Terrain roughness (z_0)	0.5 m
Turbulence decay coeff. (C_{zu})	12
Turbulence decay coeff. (C_{zv})	$0.667C_{zv}$
Wind speed profile [$\bar{U}(z)$, m/s]	$\bar{U}(h)(z/z_0)^{0.28}$
Roof-top mean wind speed [$\bar{U}(h)$]	5 – 60 m/s
Roughness velocity (u_*)	$\bar{U}(h)/2.5\ln(h/z_0)$

The main structural and aerodynamic parameters of the CAARC building model are listed in Table 3; other parameters needed in the simulations can be found in Cui and Caracoglia (2015) and Melbourne (1980). Fig. 16 shows the comparison between CPU and GPU computing time with random aerodynamic force parameter C_D. In this example the estimation of the dynamic wind load is based, for illustration purposes, on the mean wind speed at the roof-top $\bar{U}(h) = 30\ m/s$. The computing time gain with GPU is approximately the same as the one observed for the single-DOF structure in Fig. 13. This result is due to the fact that both examples employ the same

parallel computing algorithm. When the random sample size is 4096 (2^{12}), the GPU is roughly 100 times faster than the CPU. Similar to the SDOF structure in the previous section, fragility analysis is performed for $\overline{U}(h)$ from 5 m/s to 60 m/s, when the RMS lateral-sway serviceability threshold at the roof-top is defined as $\sigma_x(h) = 0.15\%B = 0.05$ m (Smith 2009). The results are illustrated in Fig. 17. As before, the GPU results are very accurate.

5.5 Dynamic fragility analysis of a continuum high-rise building structure in both along-wind and cross-wind directions, induced by random wind load effect

Apart from the effect of the wind force in the along-wind direction, the cross-wind load is also considerable on tall buildings. As the last case study in this paper, the GPU parallel computing algorithm is applied to the analysis of the CAARC tall building affected by uncertainty in aerodynamic static drag coefficient C_D, already discussed in the previous sub-section.

The two-mode generalized model, described in Eq. (9), provides a simple simulation method to compute the wind load and the dynamic response of the CAARC tall building in both along-wind and cross-wind directions, by including the coupling between the fundamental building modes in the two primary vibration planes. The mean wind direction is again coincident with the orientation of the x axis in Fig. 15.

In the case of the coupled-mode generalized dynamic model, the parallelization paradigm has two levels (or stages). The first stage consists in distributing the evaluation of the random parameters for fragility analysis in GPU *blocks*. In the second stage the along-wind generalized force and crosswind generalized force are separately computed in two GPU *threads* for each GPU *block*, previously allocated. This parallelization paradigm involves both task parallelization and data parallelization. It can therefore be labeled as a *hybrid parallelization strategy*. This strategy is demonstrated in Fig. 18.

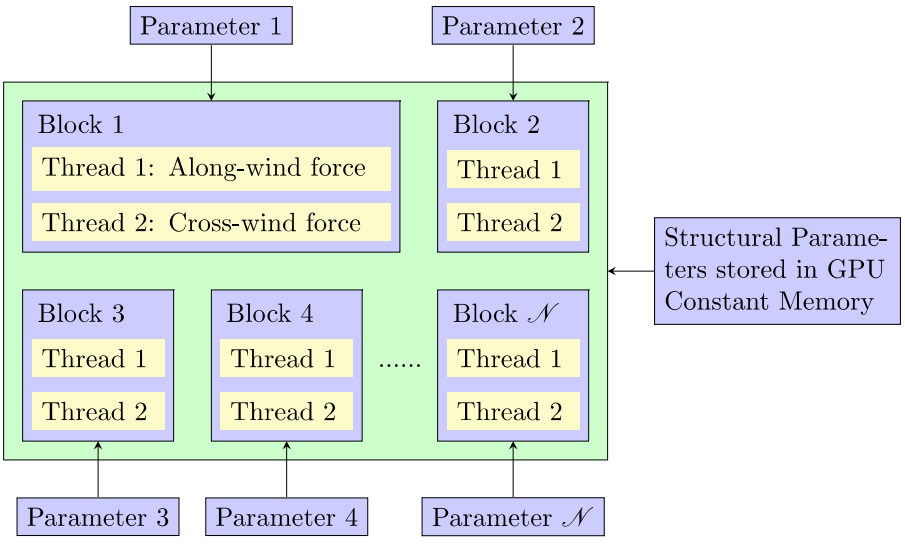


Fig. 18 Flow-chart of the two-level parallelization paradigm for dynamic analysis of a continuum high-rise structure, subjected to both along-wind and cross-wind loads

The computation of the generalized-force PSD functions, $S_{Q_xQ_x}(n)$ and $S_{Q_yQ_y}(n)$, still employs the composite Simpson's rule. The evaluation of the complex matrices $\mathbf{E}(n)$ and $\mathbf{S}_{\xi\xi}(n)$ uses "cuComplex" library on GPU and C++ "complex" library on CPU. The computing time for various sample sizes of the random variable C_D is presented in Fig. 19. Unlike the previous example, this last case study is much slower on the GPU because the parallel computing algorithm is more articulated and data transfer is more frequent. Nevertheless, the GPU speed is still roughly 10 times larger than the CPU time.

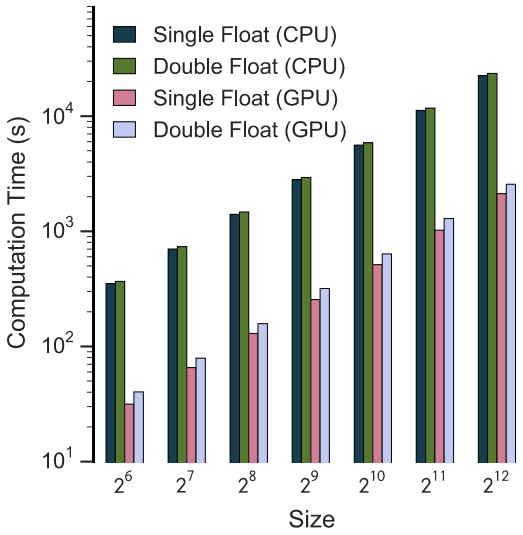


Fig. 19 Computing time gains in the fragility analysis of the CAARC building, examining two-direction coupled modal response

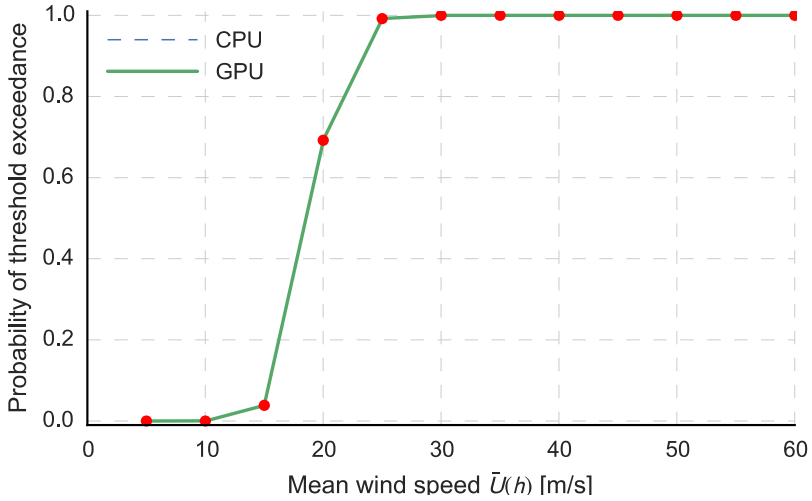


Fig. 20 Fragility curve of the CAARC tall building, corresponding to the threshold σ_x (h) = 0.5 m for coupled along-wind and cross-wind response (RMS vibration at the roof-top)

The complete fragility analysis, performed from $\overline{U}(h) = 5$ m/s to $\overline{U}(h)$=60 m/s and using the same threshold $\sigma_x(h) = 0.05$ m as before, is shown in Fig. 20. In this example, the fragility curve "translates leftward" and becomes "sharper". When the coupling effect between along-wind and cross-wind force is examined, the along-wind dynamic effect $\sigma_x(h)$ becomes more dangerous compared to the previous case, in which along-wind force and response are exclusively considered. Therefore, lower wind speed $\overline{U}(h)$ has larger probability of exceedance values when dynamic coupling is considered. A more detailed investigation on fragility for the CAARC tall building may be found in Cui and Caracoglia (2015).

6. Conclusions

Graphics Processing Unit (GPU) is an efficient massive parallel computing platform. In comparison with the CPU platform GPU is a highly competitive alternative architecture in terms of computing time, energy consumption and cost. As a result, applications of the GPU technology in various scientific fields have been rapidly increasing in recent years.

This paper explored, for the first time in wind engineering, the use of the GPU platform for simulating the dynamic response of high-rise systems. The study began by analyzing a series of simplified basic mathematical tasks on GPU architecture (e.g., matrix addition, multiplication, Monte-Carlo sampling, etc.). The same operations were later combined and employed in the subsequent implementation of parallel algorithms for wind engineering computations.

Subsequently, the GPU methodology was applied to the synthetic generation of wind turbulence. In this case the study concluded that, since the computing procedure is more articulated, sequential and complex, the overall speed of the GPU technology decreases and it is influenced by the slowest step in the sequential algorithm.

Next, the GPU technology was examined in the context of structural uncertainty and fragility analysis of high-rise systems under wind loads. Since the fragility analysis employs Monte Carlo sampling, it is advantageous to distribute the evaluation of the random inputs among different GPU threads and to estimate the response results by parallel computations.

The first example illustrated the fragility analysis of a single degree-of-freedom structure. The second case investigated a continuum high-rise structure, in which double integration is needed to compute the generalized buffeting force for structural dynamic response in the frequency domain. In both cases the GPU algorithm distributes the evaluation of the various random parameters among different computing threads. In the second example the computing time is larger, since the double integral requires more computing resources. However, the GPU technology is still advantageous in comparison with CPU.

The third example examined the simulation of wind force and modal coupling effects on a high-rise structure in both along-wind and cross-wind directions. The study proposed a two-layer distribution strategy on the GPU platform. This is believed to be a novel programming application of the GPU technology in wind engineering. From the comparison between the computing time on CPU and GPU, it was concluded that the GPU can perform distributed computing tasks faster than the CPU, on average 10 times faster. In any case, the GPU performance also depends on algorithm design and memory control strategies.

Finally, it must be noted that the simulations of the dynamic response were based on the quasi-steady formulation of the wind forces. This approach is often not suitable in the case of high-rise buildings since the cross-wind response may be influenced by wake-induced loading.

Nevertheless the main purpose of this study was the feasibility of GPU architecture for wind engineering computations; therefore, a more accurate simulation of the wind load was not considered necessary for the scope of the work. The interested reader may however refer to a recent study (Cui and Caracoglia 2015), in which it is shown that wake-induced and vortex-shedding effects can be readily included in the numerical formulation for fragility analysis with minimum additional effort. It is however anticipated that a similar performance gain would be obtained if vortex-shedding loads were included in the simulations.

Acknowledgements

This material is based upon work supported in part by the National Science Foundation (NSF) of the United States under CAREER Award CMMI-0844977 in 2009 - 2014. The second author would also like to acknowledge the support of the University of Trento, Italy, Research Fellowship Program, during his sabbatical leave in 2014. Numerical computations were carried out on the "Discovery Cluster" of Northeastern University. Any opinions, findings and conclusions or recommendations are those of the authors and do not necessarily reflect the views of either the NSF or any other institutions.

References

Aly, A.M. (2013), "Pressure integration technique for predicting wind-induced response in high-rise buildings", *Alexandria Eng. J.*, **52**(4), 717-731.

Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W. *et al*. (2006), *The landscape of parallel computing research: A view from Berkeley*, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, California, USA.

Barbato, M., Petrini, F., Unnikrishnan, V.U. and Ciampoli, M. (2013), "Performance-Based Hurricane Engineering (PBHE) framework", *Struct. Saf.*, **45**, 24-35.

Bauer, B.A., Davis, J.E., Taufer, M. and Patel, S. (2011), "Molecular dynamics simulations of aqueous ions at the liquid–vapor interface accelerated using graphics processors", *J. Comput. Chem.*, **32**(3), 375-385.

Bernardini, E., Spence, S.M. and Gioffrè, M. (2012), "Effects of the aerodynamic uncertainties in HFFB loading schemes on the response of tall buildings with coupled dynamic modes", *Eng. Struct.*, **42**, 329-341.

Cecka, C., Lew, A.J. and Darve, E. (2011), "Assembly of finite element methods on graphics processors", *Int. J. Numer. Meth. Eng.*, **85**(5), 640-669.

Colella, P. (2004), "Defining software requirements for scientific computing", Slide of 2004 presentation included in David Patterson's 2005 talk.

Corrigan, A., Camelli, F.F., Löhner, R. and Wallin, J. (2011), "Running unstructured grid-based CFD solvers on modern graphics hardware", *Int. J. Numer. Meth. Fl.*, **66**(2), 221-229.

Cui, W. and Caracoglia, L. (2015), "Simulation and analysis of intervention costs due to wind-induced damage on tall buildings", *Eng. Struct.*, **87**, 183-197.

Culler, D.E., Singh, J.P. and Gupta, A. (1999), *Parallel computer architecture: a hardware/software approach*, Gulf Professional Publishing, Houston, Texas, USA.

Dziekonski, A., Sypek, P., Lamecki, A. and Mrozowski, M. (2013), "Generation of large finite-element matrices on multiple graphics processors", *Int. J. Numer. Meth. Eng.*, **94**(2), 204-220.

Faires, D. and Burden, R. (2012), *Numerical Methods*, Cengage Learning, Boston, MA, USA.

Feng, R., Yan, G. and Ge, J. (2012), "Effects of high modes on the wind-induced response of super high-rise buildings", *Earthq. Eng. Eng. Vib.*, **11**(3), 427-434.
Foundation, F.S. (2014), GNU Scientific Library, URL: http://www.gnu.org/ software/gsl/
Gaurav, and Wojtkiewicz, S.F. (2011), "Use of GPU computing for uncertainty quantification in computational mechanics: A case study", *Scientific Programming*, **19**(4), 199-212
Georgescu, S., Chow, P. and Okuda, H. (2013), "GPU acceleration for FEM-based structural analysis", *Arch. Comput. Method. E.*, **20**(2), 111-121.
Grigoriu, M. (2002), *Stochastic calculus: applications in science and engineering*, Birkhäuser, Boston, MA, USA.
Huang, S., Li, Q. and Wu, J. (2010), "A general inflow turbulence generator for large eddy simulation", *J. Wind Eng. Ind. Aerod.*, **98**(10-11), 600-617.
Iannuzzi, A. and Spinelli, P. (1987), "Artificial wind generation and structural response", *J. Struct. Eng. - ASCE*, **113**(12), 2382-2398.
Intel (2014a), *Intel Xeon Processor E5-2670 Specifications*, URL: http://ark.intel.com/products/64595
Intel (2014b), *Intel Xeon Processor E7-8893 v2 Specifications*, URL: http://ark.intel.com/products/75260
Kareem, A. (1981), "Wind-excited response of buildings in higher modes", *J. Struct. Div. - ASCE*, **107**(4), 701-706.
Kepner, J. (2009), *Parallel MATLAB for multicore and multinode computers*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pennsylvania, USA.
Klöckner, A., Warburton, T., Bridge, J. and Hesthaven, J.S. (2009), "Nodal discontinuous Galerkin methods on graphics processors", *J. Comput. Phys.*, **228**(21), 7863-7882.
Krawezik, G.P. and Poole, G. (2010), "Accelerating the ANSYS direct sparse solver with GPUs", 2010 Symposium on Application Accelerators in High Performance Computing. Mathworks (2013), *MATLAB Documents*, The MathWorks Inc., Natick, Massachusetts, USA.
Melbourne, W.H. (1980), "Comparison of measurements on the CAARC standard tall building model in simulated model wind flows", *J. Wind Eng. Ind. Aerod.*, **6**(1), 73-88.
Melbourne, W. and Cheung, J. (1988), "Designing for serviceable accelerations in tall buildings", *Proceedings of the 4th International Conference on Tall Buildings*, Hong Kong and Shanghai.
Moore, G. (1998), "Cramming More Components Onto Integrated Circuits", *Proc. IEEE*, **86**(1), 82-85.
Nvidia (2013a), *TESLA K20 GPU active accelerator*, URL: http://www.nvidia.com/content/PDF/kepler/ Tesla-K20-Active-BD-06499-001-v04.pdf, Santa Clara, California: The Nvidia Inc.
Nvidia (2013b), *TESLA K40 GPU active accelerator*, URL:http://www.nvidia.com/content/PDF/kepler/ Tesla-K40-PCIe-Passive-Board-Spec-BD-06902-001_v05.pdf , Santa Clara, California: The Nvidia Inc.
Nvidia (2014), *CUDA C programming guide*, The Nvidia Inc., Santa Clara, California, USA.
Piccardo, G. and Solari, G. (2000), "3D wind-excited response of slender structures: Closed-form solution", *J. Struct. Eng. - ASCE*, **126**(8), 936-943.
Robert, C.P. and Casella, G. (2005), *Monte Carlo statistical methods*, Springer, Heidelberg, Germany.
Seo, D.W. and Caracoglia, L. (2012), "Statistical buffeting response of flexible bridges influenced by errors in aeroelastic loading estimation", *J. Wind Eng. Ind. Aerod.*, **104**, 129-140.
Seo, D.W. and Caracoglia, L. (2013), "Estimating life-cycle monetary losses due to wind hazards: Fragility analysis of long-span bridges", *Eng. Struct.*, **56**, 1593-1606.
Simiu, E. and Scanlan, R.H. (1996), *Wind effects on structures: fundamentals and applications to design*, John Wiley & Sons, New Jersey, USA.
Sipser, M. (2012), *Introduction to the theory of computation*, Cengage Learning, Boston, MA, USA.
Smith, M.A. and Caracoglia, L. (2011), "A Monte Carlo based method for the dynamic "fragility analysis" of tall buildings under turbulent wind loading", *Eng. Struct.*, **33**(2), 410-420.
Smith, M.A. (2009), *A Monte Carlo based method for the dynamic performance analysis of tall buildings under turbulent wind loading*, M.S. Thesis, Northeastern University, Boston, Massachusetts, USA.
Solari, G. (1988), "Equivalent wind-spectrum technique: theory and applications", *J. Struct. Eng. - ASCE*, **114**(6), 1303-1323.

Spence, S.M. and Gioffrè, M. (2012), "Large scale reliability-based design optimization of wind excited tall buildings", *Probabilist. Eng. Mech.*, **28**, 206-215.

Stantchev, G., Juba, D., Dorland, W. and Varshney, A. (2009), "Using graphics processors for high-performance computation and visualization of plasma turbulence", *Comput. Sci. Eng.*, **11**(2), 52-59.