Computers and Concrete, *Vol. 18, No. 5 (2016) 1019-1039*
DOI: http://dx.doi.org/10.12989/cac.2016.18.5.1019

A dynamic analysis algorithm for RC frames using parallel GPU strategies

Hongyu Li[1,2a], Zuohua Li[1b] and Jun Teng[*1]

[1]*School of Civil and Environment Engineering, Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen 518055, China*
[2]*College of Civil Engineering, Guilin University of Technology, Guilin 541004, China*

(Received November 25, 2015, Revised August 2, 2016, Accepted August 3, 2016)

Abstract. In this paper, a parallel algorithm of nonlinear dynamic analysis of three-dimensional (3D) reinforced concrete (RC) frame structures based on the platform of graphics processing unit (GPU) is proposed. Time integration is performed using Newmark method for nonlinear implicit dynamic analysis and parallelization strategies are presented. Correspondingly, a parallel Preconditioned Conjugate Gradients (PCG) solver on GPU is introduced for repeating solution of the equilibrium equations for each time step. The RC frames were simulated using fiber beam model to capture nonlinear behaviors of concrete and reinforcing bars. The parallel finite element program is developed utilizing Compute Unified Device Architecture (CUDA). The accuracy of the GPU-based parallel program including single precision and double precision was verified in comparison with ABAQUS. The numerical results demonstrated that the proposed algorithm can take full advantage of the parallel architecture of the GPU, and achieve the goal of speeding up the computation compared with CPU.

Keywords: nonlinear dynamic analysis; GPU; reinforced concrete; fiber model; parallel computing

1. Introduction

Earthquakes are usually as the greatest natural hazards. Recent seismic events, e.g., Northridge (1994), Kobe (1995), Wenchuan (2008), Chile (2010), Tohoku (2011), have caused considerable loss of life, the staggering monetary losses and functionality disruption. In the earthquake-resistant design, the analyzed structure should satisfy appropriate objectives for life safety and post-earthquake occupancy. Most current design codes usually require no significant damage of any components in the structure in the event of a minor earthquake. In the case of moderate earthquake, the structures should sustain repairable damages. For severe seismic action, the structure should be able to resist earthquakes without local or global collapse to maintain life and to minimize financial loss. In order to understand the response of the structure subjected to seismic excitation,

[*]Corresponding author, Professor, E-mail: tengj@hit.edu.cn
[a]Ph.D., E-mail: lhymonica@gmail.com
[b]Ph.D., Associate Professor, E-mail: lizuohua@hitsz.edu.cn

ISSN: 1598-8198 (Print), 1598-818X (Online)

further to provide adequate ductility and energy dissipation capacity for structure, the researchers and designers should takes as many means of dynamics analysis as possible. In general, there are two main ways to study structural dynamics response, one is experimental method, and the other is numerical one. In this paper, we focus on the time-history nonlinear dynamic analysis is used as a research tool.

In last several decades several methodologies to compute structural dynamics response have been proposed. Some researchers used mature methods such as, the initial Newmark-beta Method (Nathan M 1959), Wilson method (Wilson *et al*. 1972) for analysis of second-order accuracy, while the higher-order numerical integration algorithms (Golley 1996, Fung 1997) were introduced, for example, precise time step integration method (Fung 1997), all of the methods focus on improving the accuracy and efficiency of computing. With the rapid development of numerical algorithms, many authors have studied the dynamics response of reinforced concrete (RC) structures using the finite element method (FEM), focusing rigorously on the large-scale three dimensional (3D) problems (Manjuprasad *et al*. 2001; Yamada *et al*. 2008; Sasani and Kropelnicki 2008; Ohsaki *et al*. 2009). However time history analysis results from these algorithms can be assumed as correct, generally, the number of degrees of freedom (DOF) of a refined 3D model is computation intensive, especially for problems involving high nonlinearity, resulting much time consuming in the process of finite element analysis (FEA). In order to achieve a better computing efficacy and save calculation time, some parallel algorithms have been proposed, such as domain decomposition method (Yagawa *et al*. 1991), finite element tearing and interconnection method (Farhat and Roux 1991), multi-time-step integration method (Smolinski 1990). However, most of these algorithms procedure on modern computer architectures utilizing different application programming interfaces (APIs), which open multi-processing (OpenMP) for distributed-memory systems, POSIX threads and message passing interface (MPI) for shared memory systems (Adeli 2000). Majorities of parallel procedures developed on CPU-based architectures, offering advantages in availability and extensibility. However, there are still some challenges need to be solved. Developments of CPU-based parallel architectures were hindered by the high heat generation and power consumption. As increasing the number of processing units, it will cause dramatically high computational cost. Consequently, an alternative for CPU-based parallel procedure has become an important issue for large-scale structural dynamic analysis.

In recent years, with the emergence of general-purpose computing on graphic processing unit (GPU), large-scale computational problems using inexpensive off-the shelf hardware becomes possible. More and more researchers turned their attention to parallel computations on the GPU. A single GPU normally contains thousands of processing cores, capable of handling graphical processing operations efficiently in parallel, thus it has significant potential for solving large-scale engineering problems. APIs that support parallel programming on GPUs, like Compute Unified Device Architecture (CUDA) (nVidia 2013a) and Open Computing Language (OpenCL) (Khronos 2014), allow researchers to parallelize their computing programs or sub-programs to be executed on the GPU. Nowadays, applications of GPU computing are widely found in a variety of fields, such as image processing (Eklund *et al*. 2013), computational fluid dynamics (Chetverushkin *et al*. 2013), earthquake modeling (Komatitsch *et al*. 2009) and environmental science (Bryan 2013).

Further applications suitable for GPUs were developed to enable solution of the general linear algebra problems which commonly occur in engineering, such as direct solvers like Cholesky decomposition, Gauss–Jordan elimination and LU decomposition which were developed and used for solving dense linear systems (Yang *et al*. 2012, Galoppo *et al*. 2005). The conjugate gradient solvers were used to solve of the sparse linear system of equations resulting from a FEM discretization (Helfenstein and Koko 2011, Weber *et al*. 2013). There are also several applications of GPUs to nonlinear FE problems (Mafi and Sirouspour 2014, Huthwaite 2014). Göddeke and his group (Göddeke 2010, 2011, Komatitsch *et al*. 2010) have done extensive further work in the area of FEM on the GPU, introduced multigrid to FEM and optimized it for GPUs, including incorporate GPU technologies into their FEM package FEAST (Finite Element Analysis and Solutions Tools) (Turek *et al*. 2010).

It is certain without doubting that the development of high performance numerical methods and computer programs for dynamic structural analysis will plays significant role in improving structural analysis efficiency. GPU-accelerated FEM structural analyses contribute to the efficiency of linear analysis processes (Miao *et al*. 2015) and nonlinear analysis processes (Yang *et al*. 2014, Li *et al*. 2015, Wei and Luca, 2015). An extensive list of software that can be used to obtain GPU acceleration was given (Georgescu *et al*. 2013). However, in the field of structural dynamics, the lack of research and success cases to show the remarkable results is the reality yet (Kand *et al*. 2014). And there are many challenges on solution of nonlinear dynamic structural engineering problems.

In this paper, we present a novel parallel algorithm of nonlinear dynamic analysis of 3D RC frame structures on the GPU-based platform. The finite element models are simulated using a fiber beam model, and the material nonlinearities are considered. For dynamic analysis, the Newmark implicit time integration is used. The proposed GPU-based parallel Newmark-beta integration algorithm is implemented using CUDA, and the GPU-based parallel PCG method is introduced for solving the system of equations resulting from implicit time integration. The GPU technology in this work is more fine-grain parallelism, whose computing time of each subroutine is relatively small. Because each subroutine maps to the calculation of an element of an array, it is efficient to have adjacent threads operate on adjacent data. Furthermore, the performance of the developed parallel program is tested on the GPU-based platform. The same computation was conducted using a commercial FEM software ABAQUS, and the results show well agreement with the programs of this paper. Numerical experiments demonstrate that the proposed parallel program could effectively improve the efficiency of nonlinear dynamic analysis. The scenarios in this paper show a very promising future in the field of large-scale structural dynamic analysis.

2. Problem statement

The typical FEM-based nonlinear dynamic analysis process mainly includes three procedures: initialization, time integration, and post-processing. A flowchart of the nonlinear dynamic structural analysis is presented in Fig. 1. We should note that this figure presents general analysis steps; however, the actual flowcharts of most structural analysis programs are more complicated. Specifically for initialization, operations include data input, elements matrix/vector calculations and global matrix assembly, material properties, boundary condition enforcement, solution parameters etc. Different schemes are used in the time integration part to calculate new accelerations, velocities and displacements at each time step (corresponding to first loop in Fig. 1), which include implicit time integration schemes (the Newmark or Wilson method), explicit time integration schemes (the central difference method), or other non-iterative schemes. The solution of a system of linear or nonlinear equations is required at each time step (corresponding to the inner loop in Fig. 1). Post-processing is where the structural dynamic response from time integration are analyzed (displacement, strain, stress, etc.).

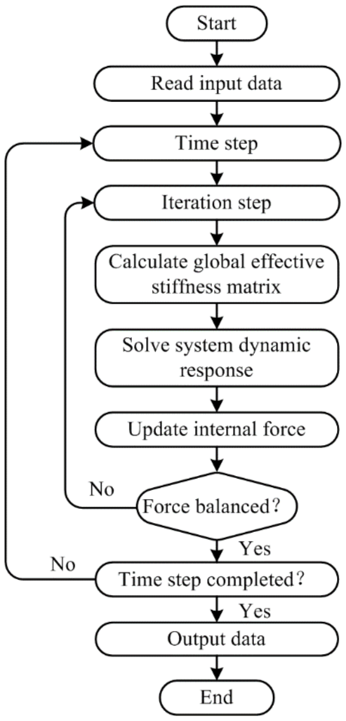


Fig. 1 A typical FEM-based flowchart for nonlinear dynamic structural analysis

The majority of computational cost involved running a structural dynamic analysis processing is typically concentrated in time integration and the solution of global system equations, which consists of two nested loops as shown in Fig.1. Some surveys about the computational time for a nonlinear dynamic analysis of a refined model showed that more than 90% of time was consumed in the process of integration and equation solution (Fahmy and Namini 1994, Noor 1997). Therefore, this work focuses on GPU parallelization of whole iteration process and does not address the issue in the parallelization of initialization procedure and post-processing.

A typical GPU architecture is organized as an array of multiprocessors or cores, which has much greater degree of parallelism within the GPU itself. The CUDA programming model introduced by NVIDIA is designed for highly parallel execution on the GPUs. A subroutine named kernel that runs on the GPU is executed by many threads in parallel. Threads are grouped in a grid of blocks with each block assigned to a unique core. A core would then schedule a block of threads into 32-thread warps. A warp is the basic unit of job scheduling on the GPU. Each warp will always follow the same instruction schedule with each thread handling a different data set. In other words, although all threads execute the same code, these threads typically operate on different data.

In the view of computer operations, an accurate FEM structural model is typically composed of tens of thousands of elements. And element has multiple nodes which have multiple degrees of freedom, resulting in a significant number of global DOFs for the whole system. Too many DOFs will cause dramatically high computational cost during using the FEM. Therefore, a parallelism scheme needed. In general, for FEM-based numerical techniques, two levels of parallelism can be employed: (1) parallelism considering each element as the computational data objects; and (2) parallelism considering each node as the computational data object. These computational data

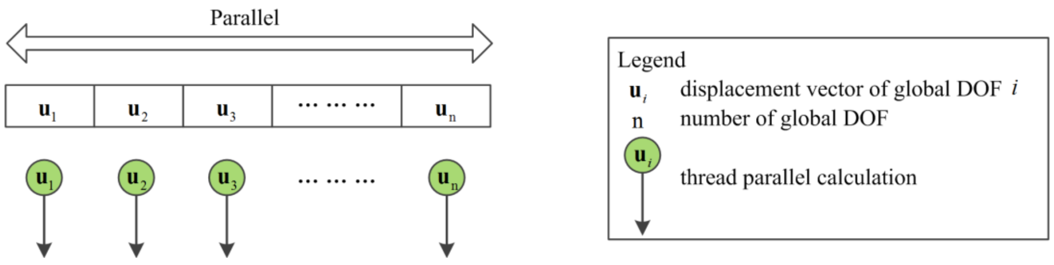


Fig. 2 Parallelism schematic of threads to global DOF

objects can be partitioned and distributed to each processor of the parallel computing platform, such as MPI mentioned above. However, the model of parallelism on the GPU is a fine-grained model, where it is most efficient to have adjacent threads operate on adjacent data, such as elements of an array. In this work, each global DOF could be treated as an independent computing unit whose variables are updated independently. In the other words, data in global matrices/vectors (stiffness matrix, force, displacement, etc.) are independent. Intensive arithmetic operations make these data particularly suitable for parallel implementation on threads. Parallelism considering each global DOF is introduced, and the corresponding operations can be simultaneously performed in each thread as shown in Fig. 2.

3. FEM for nonlinear dynamic analysis of reinforced concrete structures

In this section, the basic formulation of the dynamic analysis of 3D frame structures is briefly introduced. After deriving the equations in discrete spatial and time domains, the iterative preconditioned conjugate gradients (PCG) method for solving the discretized matrix-form equations will be discussed. Then a Timoshenko-type 3D beam element is developed by discretization of the cross section into fibers.

3.1 Dynamic FEM

In dynamic analysis, either explicit time integration or implicit time integration can be employed. Typically, fewer degrees of freedom are considered with a large number of small time steps, advocating the use of explicit time integration schemes. Though the explicit approaches are easier to be implemented, they are only conditionally stable. In contrast, a large number of degrees of freedom with a large time step is efficiently handled by implicit time integration schemes, and numerical stability is guaranteed independent of the integration step length used (Belytschko 1983). In this work, one of the most widely known implicit time integration techniques, the Newmark algorithm was used for solving the dynamic equation of nonlinear frame structures in the parallel program.

In nonlinear analyses, it is assumed that the physical properties remain constant only for short increments of time; accordingly, it is convenient to reformulate the response in terms of the incremental equation of motion, as follows

$$\mathbf{M}\Delta\ddot{\mathbf{u}} + \mathbf{C}\Delta\dot{\mathbf{u}} + \mathbf{K}\Delta\mathbf{u} = -\mathbf{M}\Delta\ddot{\mathbf{u}}_g \tag{1}$$

where, **M** is the global mass matrix, **C** is the damping matrix, **K** is the stiffness matrix, $\ddot{\mathbf{u}}_g$ is ground acceleration vector, and $\Delta\mathbf{u}$, $\Delta\dot{\mathbf{u}}$, and $\Delta\ddot{\mathbf{u}}$ is incremental displacement, velocity, and acceleration vectors respectively.

The general numerical integration approach to dynamic response analysis is step-by-step method. In this paper, we choose Newmark-beta algorithm. The incremental velocity and displacement over a short time step are calculated from

$$\Delta\dot{\mathbf{u}} = \ddot{\mathbf{u}}_t\Delta t + \gamma(\ddot{\mathbf{u}}_{t+\Delta t} - \ddot{\mathbf{u}}_t)\Delta t \tag{2}$$

$$\Delta\mathbf{u} = \dot{\mathbf{u}}_t\Delta t + \frac{1}{2}\ddot{\mathbf{u}}_t\Delta t^2 + \beta(\ddot{\mathbf{u}}_{t+\Delta t} - \ddot{\mathbf{u}}_t)\Delta t^2 \tag{3}$$

where γ and β are factors that control the stability and accuracy. Set $\gamma=1/2$ and $\beta=1/4$.

Solved Eqs. (2)-(3) to obtain $\Delta\ddot{\mathbf{u}}$ and $\Delta\dot{\mathbf{u}}$. Then substituted them into Eq. (1)

$$\left(\mathbf{K} + \frac{1}{\beta\Delta t^2}\mathbf{M} + \frac{\gamma}{\beta\Delta t}\mathbf{C}\right)\Delta\mathbf{u} = -\mathbf{M}\Delta\ddot{\mathbf{u}}_g + \mathbf{M}\left(\frac{1}{\beta\Delta t}\dot{\mathbf{u}}_t + \frac{1}{2\beta}\ddot{\mathbf{u}}_t\right) + \mathbf{C}\left(\frac{\gamma}{\beta}\dot{\mathbf{u}}_t + \left(\frac{\gamma}{2\beta} - 1\right)\Delta t\ddot{\mathbf{u}}_t\right) \tag{4}$$

when $\dot{\mathbf{u}}_t$, and $\ddot{\mathbf{u}}_t$ at time *t* were given, the incremental displacement $\Delta\mathbf{u}$ can be calculated from Eq. (4). Then the total values were obtained.

In Newmark-beta algorithm, the limits on the integration step length Δt aim to ensure convergence and stability of the solution. The integration step length must be small enough for solution accuracy, but for a cost effective solution it should not be unnecessarily small. Several researchers have adopted different limits on the integration step length (Walpole and Shepherd 1969), which is given in terms of the highest eigenvalue in the system

$$\Delta t \le \left(\frac{1}{10} \sim \frac{1}{6}\right)\frac{2\pi}{\omega_{\max}} \tag{5}$$

where $\omega_{\max}$ is the maximum frequency of the system.

3.2 Solving linear system of equations

The elemental equations for dynamic analysis after implicit numerical integration are in the global matrix equation in the form of Eq. (6) by considering interactions among the elements as well as boundary conditions

$$\mathbf{Au} = \mathbf{b} \tag{6}$$

The direct methods such as Gauss Elimination are generally employed to solve linear system of equations. In general, iterative methods are more suitable for parallelization of the computations, and also more efficient in terms of memory storage. The conjugate gradient (CG) algorithm is one of the best known iterative techniques for solving linear systems. From a numerical perspective, lack of robustness is a widely recognized weakness of CG. This drawback can be improved by using preconditioning (Benzi and Tuma 1999).

It is assumed that a preconditioner **M** is available and symmetric positive definite. A mathematically equal preconditioned system is obtained as follows

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{u} = \mathbf{M}^{-1}\mathbf{b} \tag{7}$$

Jacobi preconditioners are commonly used preconditioners for parallel formulations (Phoon *et al*. 2002). In this work, **M** is a diagonal matrix comprising of the diagonal entries of matrix **A**.

3.3 Fiber beam element formulations

In this work, RC 3D frames are simulated by fiber beam model including a 12 degrees of freedom element, which is based on the Timoshenko beam theory. This fiber beam model is improved to address the interaction between axial force, bidirectional shear, biaxial bending, and torsion.

The 3-D fiber beam element is shown in Fig.3. The proposed fiber beam element follows the plane section hypothesis, thus the strain of the fiber *i* is related to the strain of the section as

$$\boldsymbol{\varepsilon}^i(x) = \left\{\varepsilon_x \quad \gamma_{xy} \quad \gamma_{xz}\right\}^T = \mathbf{b}(x)\mathbf{d}^s(x) \tag{8}$$

where $\mathbf{b}(x)$ is the compatibility matrix considering the effect of shear given by

$$\mathbf{b}(x) = \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & 0 \\ 0 & 0 & 1 & y & 0 & 0 \end{bmatrix} \tag{9}$$

Then the tangent stiffness matrix and node force vector of the section were obtained by summation of the contribution of all fibers as follows (Mullapudi and Ayoub 2013)

$$\mathbf{K}^s(x) = \sum_{i=1}^{n}\left(\mathbf{b}^T(x)\mathbf{D}_i\mathbf{b}(x)\right)A_i \tag{10}$$

$$\mathbf{F}^s(x) = \sum_{i=1}^{n}\left(\mathbf{b}^T(x)\boldsymbol{\sigma}_i\right)A_i \tag{11}$$

where n is the number of fibers in a section; A_i *i*s the area of the fiber; $\mathbf{D}_i$ is material stiffness matrix; and $\boldsymbol{\sigma}_i$ is material stress vector.

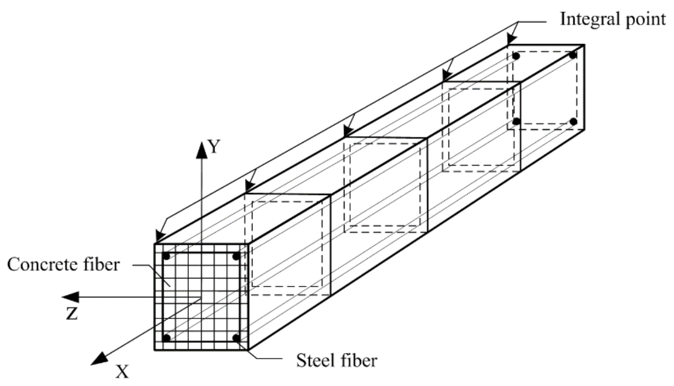


Fig. 3 Fiber beam element

The accuracy of the above summation depends on the number of the fibers. Additionally, section behavior is evaluated through fiber discretization with the appropriate material constitutive models. In order to simulate the concrete the modified Kent–Park model is applied, where the monotonic envelope of the concrete in compression follows the model of Kent and Park (1971) as extended by Scott *et al*. (1982).The hysteretic stress-strain relation of the concrete implemented with Blakely and Park model (1973) and the concrete tensile strength is also considered, which is proposed by Yassin (1994). The nonlinear behavior of the reinforcing bars was simulated within the model is discussed in Menegotto and Pinto (1977). More details about the presented spatial discretization and material model selection can be found in literature (Spacone *et al*. 1996).

The stiffness matrix and node force vector at element level are presented in Eqs. (12) and (13). In particular, these formulations are evaluated by using a numerical integration such as the Gauss-Lobatto integration scheme (Stroud and Secrest 1966). Here, the superscript *e* indicates the matrix/vector of the element; superscript *s* indicates that the matrix/vector of section (i.e. section at integral point).

$$\mathbf{K}^e = \int_0^L \mathbf{B}^T(x)\mathbf{K}^s(x)\mathbf{B}(x)dx = \sum_{i=1}^{nip} \omega_i \mathbf{B}^T(x_i)\mathbf{K}^s(x_i)\mathbf{B}(x_i) \tag{12}$$

$$\mathbf{F}^e = \int_0^L \mathbf{B}^T(x)\mathbf{F}^s(x)dx = \sum_{i=1}^{nip} \omega_i \mathbf{B}^T(x_i)\mathbf{F}^s(x_i) \tag{13}$$

where *nip* is the number of integral points; ω_i is weighted sum of the integrand function at certain integral point x_i, **B**(x) is the strain-displacement matrix, and **N**(x) is the displacement interpolation function matrix.

4. Parallelization strategies

In this section, we describe two main problems of parallelization strategies mentioned in Section 2, that is, real-time updating of the incremental matrices/vectors by using the Newmark-beta algorithm and solving of a large system of linear Eq. (6). Then we state the framework of entire program.

4.1 Implementation of Newmark-beta on the GPU

During the step-by-step solution, the specific operations of parallelization Newmark-beta implement on the GPU including the CPU procedure as comparison are presented in Fig. 4(a)-(b). We should note that this figure presents elastic analysis steps and all of the steps of Newmark-beta algorithm are running on the GPU. However, actually most structural analysis programs are nonlinear, which will be discussed in section 4.3 (see Fig. 6). For the Newmark-beta algorithm, both of the calculations and solutions at element level of Eq. (4) involve computations that are not easily partitioned along a number of threads. In addition, GPU requires the same instruction to be executed by warps. This fact further hinders the partitioning of a computational work along threads without instruction branching inside a warp. For this reason, in the implementation of CUDA kernels, computations for each global DOF are assigned to one thread. This approach simplifies the implementation, and these kernels are fully scalable. Furthermore, it can be executed for

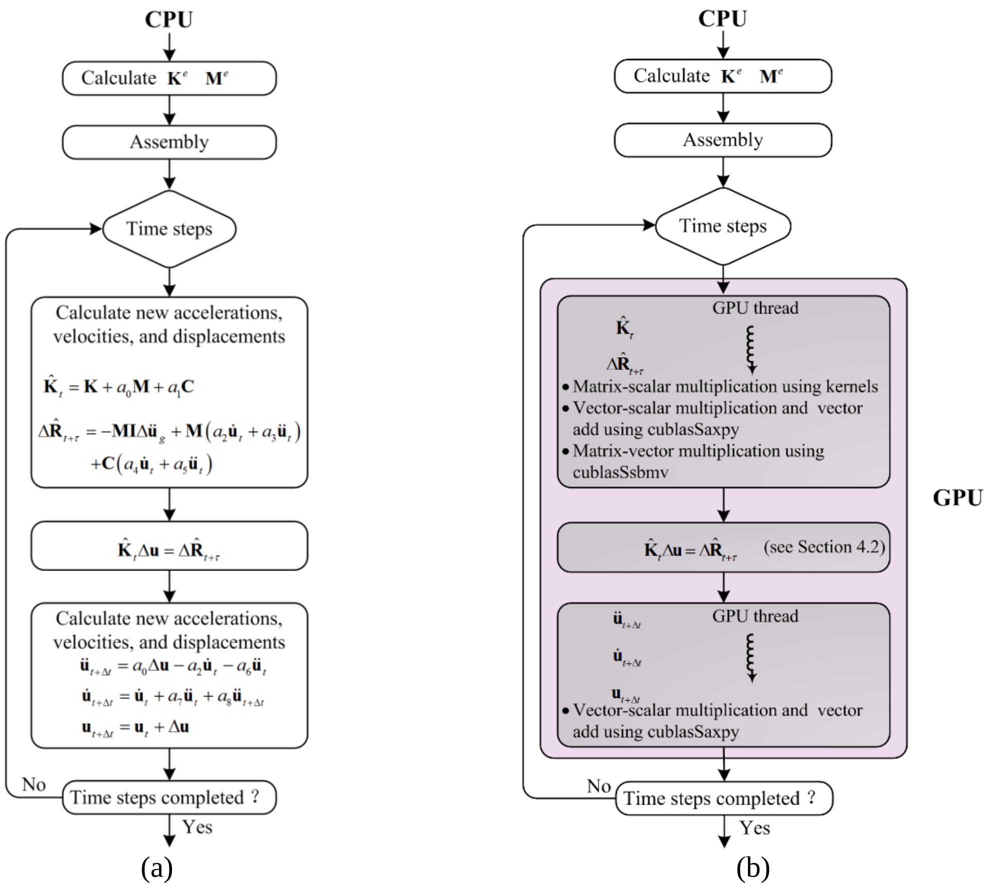

CPU
Calculate $\mathbf{K}^e$ $\mathbf{M}^e$
Assembly
Time steps
Calculate new accelerations, velocities, and displacements
$\hat{\mathbf{K}}_t = \mathbf{K} + a_0\mathbf{M} + a_1\mathbf{C}$
$\Delta\hat{\mathbf{R}}_{t+\tau} = -\mathbf{MI}\Delta\ddot{\mathbf{u}}_g + \mathbf{M}(a_2\dot{\mathbf{u}}_t + a_3\ddot{\mathbf{u}}_t) + \mathbf{C}(a_4\dot{\mathbf{u}}_t + a_5\ddot{\mathbf{u}}_t)$
$\hat{\mathbf{K}}_t\Delta\mathbf{u} = \Delta\hat{\mathbf{R}}_{t+\tau}$
Calculate new accelerations, velocities, and displacements
$\ddot{\mathbf{u}}_{t+\Delta t} = a_0\Delta\mathbf{u} - a_2\dot{\mathbf{u}}_t - a_6\ddot{\mathbf{u}}_t$
$\dot{\mathbf{u}}_{t+\Delta t} = \dot{\mathbf{u}}_t + a_7\ddot{\mathbf{u}}_t + a_8\ddot{\mathbf{u}}_{t+\Delta t}$
$\mathbf{u}_{t+\Delta t} = \mathbf{u}_t + \Delta\mathbf{u}$
No
Time steps completed ?
Yes
(a)
CPU
Calculate $\mathbf{K}^e$ $\mathbf{M}^e$
Assembly
Time steps
GPU thread
$\hat{\mathbf{K}}_t$
$\Delta\hat{\mathbf{R}}_{t+\tau}$
• Matrix-scalar multiplication using kernels
• Vector-scalar multiplication and vector add using cublasSaxpy
• Matrix-vector multiplication using cublasSsbmv
$\hat{\mathbf{K}}_t\Delta\mathbf{u} = \Delta\hat{\mathbf{R}}_{t+\tau}$ (see Section 4.2)
GPU thread
$\ddot{\mathbf{u}}_{t+\Delta t}$
$\dot{\mathbf{u}}_{t+\Delta t}$
$\mathbf{u}_{t+\Delta t}$
• Vector-scalar multiplication and vector add using cublasSaxpy
GPU
No
Time steps completed ?
Yes
(b)

Fig. 4 (a) CPU sequential and (b) GPU parallel flowcharts of Newmark-beta algorithm. The white boxes are executed on the CPU, while the filled boxes are launched on the GPU. a is the constants of integration, whose details are referred to reference (Bathe and Wilson 1976).

different-sized DOF problems. GPU can substantially accelerate these types of computations due to the data parallel structure and arithmetic-intensive nature of this problem, particularly for models with a large number of elements. Additionally, in the process of implementation of the Newmark-beta algorithms, in order to improve the efficiency of computation, the NVIDIA's CUBLAS library (nVidia, 2013b) was used in addition to our programmed kernels. It should be noted that at each time step Newmark algorithm requires solving a linear system of equations which is the most time consuming process. Implementation of the parallel PCG solver will be further discussed in Section 4.2.

4.2 Parallel PCG solver

Algorithm 1 shows the parallel PCG solver strategy. As shown in Algorithm 1, for each iteration (lines 3-9), the PCG algorithm uses a matrix-vector multiplication operation in line 3, vector dot product operations in lines 4 and 8, vector scalar multiplication and vector addition operations in lines 5, 6 and 9, and a preconditioning operation in line 7. Most of these functions are

Algorithm 1 The parallel PCG solver strategy

```
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{u}_0, \mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0, \mathbf{d}_0 = \mathbf{z}_0$        ! CPU initialization
    do i = 0, 1, …
        $\mathbf{q}_i = \mathbf{A}\mathbf{d}_i$                                  ! GPU cublasDsbmv
        $\alpha_i = \mathbf{r}^T_i\mathbf{z}_i / \mathbf{d}^T_i\mathbf{q}_i$                        ! GPU cublasDdot
        $\mathbf{u}_{i+1} = \mathbf{u}_i + \alpha_i\mathbf{d}_i$                          ! GPU cublasDaxpy
        $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i\mathbf{q}_i$                          ! GPU cublasDaxpy
        $\mathbf{z}_{i+1} = \mathbf{M}^{-1}\mathbf{r}_{i+1}$                            ! GPU diagonal scaling kernel
        $\beta_i = \mathbf{r}^T_{i+1}\mathbf{z}_{i+1} / \mathbf{r}^T_i\mathbf{z}_i$                ! GPU cublasDdot
        $\mathbf{d}_{i+1} = \mathbf{z}_{i+1} + \beta_i\mathbf{d}_i$                       ! GPU cublasDaxpy
        if norm($\mathbf{r}$) < tolerance or i > $i_{max}$ exit        ! CPU logical judgement
    end do
```

Algorithm 2 Diagonal scaling kernel

```
! kernel
attributes(global) subroutine diagonal scaling kernel ( A,B,C,N)
    implicit none
    real(8),device:: A(N),B(N),C(N)              ! #real(8) for double precision
    integer,value::N
    integer :: i
    i = blockDim%x*(blockIdx%x-1)+threadIdx%x    ! identify each thread index
    if(i<=N) C(i) = A(i)*B(i)                     ! fine-grained parallelism
    call syncthreads()                            ! each thread waits until all
end subroutine diagonal scaling kernel           ! threads have reached this call
! launch kernel
call diagonal scaling kernel <<<nb, nt>>> ( Ad,Bd,Cd,N )
```

directly available from CUBLAS library, except for the *diagonal scaling kernel* which was implemented by ourselves.

A parallelizing *diagonal scaling kernel* is straightforward and the algorithm is shown in Algorithm 2. *N* parallel threads is launching to calculate the diagonal scaling, each thread identifies itself via an index *i*. An execution configuration of <<<blocks, threads>>> specifies that the kernel is executed by *nb* parallel blocks with *nt* GPU threads. In the case which is shown in Fig. 5, *N* is equal to 6. We can simply change the number 6 in the line #define *N* to 10000 or 100000 to launch tens of thousands of parallel blocks. Note that this parallelism where different threads modify adjacent data of the array is a fine-grained parallelism.

In Jacobi preconditioning, preconditioner **M** is the diagonal matrix of **A** and was stored as a vector. Here matrix **A** is the left side of linear system of equations $\hat{\mathbf{K}}_t\Delta\mathbf{u} = \Delta\hat{\mathbf{R}}_{t+\tau}$, where $\hat{\mathbf{K}}_t$ is a $N \times N$ symmetric banded sparse matrix. Yang *et al*. have mentioned that a dynamic analysis of a refined model of frame building may exceed 20,000 DOFs (Yang *et al*. 2012), meaning the matrix $\hat{\mathbf{K}}_t$ is of the size 20,000 × 20,000, while $\Delta\hat{\mathbf{R}}$ is of the size 20,000 × 1. With a large number of DOFs, the storage spaces and computational costs are increased. As stiffness matrix is stored in the global memory, storage schema of these data structures is designed for coalesced access. Thus, to

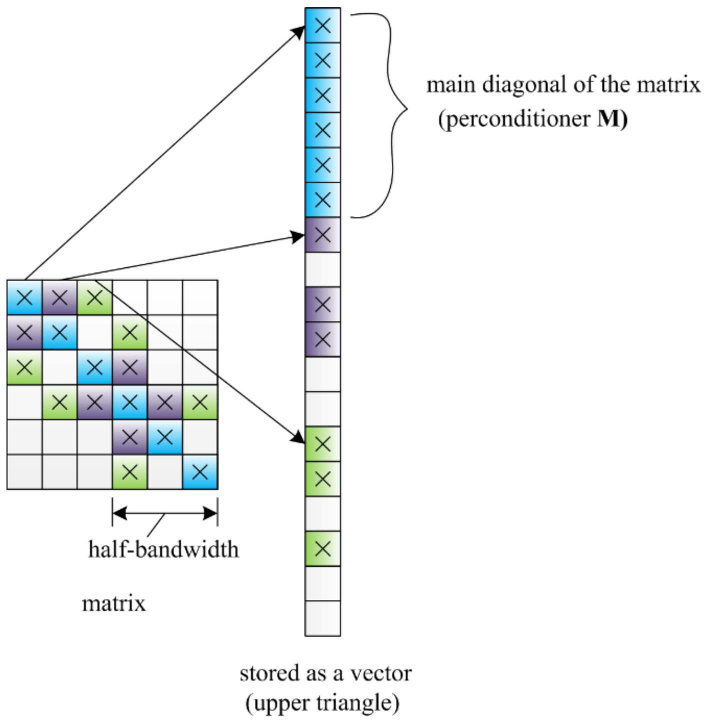


Fig. 5 Storage schema of global stiffness matrix

access to these arrays in global memory efficiently, all the FEM matrices and vectors with *m* elements can be stored in *m* one-dimensional arrays. In the storage of stiffness matrix a storage schema that is presented in Fig. 5 was used.

Since stiffness matrix is symmetric, it saves space to store only the upper (or lower) triangular part of the matrix. We will consider storing the upper triangular part. The array is of dimension $m = N \times nband$, where N is the corresponding global DOF number and *nband* is equal to *half-bandwidth* +1. We show a case in which N = 6 and *half-bandwidth* = 2 as in Fig. 5.

4.3 Framework of entire program

Based on the discussion above, a GPU-based program was developed to analyze RC 3D frame structures subjected to ground motion. The framework of entire program is illustrated in Fig. 6. First, the main program executed in the CPU, the element stiffness matrices, mass matrices and damper matrices were pre-calculated. Then the assembly process is performed by the CPU because several uncoalesced global memory accesses and consequently poor performance would occur at GPU for the same process. Second, nodal and element data that are required to do the calculation are stored in the global memory of GPU and first sent to the GPU. Then threads can be assigned on the GPU to perform the nonlinear dynamic analysis. When an update with independency of other global arrays composed of the unique DOF occurred, the nodal and element data should be sent to CPU for dealing with some serial operations, such as state determination of elements (Spacone *et al*. 1996). Finally, results obtained from the GPU are transferred into the CPU and the post process is performed on the CPU naturally.

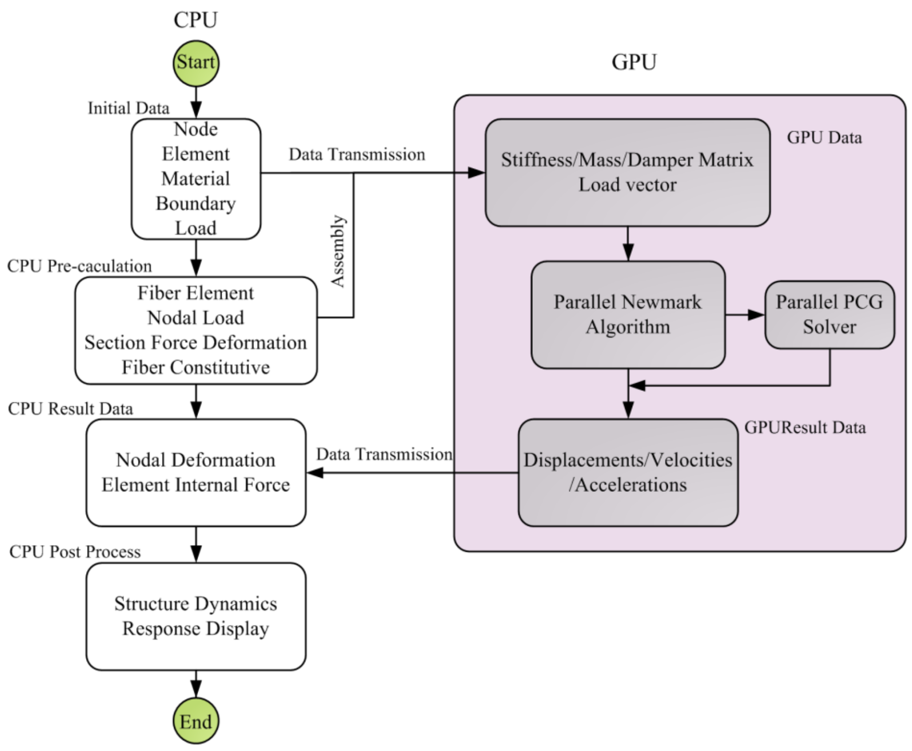


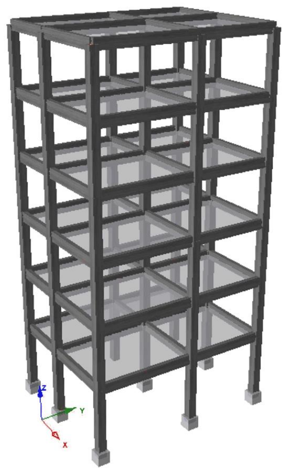
Fig. 6 The framework of entire program for large-scale structural nonlinear dynamic analysis

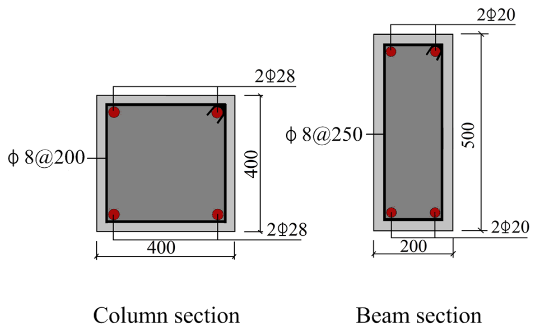
5. Numerical results

5.1 Benchmark test

Our numerical tests were conducted on (1) an Intel Core i5-2300 processor running 64-bit Windows 7 system with 8 Gb memory, 333 MHz DDR3, and (2) a NVIDIA Geforce GTX460 GPU processor in the same Core i5 system with CUDA Fortran Toolkit 5.0. The Intel Core i5-2300 offers four cores, initially clocked at 2.8 GHz. Each core has a separate 32 Kb L1 for both instruction and data caches, and a 1Mb L2 cache. It features one 6 MB L3 shared cache that is dynamically allocated to each processor core. The NVIDIA Geforce GTX460 graphics card has compute capability of 2.1 and has 7 multiprocessors, 336 CUDA cores, 1 GB of global memory and 1.4 GHz shader clock. PGI Accelerator™ Visual Fortran integrated Microsoft Visual Studio 2012 was employed to compile the program writing in CUDA FORTRAN.

To benchmark the performance of the dynamic analysis of 3D frame structures, a simple numerical example is used to validate the developed paralleled GPU program. A similar CPU-based program is also developed. The RC 3D frame model and the reinforcement details within structural members are shown in Fig. 7(a)-(b). The dimensions of cross section of beam and columns are 20cm × 50cm (width × depth) and 40cm × 40cm (width × depth) respectively. Besides gravity loads acting on the frame, dead loads 3.5 kN/m^2 and live loads 2.0 kN/m^2 at each floor are considered in the analysis. The masses of the tributary floor areas are assumed to be lumped at the beam column joints.


(a)



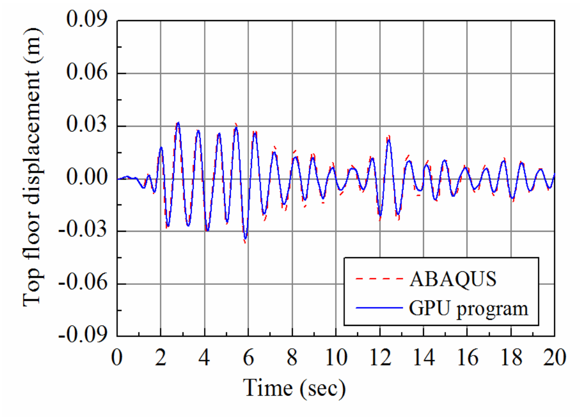
(b)

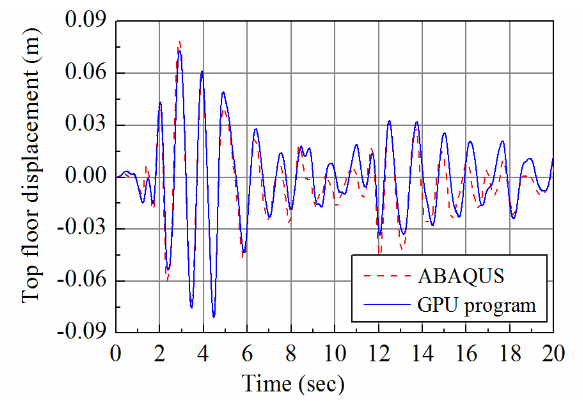
Fig. 7 The model for benchmark: (a) Modeling of the 3D six-story frame structure and (b) reinforcement details of structural members

A nonlinear steel model used in the present study is discussed in Menegotto and Pinto (1977). The longitudinal steel is assumed to have initial elasticity modulus of 206 GPa, yield stress 400.0 MPa and strain hardening ratio 1.0%. The lateral steel is assumed to have yield strength 240.0 MPa. The compressive strength of the concrete is assumed as 26.0 MPa and the concrete cover is 2.5 cm. Due to the confining effects of the transverse reinforcement of the Kent and Park model, the concrete properties in the core concrete are 26.8 MPa and 0.00206 which are concrete compressive strength and compressive strain corresponding to maximum strength, respectively.

An acceleration ground motion of recorded during the Imperial Valley, California, Earthquake in 1940 was adopted as input in the analysis. The peak ground acceleration (PGA) is normalized to 100gal and 200gal, in order to evaluate the nonlinear responses of the structure. The duration of time history analysis was set to 20.0 s at a 0.005 s time step increment and the amount of time steps was 4000. This duration was long enough to cover large- and small- amplitude ranges of response. A classic Rayleigh damping ratio of 5% was applied.

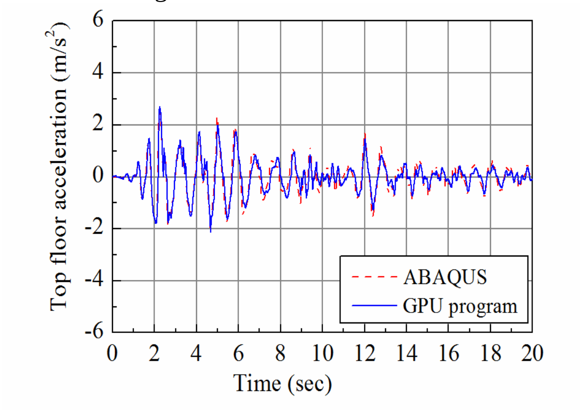
First, we compared the results of GPU-based program with ones obtained using a commercial FEM software ABAQUS. The results of top floor displacement, acceleration and base shear were obtained by GPU and ABAQUS respectively as shown in Figs. 8-10. Excellent agreement is observed up to T = 7 s. In this large amplitude region, the frequency content and the waveform of the calculated response is very close to what was obtained by ABAQUS. Beyond this point and before T = 12 s, when low amplitude response starts, the response obtained by GPU program deviates from the results of ABAQUS. The difference is more salient in the acceleration response. Such difference signifies that, in the low amplitude regions, the fiber model resulted in structural stiffness is larger because of the scheme of interpolation the displacement function. In addition, the results of inter-story drift ratio at the time of maximum response were presented in Table 1 and Table 2. The single precision and double precision programs are conducted on GPU, and the obtained errors of maximum inter-story drift are both less than 5.0% and the errors are only 2.0% at top floor. Through the comparison, the results of our work showed good agreement with the ABAQUS and the accuracy of the GPU-based program is good enough for dynamic analysis.

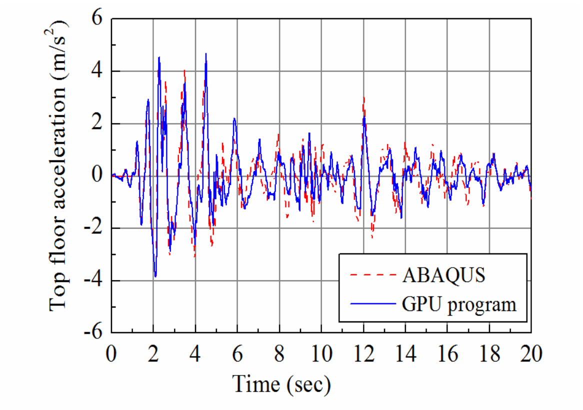




(a) (b)

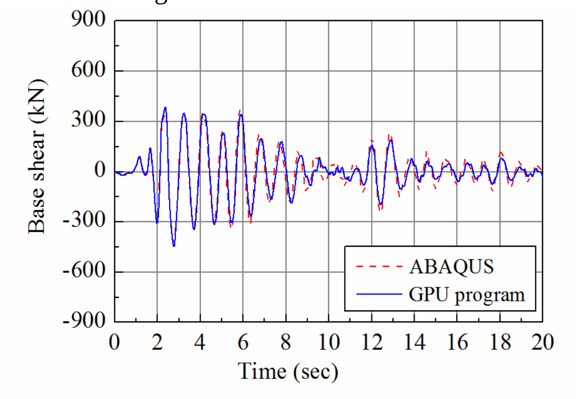
Fig. 8 Top floor displacement result by ABAQUS and GPU-based program: (a) PGA = 100 gal and (b) PGA = 200 gal

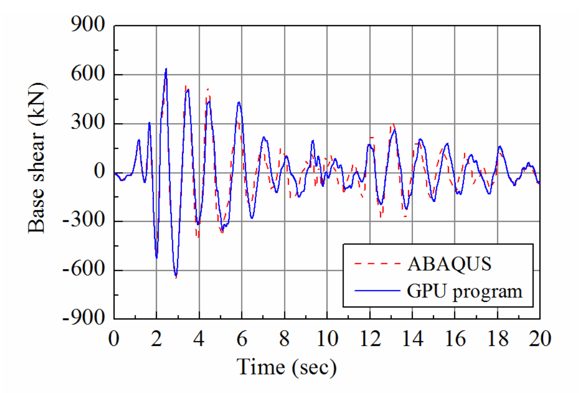




(a) (b)

Fig. 9 Top floor acceleration result by ABAQUS and GPU-based program: (a) PGA = 100 gal and (b) PGA = 200 gal





(a) (b)

Fig. 10 Base shear result by ABAQUS and GPU-based program: (a) PGA = 100 gal and (b) PGA = 200 gal

Table 1 Maximum inter-story drift ratio (PGA = 100gal)

Floor	GPU-based program		ABAQUS	Error (%)	
	Single precision	Double precision		Single precision	Double precision
1F	0.001750	0.001750	0.001719	-1.80	-1.80
2F	0.002452	0.002452	0.002504	2.08	2.08
3F	0.002283	0.002283	0.002388	4.40	4.40
4F	0.001858	0.001859	0.001925	3.43	3.48
5F	0.001315	0.001315	0.001364	3.59	3.59
6F	0.000757	0.000757	0.000768	1.02	1.02

Table 2 Maximum inter-story drift ratio (PGA = 200gal)

Floor	GPU-based program		ABAQUS	Error (%)	
	Single precision	Double precision		Single precision	Double precision
1F	0.004397	0.004397	0.004512	2.75	2.75
2F	0.006513	0.006513	0.006770	3.80	3.80
3F	0.005857	0.005858	0.006271	3.52	3.51
4F	0.004113	0.004112	0.004302	4.39	4.42
5F	0.002218	0.002219	0.002152	-1.19	-1.23
6F	0.000972	0.000972	0.000981	1.83	1.83

Table 3 Time consuming in each procedure (units: sec)

	PGA = 100 gal	PGA = 200 gal
Input data	1.5	1.5
State determination	22.1	22.4
Stiffness matrix	31.4	31.7
Solve equations	73.5	77.6
Equivalent force	6.4	6.5
Total	134.9	139.7

The time elapsed of the benchmark test was decomposed by the internal procedures, listed in Table 3. The most time consuming procedure in the nonlinear dynamic analysis is the process of solving the equations of linear system, which takes about 55% of the total computational time. For comparison, the total computational time using ABAQUS was measured accordingly. The solution time of the program on the GPU-based platform is approximately equal to 22-27% of the corresponding time required for the ABAQUS. Though the number of global DOF in this model is not large, the parallel processing ability of GPU performed very well during the computation, the time deduction was significant. We can infer that when the number of DOF increases, the efficiency of GPU will improve accordingly.

5.2 Numerical studies

Table 4 Size of models for the test cases

Model	Elements	DOFs	Size of matrix	Half-bandwidth	Number of elements in banded matrix
1	210	540	540 × 540	185	99,900
2	315	810	810 × 810	275	222,750
3	800	1920	1920 × 1920	485	931,200
4	1300	3000	3000 × 3000	605	1,815,000
5	1920	4320	4320 ×4320	725	3,132,000
6	2400	5400	5400 × 5400	905	4,887,000
7	2860	6480	6480 × 6480	1090	7,063,200
8	3250	7500	7500 × 7500	1505	11,287,500

Table 5 relationship between block size and the time cost (units: sec)

Model	Block size						
	16	32	64	128	256	512	1024
1	9.63	8.85	8.84	8.83	8.84	9.04	9.57
2	16.59	16.01	16.11	16.10	16.17	16.65	17.24
3	36.82	36.25	36.27	36.22	36.29	36.72	38.09
4	63.38	61.97	61.86	61.90	62.55	64.12	66.45
5	114.88	110.64	110.97	111.06	112.17	114.04	117.78
6	160.37	154.09	155.39	155.61	155.78	159.94	165.10
7	225.76	218.23	218.54	218.36	219.13	225.62	232.32
8	320.24	309.78	311.43	310.62	311.09	320.60	348.85

A CPU-based program was also developed and compared to the GPU version. Note that, the goal of this paper is to present a GPU-based parallelization, thus CPU-based accelerations such as SIMD instructions (MMX, SSE, AVX, etc.) are not considered. In general, the performance increases with the number of threads used. All of our kernels have been implemented in the Core i5-2300; specifically speaking, we use 4 threads on 4 cores to obtain the best performance possible.

Several 3D frame models were constructed in order to evaluate the performance of the program on the GPU and CPU. Two indicators have been proposed: the total computational time and speedup. The different number of DOFs from low to high was adopted in the numerical examples. Table 4 presents the size of models used in the analysis. Note here, the parallelism was carried out by mapping the data onto a Stream Processor (SP) as a thread to execute through the kernel function. In the other words, model 7 is taken for example; each thread has a one-to-one mapping to the data which means 11,282,500 threads are running simultaneously.

First, the block size of CUDA in the GPU parallelization program is implemented for all tested cases. Due to paucity of computation time, the analysis was carried out for 100 time steps, and each time step may contain two or more Newton-Raphson iterations. Table 5 shows the relationship between block size and the time cost. Since threads should be running in groups of at least 32 (32 threads per warp) for optimal computing efficiency while using CUDA, some of the computing capability is wasted when the block size is less than 32. Thus the block size can achieve relative higher performance ranged from 32 to 256. However, limitation on the architecture of GPU device is another factor which needs to be considered. For GTX460, the available registers for each multiprocessor are 4,681 (total 32,768 registers and 7 multiprocessors) If each block uses

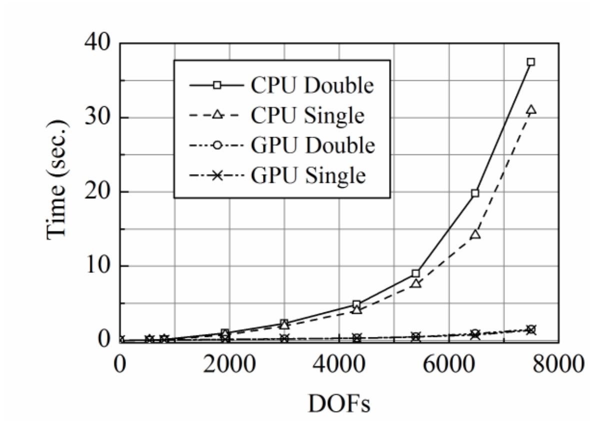


Fig. 11 Computational time for PCG solver

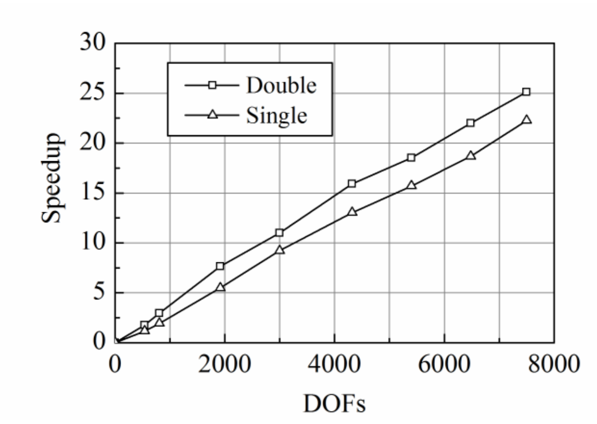


Fig. 12 Speedup for PCG solver

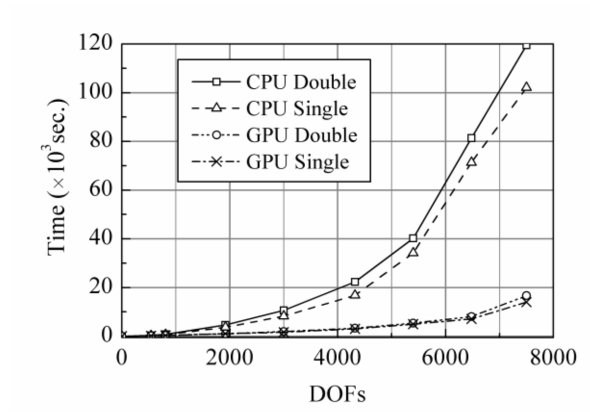


Fig. 13 Computational time for entire duration of Newmark-beta method

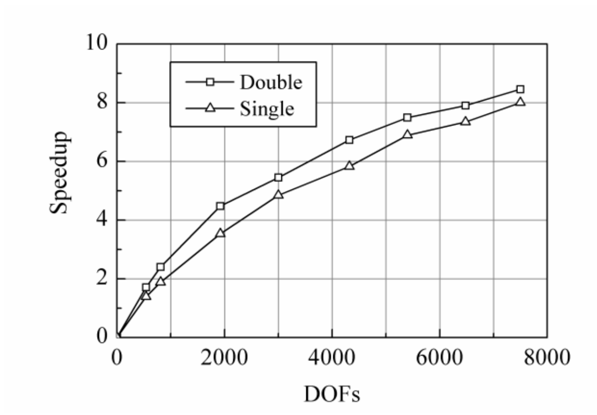


Fig. 14 Speedup for entire duration of Newmark-beta method

too many registers, the number of blocks resided on a multiprocessor will be reduced. Thus, if the block size is greater than 256, the available registers for a thread will decrease, thereby yielding lower performance for the parallel computation.

The comparison of computational time for PCG solver is shown in Fig. 11. The speedup achieved by the parallelized part of the PCG is shown in Fig. 12. As expected, all implementations of PCG on the GPU could reduce the solution time compared with the CPU. The larger number of DOFs, the bigger the difference is (see Fig. 12). It also shows that double precision calculations cost more time than single precision ones. Additionally, the achievement of maximum speedups reached are 25 and 22 times for single precision and double precision with 7500 DOFs, respectively.

Fig. 13 presents the differences of computational time for entire duration of Newmark-beta method, including the time for PCG. Similar to results of the PCG solver, both implementations achieve similar performance, but the speedup is smaller than the one obtained in the PCG part. Specifically, the speedups of double precision and single precision are 8.5 and 8 times shown in Fig. 14, respectively. This is explained by the fact that, with more integrations involved in nonlinear states, data need to be continuously transmitted from CPU to GPU or from GPU to CPU in each time-step integration (see Fig. 6). Such data transmission is relatively slow compared with the direct access in the GPU. So, such communication overheads are more time consuming. Reasonably, the communication time between GPU and CPU is trivial, furthermore, with the increase of DOFs, the communication time consuming will be not worth mentioning.

Table 6 Total computational time (in seconds) and speedup

DOFs	CPU-based program		GPU-based program		Speedup	
	single precision	double precision	single precision	double precision	single precision	double precision
10,800	190,178	247,566	17,453	18,299	10.9	13.5

An additional fine-meshed 3D frame model was computed to evaluate the performance of the developed program on the GPU-based platform. The model consists of 4,800 beam-column elements with 10,800 DOFs. The total computational time was measured. It should be noted that the total computational time refers to the total time taken by the program. The time of data input and output is also included. Table 6 lists the parameters of the total computational time and speedup. It can be found that more than 10 times speedup can be reached. We can easily imagine that, for a regular dynamic analysis, if the common method was adopted without the help of GPU and parallelized computation, the whole process might take more than two and a half day; however, it can be finished on the GPU-based platform within five hours. Also, the accuracy of our program was the same as traditional one, in some ways, ours could achieve better accuracy.

6. Conclusions

A parallel algorithm is presented for conducting nonlinear dynamic analysis of 3D frame structures on a GPU-based platform. In this work, the approach is based on the nonlinear FEM and implicit Newmark-beta integration algorithm. The PCG method was employed to solve the linear system of equations in each time-step integration. The fiber beam model was adopted in the finite element model. Also the material nonlinear is considered.

Parallelization strategies for two main implementations of Newmark-beta algorithm and PCG solver are presented in this work. In order to test he performance of program, the same analysis model was conducted on GPU and CPU respectively. The benchmark test shows that the computation accuracy of the GPU-based program is sufficient for dynamic analysis. The accuracy requirements can be satisfied by both single precision and double precision. Performance of both implementations were measured and compared with CPU implementations. The results indicate that the proposed algorithms on the GPU-based platform would be a high-speed approach for large-scale structural dynamic analysis.

Based on the proposed algorithm, the computation on the GPU shows a significant prospect. For larger problems which cannot fit in current GPU’s memory, further research efforts are needed to propose a design which allows the computation on both CPU and GPU, while reduce mutual data transfers, to proceed in parallel. Future studies also include dynamic analysis with multiple GPUs computing, the mixed precision approach, and the refine numerical section integration for improved accuracy of the analysis of RC structures.

Acknowledgments

The present work was supported by the Major International (Sino-US) Joint Research Project of the National Natural Science Foundation of China (Grant No. 51261120374), the National Natural Science Foundation of China (Grant Nos. 51278155 and 51378007), Project (HIT.NSRIF.2014093) Supported by Natural Scientific Research Innovation Foundation in Harbin Institute of Technology, Shenzhen Technology Innovation Program – Technology Development Projects (Grant No. CXZZ20140904154839135) and Shenzhen Technology Innovation Program – Technical Projects (Grant No. JSGG20150330103937411). The authors are grateful to Professor Didem Ozevin at University of Illinois at Chicago for the critical reading of the paper and the helpful comments.

References

Adeli, H. (2000), "High-performance computing for large-scale analysis, optimization, and control", *J. Aerospace Eng.*, **13**(1), 1-10.

Bathe, K.J. and Wilson, E. (1976), *Numerical mthods in finite eement analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

Belytschko, T. (1983), "An overview of semidiscretization and time integration procedures", *Computational methods for transient analysis(A 84-29160 12-64)*, Amsterdam, North-Holland, 1-65.

Benzi, M. and Tuma, M. (1999), "A comparative study of sparse approximate inverse preconditioners", *Appl. Numer. Math.*, **30**(2), 305-340.

Blakely, R.W.G. and Park, R. (1973), "Prestressed concrete sections with cyclic flexure", *J. Struct. Div.*, **99**(8), 1717-1742.

Bryan, B.A. (2013), "High-performance computing tools for the integrated assessment and modelling of social-ecological systems", *Environ. Modell. Softw.*, **39**, 295-303.

Chetverushkin, B.N., Shilnikov, E.V. and Davydov, A.A. (2013), "Numerical simulation of the continuous media problems on hybrid computer systems", *Adv. Eng. Softw.*, **60-61,** 42-47.

Eklund, A., Dufort, P., Forsberg, D. and LaConte, S.M. (2013), "Medical image processing on the GPU-Past, present and future", *Med. Image Anal.*, **17**(8), 1073-1094.

Fahmy, M.W. and Namini, A.H. (1994), "A survey of parallel nonlinear dynamic analysis methodologies", *Comput. Struct.*, **53**(4), 1033-1043.

Farhat, C. and Roux, F.X. (1991), "A method of finite element tearing and interconnecting and its parallel solution algorithm", *Int. J. Numer. Meth. Eng.*, **32**(6), 1205-1227.

Fung, T.C. (1997), "A precise time-step integration method by step-response and impulsive-response matrices for dynamic problems", *Int. J. Numer. Methods Eng.*, **40**(24), 4501-4527.

Fung, T.C. (1997), "Third-order time-step integration methods with controllable numerical dissipation", *Commun. Numer. Meth. Eng.*, **13**(4), 307-315.

Galoppo, N., Govindaraju, N.K., Henson, M. and Manocha, D. (2005), "LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware", *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, IEEE Computer Society, November.

Georgescu, S., Chow, P. and Okuda, H. (2013), "GPU acceleration for FEM-based structural analysis", *Arch. Comput. Method E.*, **20**(2), 111-121.

Göddeke, D. (2010), "Fast and accurate finite-element multigrid solvers for PDE simulations on GPU clusters", Ph.D. Dissertation, Technische Universit ät Dortmund, Fakultät für Mathematik,.

Göddeke, D. (2011), *Fast and accurate finite-element multigrid solvers for PDE simulations on GPU clusters* (Logos Verlag Berlin GmbH).

Golley, B.W. (1996), "A time-stepping procedure for structure dynamics using gauss point collocation", *Int.*

J. Numer. Meth. Eng., **39**(23), 3985-3998.

Helfenstein, R. and Koko, J. (2012), "Parallel preconditioned conjugate gradient algorithm on GPU", *J. Comput. Appl. Math.*, **236**(15), 3584-3590.

Huthwaite, P. (2014), "Accelerated finite element elastodynamic simulations using the GPU", *J. Comput. Phys.*, **257**, 687-707.

Kang, D.K., Kim, C. W. and Yang, H.I. (2014), "GPU-based parallel computation for structural dynamic response analysis with CUDA", *J. Mech. Sci. Technol.*, **28**(10), 4155-4162.

Kent, D.C. and Park, R. (1971), "Flexural members with confined concrete", *J. Struct. Div.*, **97**(7), 1969-1990.

Khronos Group (2014), The OpenCL Specification Version 2.0, Khronos OpenCL Working Group.

Komatitsch, D., Erlebacher, G., Göddeke, D. and Michéa, D. (2010), "High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster", *J. Comput. Phys.*, **229**(20), 7692-7714.

Komatitsch, D., Michéa, D. and Erlebacher, G. (2009), "Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA", *J. Parallel. Distr. Com.*, **69**(5), 451-460.

Li, H.Y., Teng, J., Li, Z.H. and Zhang, L. (2015), "Nonlinear dynamic analysis efficiency by using a GPU parallelization", *Eng. Lett.*, **23**(4), 232-238.

Mafi, R. and Sirouspour, S. (2014), "GPU-based acceleration of computations in nonlinear finite element deformation analysis", *Int. J. Numer. Methods in Bio.*, **30**(3), 365-381.

Manjuprasad, M., Gopalakrishnan, S. and Appa Rao, T.V.S.R. (2001), "Non-linear dynamic response of a reinforced concrete secondary containment shell subjected to seismic load", *Eng. Struct.*, **23**(5), 397-406.

Menegotto, M. and Pinto P.E. (1977), "Slender R.C. Compressed members in biaxial bending", *J. Struct. Div.*, **103**(3), 587-605.

Miao, X., Jin, X. and Ding, J. (2015), "A new hybrid solver with two-level parallel computing for large-scale structural analysis", *Concurr. Comp-Pract. E.*, **27**(14), 3661-3675.

Mullapudi, T.R.S. and Ayoub, A. (2013), "Analysis of reinforced concrete columns subjected to combined axial, flexure, shear, and torsional loads", *J. Struct. Eng.*, **139**(4), 561-573.

Nathan, M.N. (1959), "A method for computation of structural dynamics", *J. Eng. Mech.*, **85**(3), 67-94.

Noor, A.K. (1997), "New computing systems and future high-performance computing environment and their impact on structural analysis and design", *Comput. Struct.*, **64**(1), 1-30.

nVidia Corporation (2013a), CUDA C Programming Guide.

nVidia Corporation (2013b), CUBLAS library.

Ohsaki, M., Miyamura, T., Kohiyama, M., Hori, M., Noguchi, H., Akiba, H. and Koichi, K. (2009), "High-precision finite element analysis of elastoplastic dynamic responses of super-high-rise steel frames", *Earthq. Eng. Struct. D.*, **38**(5), 635-654.

Phoon, K.K., Toh, K.C., Chan, S.H. and Lee, F.H. (2002), "An efficient diagonal preconditioner for finite element solution of Biot's consolidation equations", *Int. J. Numer. Meth. Eng.*, **55**(4), 377-400.

Sasani, M. and Kropelnicki, J. (2008), "Progressive collapse analysis of an RC structure", *Struct. Des. Tall Spec.*, **17**(4), 757-771.

Scott, B.D., Park, R. and Priestley, M.J.N. (1982), "Stress-strain behaviour of concrete confined by overlapping hoops at low and high strain rates", *ACI J.*, **79**(1), 13-27.

Smolinski, P. (1990), "Convergence of multi-time step integration methods", *Comput. Struct.*, **35**(6), 719-724.

Spacone, E., Fillippou, F.C. and Taucer, F.F. (1996), "Fiber beam-column model for nonlinear analysis of RC frames: Part I. Formulation", *Earthq. Eng. Struct. D.*, **25**(7)**,** 711-725.

Stroud, A.H. and Secrest, D. (1966), *Gaussian quadrature formulas*, Prentice-Hall, Englewood Cliffs, New Jersey.

Turek, S., Göddeke, D., Becker, C., Buijssen, S.H. and Wobker, H. (2010), "FEAST-realization of hardware-oriented numerics for HPC simulations with finite elements", *Concurr. Comput.-Pract. E.*, **22**(16), 2247-2265.

Walpole, W.R. and Shepherd, R. (1969), "Elasto-plastic seismic response of reinforced concrete frame", *J.*

Struct. Div., **95**(ST10), 2031-2055.

Weber, D., Bender, J., Schnoes, M., Stork, A. and Fellner, D. (2013), "Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications", *Comput. Graph. Forum.*, **32**(1), 16-26.

Wei, C. and Luca, C. (2015), "New GPU computing algorithm for wind load uncertainty analysis on high-rise systems", *Wind Struct.*, **21**(5), 461-487.

Wilson, E.L., Farhoomand, I. and Bathe, K.J. (1972), "Nonlinear dynamic analysis of complex structures", *Earthq. Eng. Struct. D.*, **1**(3), 241-252.

Yagawa, G., Soneda, N. and Yoshimura, S. (1991), "A Large scale finite element analysis using domain decomposition method on a parallel computer", *Comput. Struct.*, **38**(5-6), 615-625.

Yamada, T., Ohbo, N. and Itami, H. (2008), "Three-dimensional analysis method of seismic resistance of large tunnel structure using large-scale numerical computation of soil-tunnel system", *Proceedings of the World Tunnel Congress 2008*, Agra, India, September.

Yang, D.P., Peterson, G.D. and Li, H.S. (2012), "Compressed sensing and Cholesky decomposition on FPGAs and GPUs", *Parallel Comput.*, **38**(8), 421-437.

Yang, Y.S., Hsieh, S.H. and Hsieh, T.J. (2011), "Improving parallel substructuring efficiency by using a multilevel approach", *J. Comput. Civil Eng.*, **26**(4), 457-464.

Yang, Y.S., Yang, C.M. and Hsieh, T.J. (2014), "GPU parallelization of an object-oriented nonlinear dynamic structural analysis platform", *Simul. Modell. Prac.Theo.*, **40**, 112-121.

Yassin, M.H.M. (1994), "Nonlinear analysis of prestressed concrete structures under monotonic and cycling loads", Ph.D. Dissertation, University of California, Berkeley.

CC