Advances in Engineering Software 000 (2017) 1–16

Contents lists available at ScienceDirect

Advances in Engineering Software

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

Research paper

An improved direct linear equation solver using multi-GPU in multi-body dynamics

Ji-Hyun Jung, Dae-Sung Bae*

Department of Mechanical Engineering, Hanyang University, Ansan, Gyeonggi-do 15588, Republic of Korea

ARTICLE INFO

Article history:
Received 7 May 2017
Revised 24 July 2017
Accepted 3 September 2017
Available online xxx

Keywords:
Implicit integration
Linear equation solver
Nested dissection
Multi-body dynamics
Multi-GPU

ABSTRACT

This research proposes an implementation of effective direct linear equation solver for mechanical multi-body dynamics analysis. The proposed method focuses on the solvability for any size of GPU memory and scalability for any number of GPUs by using BFS-based traversal. A multi-level tree is divided into as many sub-trees as a GPU number by using the nested dissection, each of which is assigned to each GPU. Balanced graph bisection, additional sub-trees, and work stealing lead to minimum idle GPU computing time. Numerical experiments have been performed to decide the optimal maximum block size. Three mechanical models and the other three matrices from UF collection have been solved to show the effectiveness of the proposed method. Two different kinds of 4 GPUs, GeForce GTX 460 and GTX TITAN BLACK, are involved in this experiment. The proposed method shows a good solvability even when the test GPU memory is dozens of times smaller than the required data size for numerical factorization. The proposed optimization algorithm presents a good scalability on the number of GPUs. The performance results are compared with those obtained from CHOLMOD included in SuiteSparse library.

1. Introduction

CAE (Computer Aided Engineering) has been widely utilized not only in a mechanical field but also in a number of engineering territories. Instead of a high cost experiment of real mechanical models, it has become imperative to substitute them for virtual models containing giant unknowns and to analyze them on a high performance computer system.

Modern computer system has been upgraded continuously, and the development enables larger and larger models to be solved. However, the performance improvement of a conventional computing device, CPU, has been relatively stalled by a few limitations. Accelerating ways using additional computing devices were introduced about 10 years ago, and now they are extensively studied in a variety of fields. The accelerator methodology does not need to replace the main system with a stronger one and also can control the number of computing devices when necessary.

For example, a representative CPU manufacturer, Intel, recently released a self-bootable many-core CPU [1] to cope with the accelerators such as GPUs. If we want to upgrade the computing performance of the many-core CPU, not only the same main system but also Omni-Path Fabric interconnect equipment has to be prepared. The additional system and the equipment come at a big burden. That is the reason why we pay attention to taking advantage of GPUs, the most popular accelerator, to easily make better computing performance while minimizing burdens.

There are two kinds of methods that solve linear equation [2], $\mathbf{Ax}=\mathbf{b}$, for implicit integration on mechanical flexible dynamic analysis. One is an iterative method [3], which navigates a solution by repetitive operations of a simple vector-vector and matrix-vector combination. This method has advantages that it needs highly small amount of data storage space and its implementation is considerably easy [4]. The feature of low storage space is particularly suitable for GPU device that has limited ‘global memory space’, which is shortly called ‘GPU memory’ hereinafter, and it has been applied to a wide range of industries, mechanical [5–9], electrics simulation [10], finance [11] areas and the like. Meanwhile, it requires a high quality pre-conditioner for fast convergence of the solution [12,13]. In addition, there is a potential that the solution may not be found at times. Those drawbacks force another solving way, direct method.

Although the direct method [14] has also some disadvantages that it calls for far more computational complexity as well as much larger data storage space than the iterative one, it could obtain a solution in most cases with finite number of floating-point operations unless the matrix are exactly or almost singular. The method can conduct stable finite element analysis [15–17]. There have been several studies on implementing the direct linear solver on a GPU [18–22], but those studies are less activated than that of the iterative method because its implementation has to overcome

* Corresponding author.
E-mail address: dsbae@hanyang.ac.kr (D.-S. Bae).

https://doi.org/10.1016/j.advengsoft.2017.09.001
0965-9978/© 2017 Elsevier Ltd. All rights reserved.

hardship such as slow PCI-Express link speed and independent memory space. And most of the studies focus on only a single GPU environment. CHOLMOD [23] is a representative direct linear equation solver routine supporting multi-GPU. As a result, the performance results of the proposed method are compared to those obtained from CHOLMOD.

The rest of this paper is organized as follows: Section 2 summarizes an implicit integration method for the constrained mechanical systems. BFS-based nested dissection and its numerical factorization implementation are presented. Section 3 deals with a multi-level tree segmentation method to develop a parallel algorithm for multi-GPU. The parallel algorithm is optimized to have an effective GPU scalability. The various block sizes for the numerical factorization and parallel algorithm have been numerically tested. A selected optimal maximum block size has been applied to the proposed parallel algorithm in Section 4. The performance results are discussed and compared to those obtained from CHOLMOD. Conclusions and future work are drawn in Section 5.

2. Linear equation solver

2.1. Equation of motion

The constrained mechanical system is often represented as DAE (differential-algebraic equation). A solution of DAE is more difficult than that of ODE (ordinary differential equation). This study chooses an implicit numerical integration method to solve DAEs. Kinematic constraints including their derivatives and equations of motion are solved simultaneously [24]. The equations of motion for a constrained mechanical system can be described as

$$\mathbf{v} - \dot{\mathbf{q}} = \mathbf{0} \tag{2.1}$$

$$\mathbf{F}(\mathbf{q},\ \mathbf{v},\ \mathbf{a},\ \boldsymbol{\lambda}) = \mathbf{0} \tag{2.2}$$

$$\boldsymbol{\Phi}(\mathbf{q},\ t) = \mathbf{0} \tag{2.3}$$

where $\mathbf{q}$ is the generalized coordinate vector in Euclidean space $\mathbf{R^n}$, $\mathbf{v}$ is the generalized velocity vector in $\mathbf{R^n}$, $\mathbf{a}$ is the generalized acceleration vector in $\mathbf{R^n}$, $\boldsymbol{\lambda}$ is the Lagrange multiplier vector for constraints in $\mathbf{R^m}$, $\boldsymbol{\Phi}$ represents the position level constraint vector in $\mathbf{R^m}$, and the Jacobian $\boldsymbol{\Phi}_\mathbf{q} \in \mathbf{R}^{\mathbf{m}\times\mathbf{n}}$ is assumed to have full row-rank. Successive differentiations of Eq. (2.3) yield velocity and acceleration level constraints,

$$\ddot{\boldsymbol{\Phi}} \cdot (\mathbf{q},\ \mathbf{v},\ t) = \boldsymbol{\Phi}_\mathbf{q}\mathbf{v} + \boldsymbol{\Phi}_t = \boldsymbol{\Phi}_\mathbf{q}\mathbf{v} - \boldsymbol{\nu} = \mathbf{0} \tag{2.4}$$

$$\ddot{\boldsymbol{\Phi}}(\mathbf{q},\ \mathbf{v},\ \mathbf{a},\ t) = \boldsymbol{\Phi}_\mathbf{q}\mathbf{a} + \frac{d}{dt}(\boldsymbol{\Phi}_\mathbf{q})\mathbf{v} + \boldsymbol{\Phi}_{tt} = \boldsymbol{\Phi}_\mathbf{q}\mathbf{a} - \boldsymbol{\gamma} = \mathbf{0} \tag{2.5}$$

Equations from 2.1 to 2.5 comprise a system of over-determined differential-algebraic equations (ODAE). An algorithm based on backward differentiation formulas (BDF) to solve ODAE is described as

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \mathbf{F}(\mathbf{x}) \\ \ddot{\boldsymbol{\Phi}} \\ \dot{\boldsymbol{\Phi}} \\ \boldsymbol{\Phi} \\ \mathbf{U}_1^\mathrm{T}\left(\frac{h}{b_0}\mathbf{R}_1\right) \\ \mathbf{U}_2^\mathrm{T}\left(\frac{h}{b_0}\mathbf{R}_2\right) \end{bmatrix} = \begin{bmatrix} \mathbf{F}(\mathbf{q},\ \mathbf{v},\ \mathbf{a},\ \boldsymbol{\lambda}) \\ \boldsymbol{\Phi}_\mathbf{q}\mathbf{a} - \boldsymbol{\gamma} \\ \boldsymbol{\Phi}_\mathbf{q}\mathbf{v} - \boldsymbol{\nu} \\ \boldsymbol{\Phi}(\mathbf{q},\ t) \\ \mathbf{U}_1^\mathrm{T}\left(\frac{h}{b_0}\mathbf{a} - \mathbf{v} - \zeta_1\right) \\ \mathbf{U}_2^\mathrm{T}\left(\frac{h}{b_0}\mathbf{v} - \mathbf{q} - \zeta_2\right) \end{bmatrix} = \mathbf{0} \tag{2.6}$$

where $\zeta_1 \equiv \frac{1}{b_0}\sum_{i=1}^{k}\mathbf{b}_i\mathbf{v}_{n-i}$ and $\zeta_2 \equiv \frac{1}{b_0}\sum_{i=1}^{k}\mathbf{b}_i\mathbf{q}_{n-i}$ in which k is the order of integration, and the $\mathbf{b}_i$ are BDF coefficients.

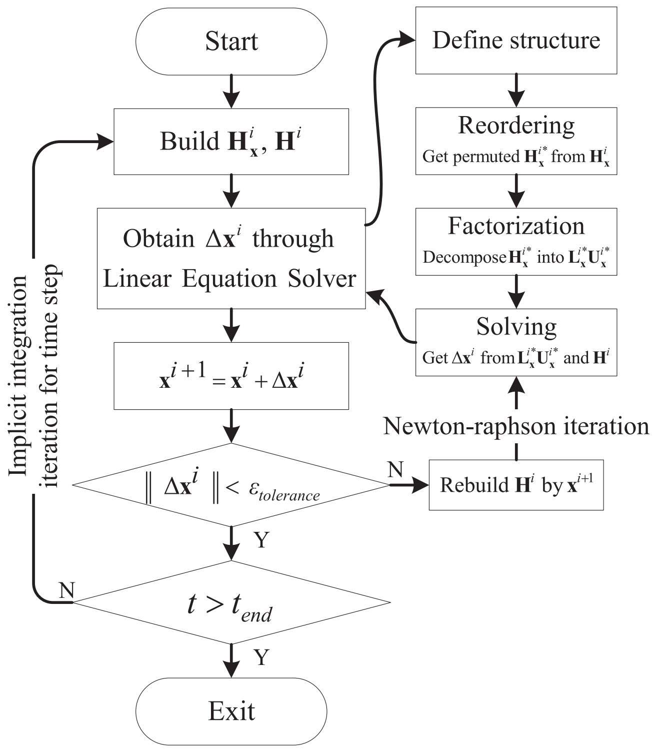
Start
Build $\mathbf{H}_\mathbf{x}^i$, $\mathbf{H}^i$
Obtain $\Delta\mathbf{x}^i$ through Linear Equation Solver
$\mathbf{x}^{i+1} = \mathbf{x}^i + \Delta\mathbf{x}^i$
$\|\Delta\mathbf{x}^i\| < \varepsilon_{tolerance}$
N
Y
$t > t_{end}$
N
Y
Exit
Implicit integration iteration for time step
Define structure
Reordering
Get permuted $\mathbf{H}_\mathbf{x}^{i*}$ from $\mathbf{H}_\mathbf{x}^i$
Factorization
Decompose $\mathbf{H}_\mathbf{x}^{i*}$ into $\mathbf{L}_\mathbf{x}^{i*}\mathbf{U}_\mathbf{x}^{i*}$
Solving
Get $\Delta\mathbf{x}^i$ from $\mathbf{L}_\mathbf{x}^{i*}\mathbf{U}_\mathbf{x}^{i*}$ and $\mathbf{H}^i$
Newton-raphson iteration
Rebuild $\mathbf{H}^i$ by $\mathbf{x}^{i+1}$

Fig. 1. Flow chart of DAE solution scheme through implicit numerical integration.

$\mathbf{x} = [\mathbf{a}^\mathrm{T}\quad \mathbf{v}^\mathrm{T}\quad \mathbf{q}^\mathrm{T}\quad \boldsymbol{\lambda}^\mathrm{T}]^\mathrm{T}$ and the columns of $\mathbf{U}_i \in \mathbf{R}^{n\times(n-m)}$ $(i=1,\ 2)$ constitute bases for the parameter space of the position and velocity level constraints.

The matrices $\mathbf{U}_i$ are chosen so that $\begin{bmatrix}\boldsymbol{\Phi}_\mathbf{q} \\ \mathbf{U}_i^\mathrm{T}\end{bmatrix}$ is nonsingular. Therefore, the parameter space spanned by the columns of $\mathbf{U}_i$ and the subspace spanned by the columns of $\boldsymbol{\Phi}_\mathbf{q}^\mathrm{T}$ constitute the entire space $\mathbf{R^n}$.

Eq. (2.6) can be solved since the number of equations and that of unknowns are the same. Newton's numerical method can be applied to acquire the solution $\mathbf{x}$ [25].

$$\mathbf{H}_\mathbf{x}^i\Delta\mathbf{x}^i = -\mathbf{H}^i \tag{2.7}$$

$$\mathbf{x}^{i+1} = \mathbf{x}^i + \Delta\mathbf{x}^i \quad i = 1,\ 2,\ 3,\ \ldots \tag{2.8}$$

Eq. (2.7) is represented as $\mathbf{Ax}=\mathbf{b}$, a linear equation problem. 4 steps are needed to solve the problem in direct method. It defines sparse matrix structure, performs appropriate reordering to contain the outbreak of fill-ins in factorization process, and does factorization numerically. Finally, a solution is acquired by fore-, backward substitution of a RHS (Right Hand Side). Fig. 1 shows a work flow of an implicit numerical integration to solve DAEs along with the linear equation solver steps.

2.2. Nested dissection

The numerical factorization is the most time-consuming step among all the steps of linear equation solver. The time consumption of the numerical factorization step is greatly affected by adopted reordering method.

A mechanical system model consists of various kinds of elements. The model can be mapped as a Graph (G) containing Vertex (V) and Edge (E) [26]. Fig. 2(a) depicts a model and its corresponding Graph. A group of vertices splitting a graph into