11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
E. Oñate, J. Oliver and A. Huerta (Eds)

HEURISTIC METHOD OF DYNAMIC STRESS ANALYSIS IN MULTIBODY SIMULATION USING HPC

V.V. GETMANSKIY*, A.S. GOROBTSOV*, T.D. ISMAILOV* AND A.E. ANDREEV*

*Volgograd State Technical University,
Faculty Electronics and Computer Science,
Volgograd, Lenin avenue, 28, 400005, Russia,
e-mail: vm@vstu.ru

Key Words: *Dynamic stress analysis, Multibody simulation, Ambiguous constraints, Discrete elements, Domain decomposition, Parallel computing.*

Abstract. A parallel method of stress-strain simulation of single body in multibody dynamic model is proposed in this paper. Method is based on discrete elements approximation of deformable body and domain decomposition. Multithreaded implementation using POSIX threads library with custom synchronization primitives gives a relatively good speedup of parallel computing on NUMA architecture in comparison with OpenMP implementation. Simulation of 13M elements mesh is performed on up to 32 cores. The results and efficiency estimation of concurrent simulation are presented.

1 INTRODUCTION

Actual task in vehicle design is a stress-strain analysis of vehicle parts with maximal dynamic loads. Traditional stress estimation approaches are based on FEM methods which compute only quasi-static stress distribution and give poor results in nonlinear dynamic cases.

For example a lower arm model in truck suspension system is considered. It is connected to vehicle frame and wheel hub. Reactions from joints to these bodies are used for determining the stress distribution. The reactions are obtained from multibody simulation of full vehicle. Weak coupling of multibody solver and stress solver is used in proposed method [1]. It uses separate solvers with transferring data during simulation and assigning corresponding data to coupled parameters [2, 3]. Coupled parameters are reaction forces which are transferred from multibody model to stress submodel at each time step. Multibody solver (MBS) is based on augmented Lagrangian formulation with natural coordinates [4, 5].

Parallelization of stress solver using OpenMP is limited by shared memory systems and loses in performance in case of NUMA architecture. Iterative procedure allows loops paralleling which go along with threads reallocation on each iteration. Overheads caused by threads reallocation are analysed for example in work [6]. Small amount of time for parallel blocks in comparison with total simulation time does not allow using OpenMP efficiently. As a solution to the problem stated above it was developed a parallel stress-strain simulation method based on domain decomposition and custom threads synchronization routines.

2 DISCRETE ELEMENTS METHOD FOR DEFORMABLE BODY REPRESENTATION

A lower arm in vehicle suspension system has large dynamic loads when the vehicle runs on random road profile. Some consideration for lower arm design using FEM simulation is presented in work [7]. However quasi-static approach lacks in case of dynamic loads. In proposed method deformable body is approximated by rigid discrete elements. Computational domain is regular orthogonal mesh based on CAD geometry of the part. Each mesh node is a rigid discrete element with 6 degrees of freedoms. Each element is connected to adjacent elements by flexible joint. Regular discrete elements structure greatly simplifies the construction of differential equation system in natural coordinates. The inertia matrix is diagonal and all elements are described by similar ordinary differential equations to within number of links to adjacent elements. Flexible body has a rigid counterpart in multibody model which is called reference body. The system of equations for flexible body has a following form in coordinate system of reference body:

$$\mathbf{M}\ddot{\mathbf{y}} = \mathbf{q}(\dot{\mathbf{y}}, \mathbf{y}, t) - \mathbf{M}\mathbf{a}(t) + \mathbf{s}(\dot{\mathbf{y}}, \mathbf{y}), \tag{1}$$

where $\mathbf{y}$ is a vector of discrete elements coordinates, $\mathbf{M}$ is a matrix of inertia for set of discrete elements, $\mathbf{a}$ is a vector constructed of reference body acceleration vector components, $\mathbf{s}(\dot{\mathbf{y}}, \mathbf{y})$ is a stabilizing component. Reference body acceleration is obtained from multibody model. The function $\mathbf{q}(\dot{\mathbf{y}}, \mathbf{y}, t)$ describes the forces between rigid elements. Stress values are computed based on the values of the forces on each iteration. Random perturbing effect makes flexible body model unstable. Additional stabilizing component $\mathbf{s}(\dot{\mathbf{y}}, \mathbf{y})$ is introduced in Equation (1) to constrain the position of discrete elements. Right-hand sides of the equation are completed with inertia forces taking into account motion of the reference body in multibody model.

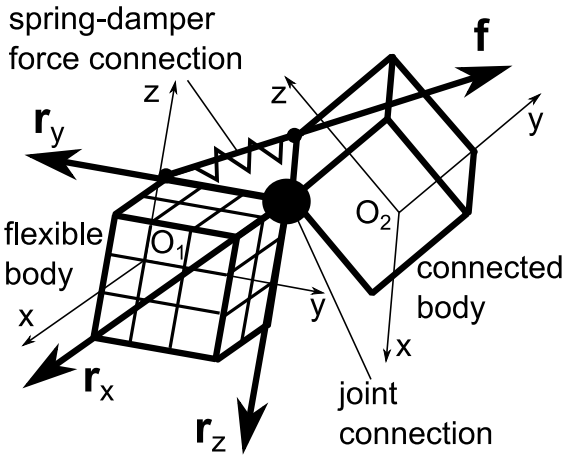

Figure 1: Flexible body to rigid body connection

Flexible body is connected to multibody model with joints. To complete the equation it is necessary to add reaction forces in flexible joints to subset of Equations (1) corresponded to

solid elements on contact surfaces. There are two kinds of joints shown in Figure 1: spring-damper force $\mathbf{f}_1 = \mathbf{f}$ and stiff joint with reaction force $\mathbf{f}_2 = (\mathbf{r}_x, \mathbf{r}_y, \mathbf{r}_z)$. The equation of motion for boundary discrete element has a following form in case of uniform distribution of reaction force along boundary:

$$\mathbf{M}_i \ddot{\mathbf{y}}_i = \mathbf{f}(t)/|F| + \mathbf{q}_i(\dot{\mathbf{y}}, \mathbf{y}, t) - \mathbf{M}_i \mathbf{a}_i(t) + \mathbf{s}_i(\dot{\mathbf{y}}, \mathbf{y}), \; i \in F \tag{2}$$

where F is a set of node indices of boundary with applied reaction $\mathbf{f}$. So the reference body data obtained from multibody solver is enough to calculate forces of inertia and reactions in joints.

Equation (1) is solved using Runge-Kutta 4-th order method (RK4). The data of reference body is transferred to stress solver on sync time steps. Numerical integration time steps of multibody solver (Δt_{main}) and stress solver (Δt_{sub}) are different in general case. For simplification it is posed that time steps are divisible. If $\Delta t_{\text{sub}} < \Delta t_{\text{main}}$, then $\Delta t_{\text{main}} = N \cdot \Delta t_{\text{sub}}$, where N is an integer multiplier. There is general time counter with minimal time step $\Delta t = \min(\Delta t_{\text{sub}}, \Delta t_{\text{main}})$ and coupling time counter with maximal time step $\Delta t_{\text{upd}} = \max(\Delta t_{\text{sub}}, \Delta t_{\text{main}})$. Solver algorithm is presented below.

The following serial algorithm is implemented and used for simulation:

$t \leftarrow 0$, $t_{\text{sub}} \leftarrow 0$, $t_{\text{main}} \leftarrow 0$, $\Delta t \leftarrow \min(\Delta t_{\text{sub}}, \Delta t_{\text{main}})$, $\Delta t_{\text{upd}} \leftarrow \max(\Delta t_{\text{sub}}, \Delta t_{\text{main}})$

While $t < T$

  If $t = t_{\text{main}}$ Then

    MBS iteration solving for current time t

    $t_{\text{main}} \leftarrow t_{\text{main}} + \Delta t_{\text{main}}$

  If $t = t_{\text{upd}}$ Then

    Right-hand side update for stress-strain solver

    $t_{\text{upd}} \leftarrow t_{\text{upd}} + \Delta t_{\text{upd}}$

  If $t = t_{\text{sub}}$ Then

    Stress-strain solver iteration for time t_{sub}

    $t_{\text{sub}} \leftarrow t_{\text{sub}} + \Delta t_{\text{sub}}$

  Writing simulation results

  $t \leftarrow t + \Delta t$

Each iteration of stress solver gives computed positions of discrete elements. They are used to obtain displacements. Stresses in deformable body in each discrete element can be calculated using formula

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{bmatrix} = \begin{bmatrix}\psi & \lambda & \lambda & 0 & 0 & 0\\ \lambda & \psi & \lambda & 0 & 0 & 0\\ \lambda & \lambda & \psi & 0 & 0 & 0\\ 0 & 0 & 0 & \mu & 0 & 0\\ 0 & 0 & 0 & 0 & \mu & 0\\ 0 & 0 & 0 & 0 & 0 & \mu\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{33}\\ 2\varepsilon_{23}\\ 2\varepsilon_{13}\\ 2\varepsilon_{12}\end{bmatrix}, \tag{3}$$

where σ_{ij} are stress tensor components, ε_{ij} are strain tensor components, $\psi = \lambda + 2\mu$, $\lambda = E \cdot \nu / (1 - \nu^2)$, $\mu = E/(2 \cdot (1 + \nu))$, E is an elasticity modulus. Equivalent tensile stresses are calculated using von Mises criterion:

$$\sigma = \sqrt{0.5 \sum_{i=1}^{2} \sum_{j=i+1}^{3} \left((\sigma_{ii} - \sigma_{jj})^2 + 6\sigma_{ij}^2 \right)}. \tag{4}$$

3 PARALLEL STRESS-STRAIN SOLVER

3.1 Parallel algorithm

Overlapping domain decomposition is used for parallel algorithm. The positions of inner nodes of each subdomain are solved correctly in single iterations. The positions of outer nodes which overlap are solved incorrectly but they have corresponding inner nodes in adjacent subdomain. It is required to exchange data from inner elements of adjacent subdomains to outer elements to correct the positions. Each subdomain iterates independently except data exchange steps.

The equation order reduction is used while integrating Equation (1). In this case the equation is written in simplified form as

$$\dot{\mathbf{x}} = \tilde{\mathbf{q}}(t, \mathbf{x}), \tag{5}$$

where $\mathbf{x}$ is a new vector of unknowns, $\mathbf{x} = (\mathbf{y} \quad \dot{\mathbf{y}})^{\mathrm{T}}$, $\tilde{\mathbf{q}}$ is a new right-hand side function.

For integration time step of h RK4 formulas for ODE system are

$$\mathbf{x}_{i+1} = \mathbf{x}_i + 1/6 \cdot (\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4), \tag{6}$$

where

$\mathbf{k}_1 = h\tilde{\mathbf{q}}(t_i, \mathbf{x}_i)$,

$\mathbf{k}_2 = h\tilde{\mathbf{q}}(t_i + h/2, \mathbf{x}_i + \mathbf{k}_1/2)$,

$\mathbf{k}_3 = h\tilde{\mathbf{q}}(t_i + h/2, \mathbf{x}_i + \mathbf{k}_2/2)$,

$\mathbf{k}_4 = h\tilde{\mathbf{q}}(t_i + h, \mathbf{x}_i + \mathbf{k}_3)$.

Calculation of each $\mathbf{k}_i$ requires computing right-hand sides of Equation (5). It is necessary to exchange data after each right-hand side calculation to take into account adjacent subdomains. So it is necessary 4 barrier synchronizations after each of $\mathbf{k}_i$ computing.

Parallel algorithm implies multibody solver and each subdomain solvers in separate threads. It is stated below.

Parallel algorithm

$t \leftarrow 0,\ t_{\text{sub}} \leftarrow 0,\ t_{\text{main}} \leftarrow 0,\ \Delta t \leftarrow \min(\Delta t_{\text{sub}}, \Delta t_{\text{main}}),\ \Delta t_{\text{upd}} \leftarrow \max(\Delta t_{\text{sub}}, \Delta t_{\text{main}})$

MBS thread

While $t < T$
    If $t = t_{\text{main}}$ Then
        MBS iteration solving for current time t
        $t_{\text{main}} \leftarrow t_{\text{main}} + \Delta t_{\text{main}}$
    If $t = t_{\text{upd}}$ Then
        Sending reactions to submodels
        Barrier
    Writing simulation results
    $t \leftarrow t + \Delta t$

Submodel thread

While $t < T$
    If $t = t_{\text{upd}}$ Then
        Barrier
        Right-hand side update for stress-strain solver
        $t_{\text{upd}} \leftarrow t_{\text{upd}} + \Delta t_{\text{upd}}$
    If $t = t_{\text{sub}}$ Then
        Stress-strain solver iteration for time t_{sub}
        For $i \leftarrow 1$ To 4 Do
            $\mathbf{k}_i$ calculating
            Data exchange
            Barrier
        $\mathbf{y}(t)$ calculating
        $t_{\text{sub}} \leftarrow t_{\text{sub}} + \Delta t_{\text{sub}}$
    Writing simulation results
    $t \leftarrow t + \Delta t$

3.2 Synchronization

Data exchange is organized using sync table. Sync has a structure of adjacency matrix for subdomains and contains node number from which data is send and to which it is received.

The rows of sync table for sending data contain indices of nodes from which data is send from subdomain with number correspond to row number.

The rows of sync table for receiving data contain indices of nodes to which data is received to subdomain with number correspond to row number.

At exchange step data is packed to buffer before send using addresses from i-th row of sync table for i-th parallel process and unpacked to addresses in subdomain solver memory after receive.

3.3 Domain decomposition

Important characteristic of developed method is load balancing and minimization of data exchange. Both of these tasks are solved by domain decomposition methods. Two methods were researched: recursive coordinate bisection and multilevel methods for graph partitioning described in work [8].

First geometrical method is suited for convex geometry. Orthogonality of the mesh gives a plane sections with minimal number of cut links in case of same amount of nodes in cut boundary. But this method does not give minimal number of total cut links. Method does not guarantee the connectivity of subdomain in concave cases. Some modifications for connectivity also don't give a solution in general case. Decomposition to overlapping subdomains makes the mathematical formulation more simple and easy to implement for parallel computing. The same approach is used in parallel implementation of FEM simulation [9]. Discrete elements approach also can be used in fatigue analysis which is usually accompanied with domain decomposition [10].

Second method of multilevel graph partitioning solves NP task of edge cut minimization with relative amount of nodes in each part restrictions. Shortly algorithm consists of 3 steps:

- Source graph is compressed by combining vertices until several hundred vertices remains. Random matching is used or heavy edge matching in case of weighted edges is used.
- Compressed graph is decomposed by some decomposition methods, for example recursive bisection.
- Compressed graph is unpacked to graph with initial amount of nodes with conserving of decomposition. Kernighan-Lin heuristic is used usually for refining the parts during unpack [11].

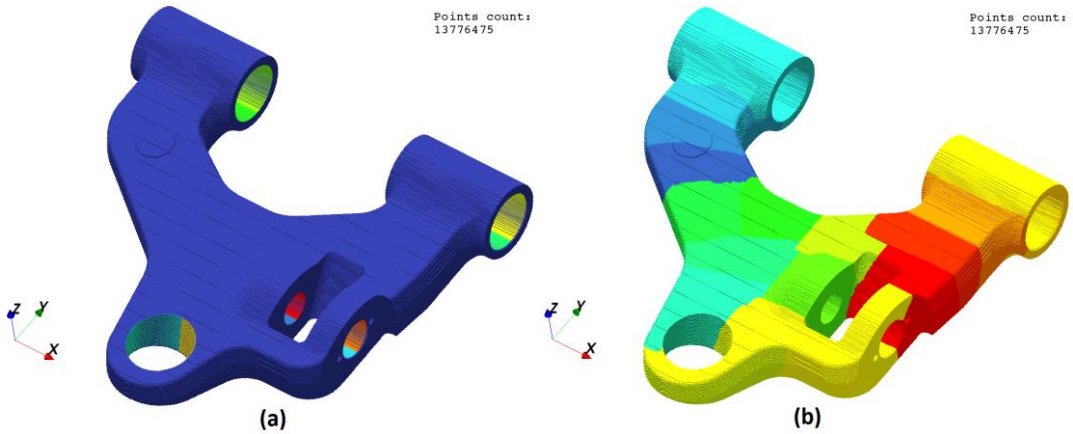

Figure 2: Suspension lower arm model before (a) and after (b) decomposition

So a multilevel algorithm which is implemented in METIS library is used for decomposition. Decomposition results for lower arm model of 13M elements are shown in Figure 2.

Imbalance and redundancy factors are used for quality estimation of decomposition. Redundancy R_n and imbalance D_n for decomposition of model of N nodes to n submodels are calculated as follows:

$$R_n = \sum_{i=1}^{n} \frac{N_i}{N},\ D_n = \max_i \left(\frac{|\ N_i - N \cdot P_i\ |}{N_i} \right), \tag{7}$$

where P_i – required relative amount of nodes in submodel i, N_i – number of nodes in submodel i.

Redundancy and imbalance in worst case of 32 subdomains are $R_{32} = 3\%$, $D_{32} = 4\%$.

4 MULTITHREADED IMPLEMENTATION

For parallelization it was used *pthreads* library.

Parallel implementation of the program uses 2 custom synchronization primitives:

- barrier for specified number of threads;
- signal without waiting.

Several multithreaded synchronization strategies can be found in work [12].

4.1 Barrier

This synchronization primitive ensures that the code which is situated in the program after the method *Wait* will not be executed by the current thread until the number of waiting threads in this method does not reach N. Such implementation allows the reuse of the barrier required for the correct work of the cyclic part of the program. The barrier prevents premature execution of program code located after a synchronization point providing equal performance of multiple threads under unbalanced loading.

Wait method algorithm
Lock *mutex*
Increase counter n: $n \leftarrow n+1$
If $n < N$ Then
    Wait for *condition* signalizing and unlock *mutex*
Else
    Reset counter n: $n \leftarrow 0$
    Signalize *condition* to waiting threads
Unlock *mutex*

4.2 Signal without waiting

This synchronization primitive extends the functionality of a conventional variable (condition) for use in the cyclic part of the program. Signal without waiting ensures that the thread that calls method *Wait* will wait until the number of calls of this method by that thread

does not match the number of calls of method *Signal* by the signalling thread. Signal without waiting prevents the execution of the program code located after the synchronization point by the waiting threads until receiving a signal from the signalling thread.

Wait method algorithm
Lock *mutex*
Get identifier i of the current thread
If the map *counters* does not contain key i
    Add pair $\langle i, 0 \rangle$ to the map *counters*
Find the counter n_w in the map *counters* by the key i
Increase the counter n_w: $n_w \leftarrow n_w + 1$
If $n_w = 1$
    Increase the counter n: $n \leftarrow n + 1$
If $n_w > n_s$
    If the flag f is not set: $f = \text{false}$
        Set the flag f: $f \leftarrow \text{true}$
    Wait for *condition* signalizing and unlock *mutex*
Else
    If $n = N$
        Reset the counter n: $n \leftarrow 0$
        Decrease the counter n_s: $n_s \leftarrow n_s - 1$
        Decrease all counters in the map *counters*
Unlock *mutex*

Signal method algorithm
Lock *mutex*
Increase the counter n_s: $n_s \leftarrow n_s + 1$
If the flag f is not set: $f = \text{true}$
    Reset the flag f: $f \leftarrow \text{false}$
    Signalize *condition* to waiting threads
Else
    If $n = N$
        Reset the counter n: $n \leftarrow 0$
        Decrease the counter n_s: $n_s \leftarrow n_s - 1$
        Decrease all counters in the map *counters*
Unlock *mutex*

5 RESULTS

Meshes with different number of nodes are used for simulation. It is necessary to use high detailed meshes of several million elements for detailed geometry description by orthogonal mesh.

Simulation was performed on Core-i7 920 platform and on 2-socket Opteron 6272 2x16 cores platform. The first platform has turbo-boost and hyper-threading capabilities. The second platform has NUMA architecture.

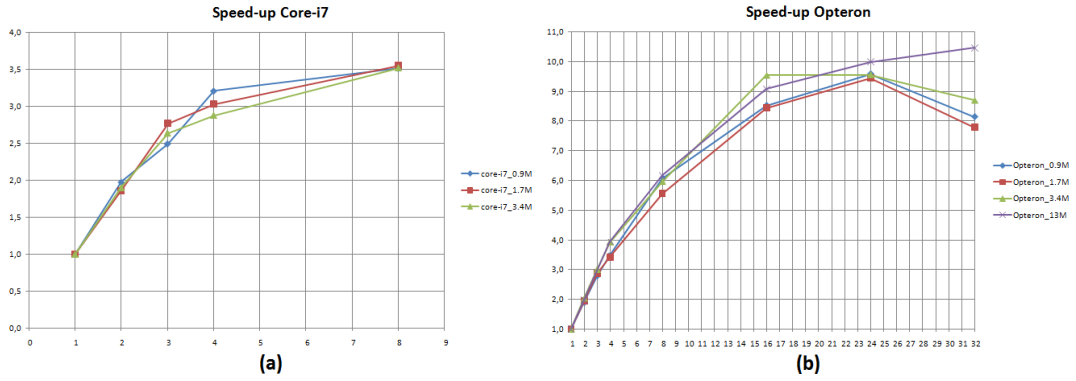

Figure 3: Speedup of multithreaded simulation on platform with single 4-core CPU Intel Core i7-920 @ 2,67GHz (a) and two 16-core CPUs AMD Opteron 6272 @ 2.1GHz (b)

Speedup results are presented in Figure 3. Efficiency for 2–4 cores is almost perfect on NUMA platform. It is decreases to 75% on 8 cores and to 25–30% on 32 cores. Efficiency decreases faster to 80% on 4 cores of Core-i7 CPU, and to 44% using 8 threads (with hyper-threading).

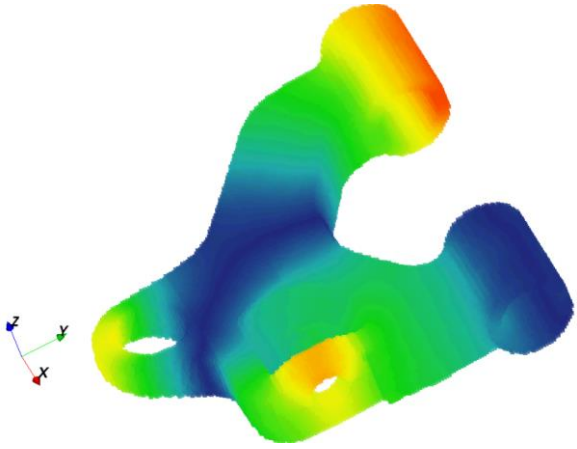

Figure 4: Suspension lower arm stress-strain simulation results

The obtained speedup is well correlated with the research on custom OpenMP implementation for NUMA.

6 CONCLUSION

The proposed method gives quite good speedup on shared memory platform with NUMA. Efficiency decreases with increasing of number of threads but the speedup of 10 is achieved on 32 cores. A custom thread handling significantly reduces overheads on NUMA-architecture in comparison with OpenMP. Parallel stress-strain simulation method is also applicable to distributed memory system such as clusters and can be implemented using MPI.

REFERENCES

[1] V.V. Getmanskiy, A.S. Gorobtsov, E.S. Sergeev, T.D. Ismailov, O.V. Shapovalov, "Concurrent simulation of multibody systems coupled with stress-strain and heat transfer solvers", Journal of Computational Science, 3(6), 492-497, 2012.

[2] M. Arnold, "Numerical methods for simulation in applied dynamics", M. Arnold, W. Schiehlen, (Editors), Simulation Techniques for Applied Dynamics, 191-246, Springer Vienna, 2008.

[3] R. Kübler, W. Schiehlen, "Two Methods of Simulator Coupling", Mathematical and Computer Modelling of Dynamical Systems, 6(2), 93-113, 2000.

[4] E. Bayo, J. Garcia de Jalon, M.A. Serna, "A modified Lagrangian formulation for the dynamic analysis of constrained mechanical systems", Comput. Methods Appl. Mech. Eng., 71(2), 183-195, 1988.

[5] M. Valasek, Z. Sika, O. Vaculin, "Multibody formalism for real-time application using natural coordinates and modified state space", Multibody System Dynamics, 17, 209-227, 2007.

[6] P. Carribault, M. Pérache, H. Jourdren, "Enabling low-overhead hybrid MPI/OpenMP parallelism with MPC", Proc. of the 6th international conference on Beyond Loop Level Parallelism in OpenMP accelerators, Tasking and more, 1-14, Springer-Verlag, Berlin, Heidelberg, 2010.

[7] M.M. Rahman, M.M. Noor, K. Kadirgama, M.A. Maleque, R.A. Bakar, "Modeling, analysis and fatigue life prediction of lower suspension arm", Advanced Materials Research, 264, 1557-1562, 2011.

[8] G. Karypis, V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning", Proc. of the 1998 ACM/IEEE conference on Supercomputing IEEE Computer Society, 1-13, Washington, DC, USA, 1998.

[9] J. Zhang, L. Zhang, H. Jiang, "An Implementation Method of Parallel Finite Element Computation Based on Overlapping Domain Decomposition", Proceedings of the Second international conference on High Performance Computing and Applications, 563-570, 2010.

[10] L.C. Li, C.A. Tang, G. Li, S.Y. Wang, Z.Z. Liang, Y.B. Zhang, "Numerical Simulation of 3D Hydraulic Fracturing Based on an Improved Flow-Stress-Damage Model and a Parallel FEM Technique", Rock Mechanics and Rock Engineering, 45(5), 801-818, 2012.

[11] B.W. Kernighan, S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", Bell Sys. Tech. J., 49(2), 291-308, 1970.

[12] A.B. Downey, "The little book of semaphores 2nd edition", CreateSpace, 2009.