Journal of MECHANICAL ENGINEERING
– Strojnícky časopis

VOLUME 66, NO 2, 2016

Print ISSN 0039-2472
On-line ISSN 2450-5471

pp. 67 – 78
DOI:10.1515/scjme-2016-0020

PARALLEL COMPUTATION ON MULTICORE PROCESSORS USING EXPLICIT FORM OF THE FINITE ELEMENT METHOD AND C++ STANDARD LIBRARIES

REK Václav[1], NĚMEC Ivan[2]

[1] Brno University of Technology, Faculty of Civil Engineering, Department of Structural Mechanics, Veveří 331/95, 602 00 Brno, Czech Republic, e-mail: RekV@seznam.cz
[2] Brno University of Technology, Faculty of Civil Engineering, Department of Structural Mechanics, Veveří 331/95, 602 00 Brno, Czech Republic, e-mail: nemec@fem.cz

Abstract: In this paper, the form of modifications of the existing sequential code written in C or C++ programming language for the calculation of various kind of structures using the explicit form of the Finite Element Method (Dynamic Relaxation Method, Explicit Dynamics) in the NEXX system is introduced. The NEXX system is the core of engineering software NEXIS, Scia Engineer, RFEM and RENEX. It has the possibilities of multithreaded running, which can now be supported at the level of native C++ programming language using standard libraries. Thanks to the high degree of abstraction that a contemporary C++ programming language provides, a respective library created in this way can be very generalized for other purposes of usage of parallelism in computational mechanics.

KEYWORDS: Finite Element Method, Parallel Computing, C++ Standard Libraries

1 Introduction

Thanks to rapid advances in computer technology in the field of multicore and multiprocessor technology, during the last decade a lot of attention has been devoted to the parallel processing of data in many scientific and industrial sectors. It is in contrast with the past decades when an increase in computing power, especially in personal computers, was achieved by increasing the clock speed of processors. This type of technological evolution has its limitations compared to multi-processors and multi-core computer architectures respectively. Many older applications designed to run sequentially have begun to become obsolete, mainly due to the performance of available hardware.

As in the past, and even now, development tools for the development of software applications are slightly lagging behind the choices of available hardware. Alternatives of how to benefit from multi-core processors were commercial technologies from the company Intel, or other free available technologies. Another possibility was to use an application interface provided directly for the respective operating system such as Win32 API [1] for Windows or POSIX [2] for Unix-like operating systems, which are often quite cumbersome and limited to the possibilities of C programming language [8].

Since the year 2011, when the new standard of C++ programming language was introduced, developers have been given the possibility to use threads and all other necessary. resources to support thread synchronization on the level of the native programming language using its standard libraries [3].

Using the new version of the C++ programming language has been possible for some time in Microsoft Compiler since the Microsoft Visual Studio 2010 IDE (abbrev. for Integrated Development Environment) or in some freely available compilers.

The new standard libraries of the C++ programming language already provide an effective interconnection of strong object-oriented programming language and multithread running, which until recently, had been largely limited. Computational software tools that are already written in C or C++ programming languages are now able to enrich the possibility to use parallelism while maintaining the portability of code. In a field of an explicit form of the Finite Element Method ([6], [7]) or explicit meshless methods [16] the possibility of usage of parallelism is more than desirable, mainly because of time-consuming calculations, which is caused by the conditional stability of explicit methods used for the direct integration of equations of motion.

2 Explicit form of the Finite Element Method

The Explicit form of the Finite Element Method is based on a reformulation of the problem of static equilibrium to an artificial problem of dynamics with damping. Damping can be considered both explicitly with damping coefficients, and implicitly [13], [14].

There are generally more ways to derive the governing equations of motion. Namely, it is the Principle of Virtual Work, Hamilton's Principle and Hamilton's Law of Varying Action and Principle of Balance of Mechanical Energy respectively. These principles are well described in [9], both continuum and the dynamics of the particles.

Hamilton's Principle and Hamilton's Law of Varying Action, which are based on the formulation of scalar functions of potential and kinetic energy respectively, are probably the most well-known principles in theoretical physics.

The result is a well-known semi-discrete second-order differential equations of motion, which are then solved by direct integration using the method of central differences [9], [15].

3 Governing equations

Governing equations are based on The Principle of Conservation of Linear Momentum [9], [10]. Let us consider the body Ω with mass density $\varrho = \varrho(\mathbf{x}, \mathrm{t})$ in motion, which is subjected to body forces $\mathbf{b}(\mathbf{x}, \mathrm{t})$ and the traction force $\mathbf{t}(\mathbf{x}, \mathrm{t})$ acting on a surface $\partial\Omega$. Let $\dot{\mathbf{u}}(\mathbf{x}, \mathrm{t})$ be the Eulerian velocity field, then linear momentum $\mathbf{I}$ of the body Ω is defined as

$$\mathbf{I} = \int_{\Omega} \varrho \dot{\mathbf{u}}(\mathbf{x}, \mathrm{t}) \; dV$$

The principle of conservation of linear momentum as a counterpart to Newton's second law states that the rate of change of the linear momentum of an arbitrary part of a continuous medium is equal to the resultant force acting on the part under consideration, then:

$$\int_{\partial\Omega} \mathbf{t}(\mathbf{x}, \mathrm{t}) \; dA + \int_{\Omega} \mathbf{b}(\mathbf{x}, \mathrm{t}) \; dV = \frac{D}{Dt} \int_{\Omega} \varrho \dot{\mathbf{u}}(\mathbf{x}, \mathrm{t}) \; dV.$$

Using the $\mathbf{t} = \sigma \,.\, \mathbf{n}$, Gauss's divergence theorem, rearranging the resulting equation and considering the fact that the equation is also valid locally, so we get Cauchy's first law of motion (Eulerian description). For completeness it is necessary to consider prescribed Neumann's, Dirichlet boundary condition and initial state of body in time t_0

$$\nabla \,.\, \sigma + \varrho \mathbf{b} - \rho \ddot{\mathbf{u}} = 0 \text{ in } \Omega \tag{1}$$

$$\mathbf{u}(t_0) = \bar{\mathbf{u}}, \; \dot{\mathbf{u}}(t_0) = \dot{\bar{\mathbf{u}}_{\mathbf{0}}}, \; \mathbf{t} = \boldsymbol{\sigma} \,.\, \mathbf{n} \text{ in } \partial\Omega_N, \; \mathbf{u} = \mathbf{u_0} \text{ in } \partial\Omega_D \,,$$

Where $\varrho\mathbf{b}$ and $\varrho\ddot{\mathbf{u}}$ represent the density of body forces and material time derivative of Eulerian velocity field respectively. With respective initial conditions at time t_0 for displacement $\mathbf{u}$ and velocity field $\dot{\mathbf{u}}$ and natural boundary condition for traction forces $\mathbf{t}$ acting on boundary $\partial\Omega_N$ and prescribed displacement field $\mathbf{u_0}$ on boundary $\partial\Omega_D$ it leads to a complete formulation of the initial boundary value problem.

Cauchy's equation of motion is a formal vector representation of the governing equations of motion. It is a principle in continuum mechanics, which is analogous to D'Alembert's principle in the dynamics of particles.

4 Discretization of governing equations by the Finite Element Method

To get a suitable form of Cauchy's equations of motion for the finite element analysis let us use Hamilton's law of varying action which is deduced from an analogy to the Lagrangian form of D'Alembert's principle [9].

Discretization in space is carried out in the sense of Galerkin projection. Hamilton's principle or Hamilton's law of varying action naturally results in a weighted residual form in time, and the governing equation in time equals the finite element equation of motion.

Let us consider Cauchy's equilibrium equations under the Lagrangian formulation, particularly due to suitability for a solution of the generally nonlinear problems occuring in mechanics of deformable bodies.

The spatial configuration of the body Ω is defined as invertible mapping $\boldsymbol{\chi}_t\colon \Omega \to \mathbb{R}^3$ for fixed time t. The position of a continuum particle (material point) constituting the undeformed body is denoted by Cartesian coordinates $\boldsymbol{X} = (X^1, X^2, X^3) \in \Omega \subset \mathbb{R}^3$. Within a close interval of time $\mathrm{T} = \langle 0, T\rangle$, a smooth motion of a body Ω is described as mapping $\boldsymbol{\chi}(\boldsymbol{X}, t)\colon \Omega \to \mathbb{R}^3 \times \mathrm{T} \to \bar{\Omega} \subset \mathbb{R}^3, \forall t \in \mathrm{T}$, where $\bar{\Omega}$ denotes a set of infinite particles of the body in the spatial or current configuration. The space occupied by a set of all configurations in the dynamical motion is called the configuration space

$$C = \begin{cases} \boldsymbol{\chi}(\boldsymbol{X}, t)\colon \Omega \to \mathbb{R}^3 \times \mathrm{T} \to \bar{\Omega} \subset \mathbb{R}^3 | \boldsymbol{\chi} \in W_2^m(\Omega) = H^{\mathrm{m}}(\Omega) \\ \quad \boldsymbol{\chi}_X(t)\colon C^2(\mathrm{T}), m > 2, |\nabla\boldsymbol{\chi}| > 0, \text{ and } \boldsymbol{\chi}|_{\partial\Omega_{\boldsymbol{\chi}}} = \bar{\boldsymbol{\chi}} \end{cases},$$

where $\nabla\boldsymbol{\chi} = \frac{\partial\boldsymbol{\chi}}{\partial\boldsymbol{X}}$ and $\bar{\boldsymbol{\chi}}$ denotes prescribed displacement on boundary $\partial\Omega_{\boldsymbol{\chi}}$. Every configuration is infinite dimensional manifold, the second derivative of the motion must be square integrable in the Lebesgue sense up to the m-th order derivative with respect to the space independent variables $\left(\boldsymbol{\chi} \in H^{\mathrm{m}}(\Omega)\right)$.

Cauchy's equations of motion can then be reformulated to the Lagrangian form as follows

$$\begin{gathered} \nabla . \mathbf{P}(\boldsymbol{X}, t) + \varrho\mathbf{B}(\boldsymbol{X}, t) - \rho\boldsymbol{\chi}(\ddot{\boldsymbol{X}}, t) = 0 \text{ in } \Omega \\ \mathbf{T}(\boldsymbol{X}, \boldsymbol{N}, t) = \mathbf{P}(\boldsymbol{X}, t) . \boldsymbol{N}(\boldsymbol{X}, t) \text{ on } \partial\Omega_N, \ \boldsymbol{\chi} = \boldsymbol{\chi_0} \text{ on } \partial\Omega_D , \\ \boldsymbol{\chi}(t_0) = \bar{\boldsymbol{\chi}}, \qquad \dot{\boldsymbol{\chi}}(t_0) = \overline{\dot{\boldsymbol{\chi}}_{\mathbf{0}}}, \end{gathered}$$

where $\mathbf{P}(\boldsymbol{X}, t)$ is the first Piola-Kirchhoff stress tenzor, $\mathbf{B}(\boldsymbol{X}, t)$ denotes body forces, $\mathbf{T}(\boldsymbol{X}, \boldsymbol{N}, t)$ is prescribed traction on boundary $\partial\Omega_N$ in direction of outward normal $\boldsymbol{N}$ to the surface in current configuration, $\boldsymbol{\chi_0}$ and $\overline{\dot{\boldsymbol{\chi}}_{\mathbf{0}}}$ denote initial state and velocity of the body in time t_0. For the purpose of application of Hamilton's law of varying action it is needed to compose kinetic $\mathcal{K}$ and potential $\mathcal{U}$ energy of the respective continuum body.

$$\mathcal{U}(\boldsymbol{\chi}) = \int_\Omega \varrho_0 \boldsymbol{\Psi}(\mathbf{F})\, dV - \int_{\partial\Omega} \mathbf{T}(\boldsymbol{X}, \boldsymbol{N}) . \boldsymbol{\chi}(\boldsymbol{X}, t)\, dA - \int_\Omega \mathbf{B}(\boldsymbol{X}) . \boldsymbol{\chi}(\boldsymbol{X}, t)\, dV$$

$$\mathcal{K}(\dot{\boldsymbol{\chi}}) = \frac{1}{2}\int_{\Omega} \varrho_0 \dot{\boldsymbol{\chi}} \cdot \dot{\boldsymbol{\chi}}\, dV \ ,$$

where $\varrho_0 \boldsymbol{\Psi}(\mathbf{F})$ is density of the Helmholtz free energy and $\mathbf{F}$ is deformation gradient respectively. Then Hamilton's law of varying action has a form

$$\delta\mathcal{K}(\dot{\boldsymbol{\chi}}) - \delta\mathcal{U}(\boldsymbol{\chi}) - \frac{d}{dt}\int_{\Omega} \varrho_0 \dot{\boldsymbol{\chi}} \cdot \delta\boldsymbol{\chi}\, dV = 0$$

$$\delta\mathcal{L}(\boldsymbol{\chi}, \dot{\boldsymbol{\chi}}) - \frac{d}{dt}\int_{\Omega} \varrho_0 \dot{\boldsymbol{\chi}} \cdot \delta\boldsymbol{\chi}\, dV = 0\,,$$

where $\mathcal{L}(\boldsymbol{\chi}, \dot{\boldsymbol{\chi}})$ is the autonomous Lagrangian $\mathcal{L}(\boldsymbol{\chi}, \dot{\boldsymbol{\chi}}): TC \to \mathbb{R}^3$. Now let us consider the approximation of displacement field

$$\tilde{\boldsymbol{\chi}}(\boldsymbol{X}, t\,) \approx \tilde{\boldsymbol{\chi}}^h = \boldsymbol{X} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \mathbf{N}_i(\boldsymbol{X}\,)\mathbf{U}_i^e\,(t) \ = \boldsymbol{X} + \mathbf{N}_e(\boldsymbol{X})\boldsymbol{q}_e \qquad (2)$$

$$\boldsymbol{q}_e(t) = \{\mathbf{U}_1^e(t), \dots, \mathbf{U}_n^e(t)\}.$$

where $\mathbf{U}_i^e(t)$, $\boldsymbol{q}_e(t)$ denote nodal displacement vector and a set of generalized coordinates and $\mathbf{N}_i(\boldsymbol{X})$ denotes linear combination of coordinate polynomial shape functions [11], [12], [15]. Interpolating shape functions are defined over the domain of each finite element which interpolate within each element e the displacement of element node i (trial functions of class $C^{m-1}(\Omega)$). For the entire domain of interest, we introduce a configuration space with generalized coordinates

$$Q = \left\{\boldsymbol{q} | \boldsymbol{q}(t) = \left(U_1(t), \dots, U_{nDof}(t)\right) \in \mathbb{R}^{3\mathrm{n}}\right\}.$$

and the generalized velocities which belong to a tangent space

$$T_pQ = \left\{\dot{\boldsymbol{q}} | \dot{\boldsymbol{q}}(t) = \left(\dot{U}_1(t), \dots, \dot{U}_{nDof}(t)\right) \in \mathbb{R}^{3\mathrm{n}}\right\}.$$

Then the velocity phase space (tangent bundle) is defined as follows

$$TQ = \left\{(\boldsymbol{q}, \dot{\boldsymbol{q}}) | \boldsymbol{q} \in Q \ \text{ and } \dot{\boldsymbol{q}} \in T_pQ\right\} = \bigcup_{\boldsymbol{q}\in Q} T_pQ\,.$$

By means of equation (2), Hamilton's law of varying action within the e-th element domain Ω_e can be discretized in space as follows:

$$\int_{t_1}^{t_2} \left(\left(\frac{\partial \mathcal{L}_{\Omega_e}^h}{\partial \boldsymbol{q}_e}\right) - \frac{d}{dt}\left(\frac{\partial \mathcal{L}_{\Omega_e}^h}{\partial \dot{\boldsymbol{q}}_e}\right) \right) \cdot \delta\boldsymbol{q}_e(t)\, dt$$

Hence, in terms of generalized coordinates equation (15), leads to form

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}_{\Omega_e}^h}{\partial \dot{\boldsymbol{q}}_e}\right) - \left(\frac{\partial \mathcal{L}_{\Omega_e}^h}{\partial \boldsymbol{q}_e}\right) + \left(\frac{\partial \mathcal{D}_{\Omega_e}^h}{\partial \dot{\boldsymbol{q}}_e}\right) = 0.$$

$\mathcal{D}_{\Omega_e}^h$ is the Rayleigh dissipation function, which denotes damping forces, which consider the friction effect of the surroundings and the internal friction of viscose material, where it denotes the dissipative rate naturally in a more general form. For the problem of the damping effect in terms of material friction it would get the form of material described by Kelvin's rheological model. Here we consider the simple form of Rayleigh's dissipative potential in the form $\mathcal{D}_{\Omega_e}^h = \frac{1}{2}\dot{\boldsymbol{q}}_e^T C \dot{\boldsymbol{q}}_e$. Damping is artificially added to the equations of motion due to the

convergence rate control of calculation. For simplicity we consider simple linear elastic material with infinitesimal strains and displacements. At the end we get semi-discrete second-order differential equations of motion, which we use for direct integration by the finite difference method

$$M\ddot{\boldsymbol{q}} + C\dot{\boldsymbol{q}} + K\boldsymbol{q} = F_E. \tag{3}$$

Where $\ddot{\boldsymbol{q}}, \dot{\boldsymbol{q}}$ and $\boldsymbol{q}$ are vectors containing nodal accelerations, velocities and displacements respectively. $K\boldsymbol{q}$ represents internal forces (F_I) and (F_E) the external forces. M is the lumped mass matrix and C is the damping diagonal matrix.

5 Direct integration of equations of motion by Central Difference Method

Time derivatives of the equations of motions (3) are approximated by the Central Difference Method to get explicit formulas for the estimation of acceleration, velocity and displacement field respectively for time step. It is also possible to use the already existing integrator based on Newmark- β method $(\beta = 0, \gamma = 0.5)$ or Hilber-Hughes-Taylor- α method ($\alpha = 0 \rightarrow$ Newmark- β method).

$$\ddot{\boldsymbol{q}} \approx \ddot{\boldsymbol{q}}_h^n = \frac{1}{\Delta t^2}(\boldsymbol{q}_h^{n+1} - \boldsymbol{q}_h^n + \boldsymbol{q}_h^{n-1}),\ \dot{\boldsymbol{q}} \approx \dot{\boldsymbol{q}}_h^n = \frac{1}{2\Delta \mathrm{t}}(\boldsymbol{q}_h^{n+1} - \boldsymbol{q}_h^{n-1}) \tag{4}$$

Substitution $\ddot{\mathbf{q}}$ and $\dot{\mathbf{q}}$ from Equations (4) into Equation (3) gives

$$\left(\frac{1}{\Delta t^2}M + \frac{1}{2\Delta \mathrm{t}}C\right)\boldsymbol{q}_h^{n+1} = (F_E^n - K^n\boldsymbol{q}_h^n) + \frac{2}{\Delta t^2}M\boldsymbol{q}_h^n + \left(\frac{1}{2\Delta \mathrm{t}}C - \frac{1}{\Delta t^2}M\right)\boldsymbol{q}_h^{n-1} \tag{5}$$

For the ith degree of freedom, Equation (5) leads to the explicit formula for new displacement

$$\boldsymbol{q}_h^{n+1} = \alpha^{df}\left(F_{E,i}^{n,df} - F_{I,i}^{n,df}\right) + \beta^{df}\boldsymbol{q}_{h,i}^{n,df} - \gamma^{df}\boldsymbol{q}_{h,i}^{n-1,df} \tag{6}$$

where

$$\alpha^{df} = \frac{2\Delta t^2}{2m_i^{df} + c_i^{df}\Delta t}, \beta^{df} = \frac{4m_i^{df}}{2m_i^{df} + c_i^{df}\Delta t}, \gamma^{df} = \frac{2m_i^{df} + c_i^{df}\Delta t}{m_i^{df} + c_i^{df}\Delta t}.$$

Due to the conditional character of the resulting explicit time approximation of respective differential equations of motion, it is necessary to fulfill the Courant-Friedrichs-Levy conditions of stability for time increments Δt [9], [15]. Damping coefficients C and mass coefficients M respectively may be selected arbitrarily in the sense of the numerical dynamic relaxation method. Due to material and geometrical nonlinearities, coefficients of viscose damping must be chosen under consideration of a critically damped dynamic system to get a non-oscillating response. On the other hand, a dynamic system would experience non-physical states during solution.

6 Parallel Realization of Explicit Algorithm

The basic form of the explicit algorithm is divided into 2 phases as follows (it is shown schematically in Fig. 1, 2).

The displacement vector for each finite element is simply composed by code numbers of finite element nodes. Each thread handles the respective range of finite elements based on the prepared schedule from the input data.

Due to the explicit character of the respective algorithm, it is necessary to compose stiffness matrices only without necessity of the explicit assembly of the global stiffness matrix. The same possibility is also valid for the external force vector which comes from the

continuous character of forces on respective finite elements. This step is also required for the calculation of stresses for each finite element and the calculation of the internal force vector which is used in the next phase for the calculation of residual forces in application of the explicit formula (6) and convergence check. Associated values of current displacements for each finite element are stored in a simple data array in successive order of their respective code numbers. The displacement vector for each finite elements simply composed by code numbers of finite element nodes. Each thread handles the respective range of finite elements based on the prepared schedule.

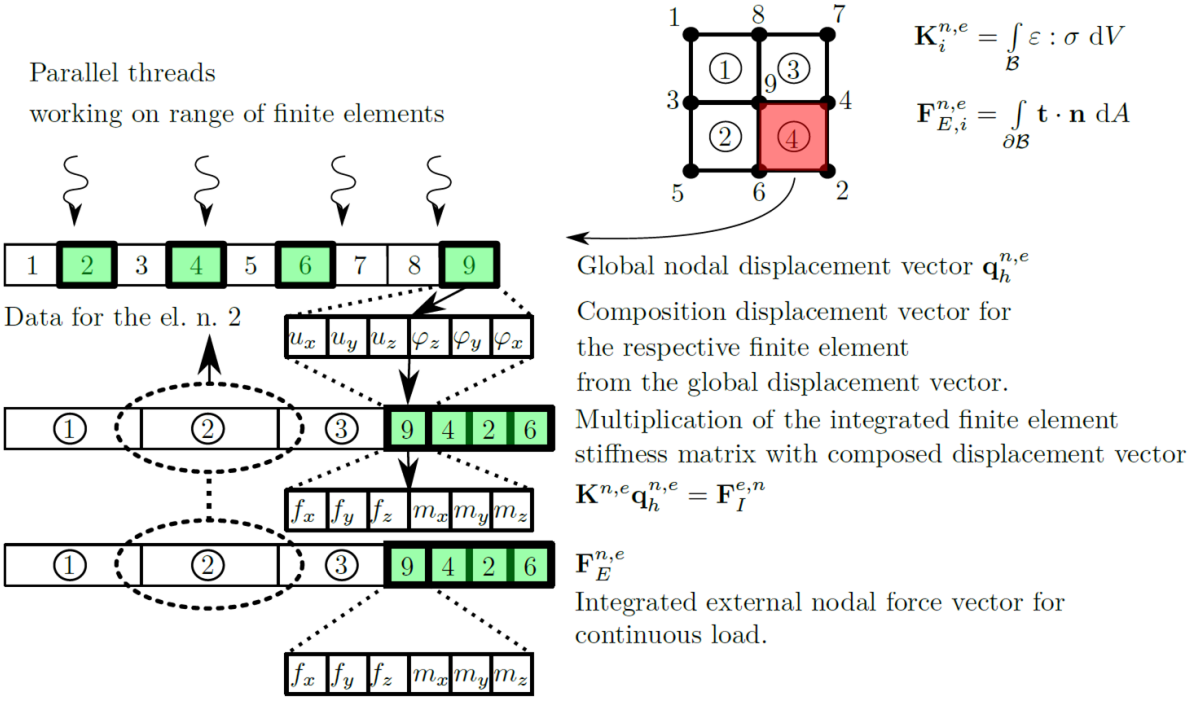

Fig. 1 Composition of external and internal force vectors

External nodal forces from the input data and integrated internal and external forces from the first phase are applied in the next phase to calculate current residual forces. Explicit integration of equations of motion, calculation of new displacement vector and consideration of boundary conditions respectively are performed in the second phase. Unlike the first phase, the scheduling of threads is taken as the range of the finite element nodes with their respective degrees of freedom. External nodal forces from the input data and integrated internal and external forces from the first phase are applied here to calculate current residual forces. For each finite element node, the sum of the contributions from the neighboring finite elements for the respective node is performed. This procedure is simply shown in Figure 2.

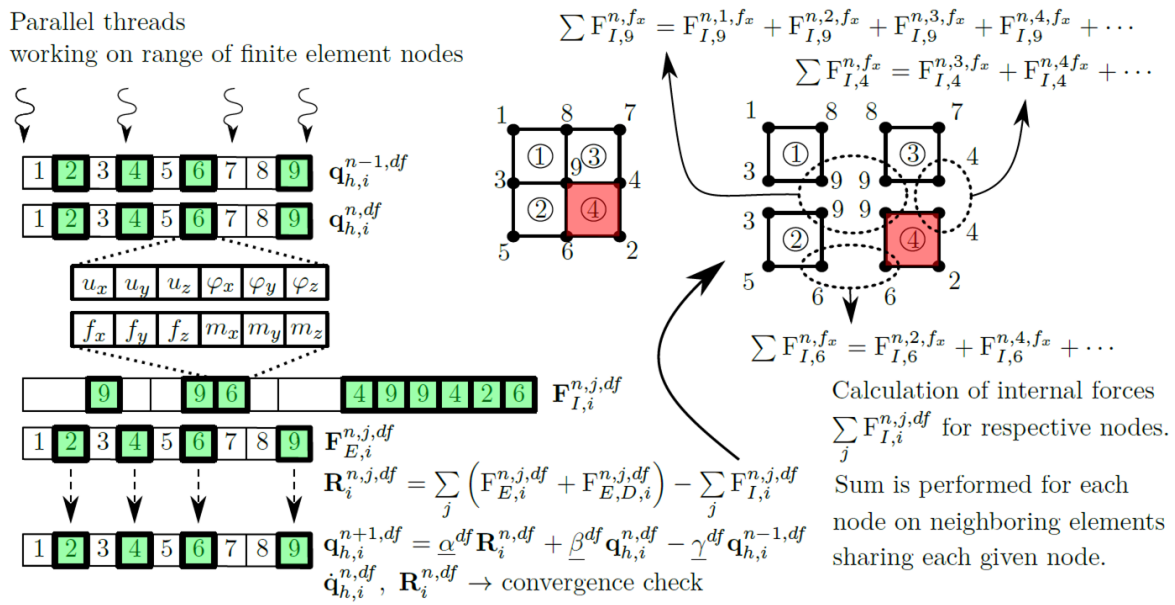

Fig. 2 Composition of new deflections of finite element nodes

7 Class model of parallel solver

The Sequential solver that was originally written in the C programming language has been modified and supplemented by subsequent classes. In Fig. 3 and 4 the class templates and their relationships using the simplified form of UML language are described. Class "FEVRThreadScheduler" contains all required data for computation. The class "FEVRThreadFactoryCpp" is responsible for C++ thread creation on CPU according to data which comes from the "FEVRThreadScheduler" object. Access to input data in each thread is done by using "FEVRSolverThreadInputData" class instance. The required class is "FEVRComputeRunner" which contains member function used as a procedure which is called by every running thread.

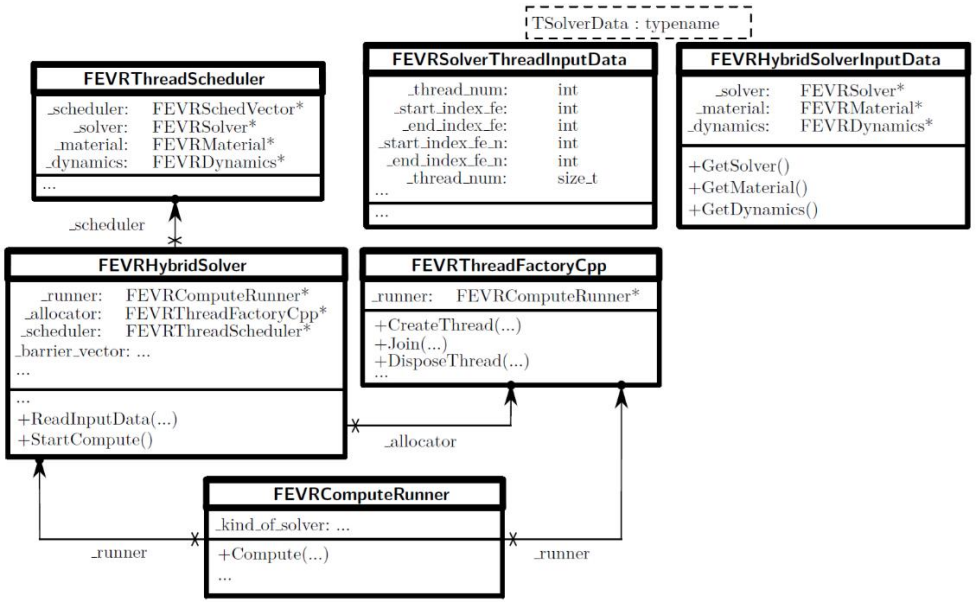

Fig. 3 Simplified UML class diagram of parallel solver input data

Software layer of the parallel solver, which is responsible for the appropriate use of the functionality which is shown in the previous figure is shown in Fig. 4.

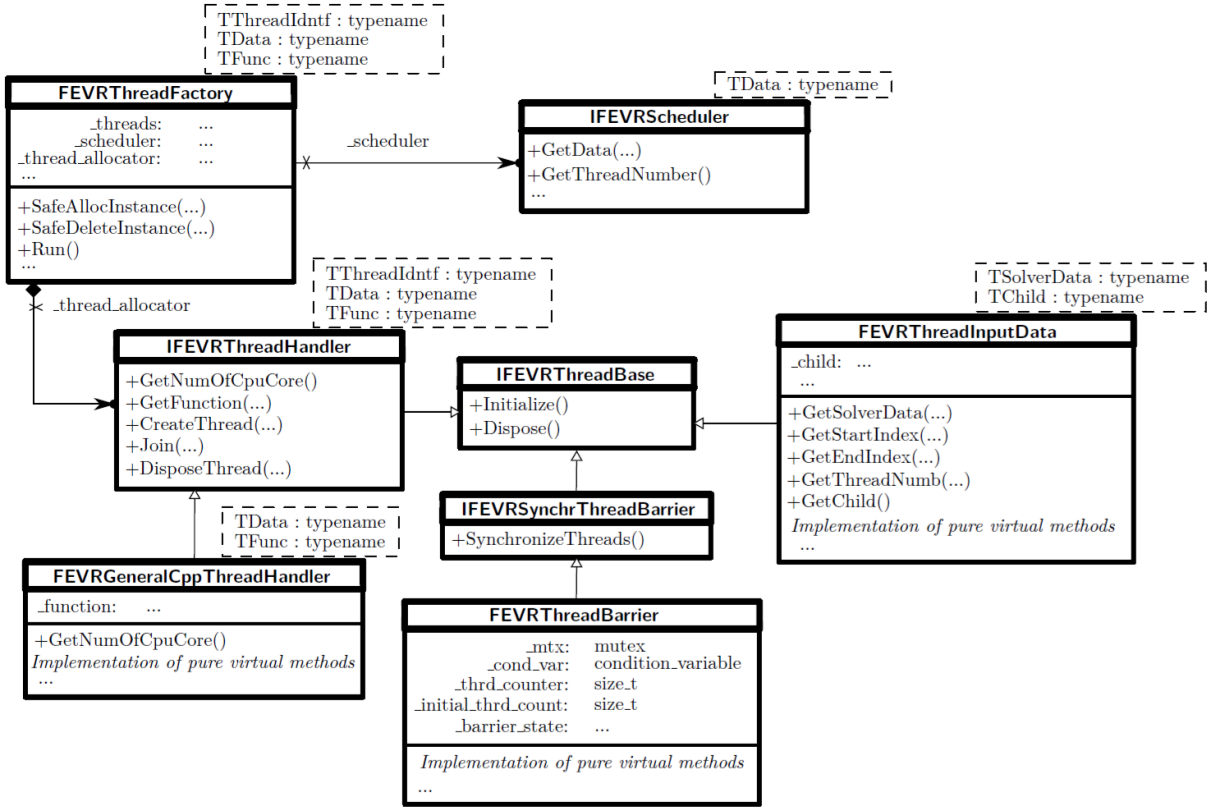

Fig. 4 Simplified UML class diagram of parallel solver input data

Class “FEVRThreadFactory” is responsible for the creation of threads and also their termination. Also described in the figure is class “FEVRThreadBarrier”, that is used by each thread to synchronize computing parts when it is necessary to ensure the synchronization of all threads on the one time level. This is mainly required for the phase between the integration of finite elements and the calculation of the new displacements.

8 C++ Standard Library

For the purpose of the explicit algorithm running on the CPU threads part of the functionality of the new C++ standard libraries is used. These standard libraries are part of the C++ programming language since version 11. New standard libraries support also asynchronous programming which is useful especially in the case of network communication and access to peripheral devices (=I/O). The respective class runs the function asynchronously (potentially in a separate thread which may be part of a thread pool) and returns “std::future” that will eventually hold the result of that function call. It is useful mainly for the purpose to hold UI (User Interface) in a responsive state [3].

An important class that is used to create CPU threads is in header file “thread.h”.

```
template<class _Fn, class... _Args>
explicit thread(_Fn&& _Fx, _Args&&... _Ax)
{
  // construct with _Fx(_Ax...)
  cay_copy(_STD forward<_Args>(_Ax))...));
}
```

Fig. 5 Constructor for creation of CPU thread

Creating a C++ thread which is implemented in presented solver is shown in Figure 6. For comparison the figure shows the creation of the CPU thread using native API of the Windows operating system. It is shown in the lower part of Fig. 6.

```
std::thread* FEVRThreadFactoryCpp
  ::CreateThread(std::function<void(IFEVRTID*)> fce, IFEVRTID* data)
{
  return new std::thread(fce, data);
}
HANDLE FEVRThreadFactoryCpp
  ::CreateThread(std::function<void(IFEVRTID*)> fce, IFEVRTID* data)
{
  unsigned threadID;
  typedef DWORD function_t(DT*);
  function_t** ptr_worker = fce.target<function_t*>();
  return (HANDLE)_beginthreadex(NULL, NULL,
  (_beginthreadex_proc_type)(*ptr_worker), data, NULL, &threadID);
}
```

Fig. 6 Creation of the CPU thread by using of C++ standard library and Win32 Api

The last important functionality that is needed in a multi-threaded program is synchronization of threads. The code used in solver is shown in Fig. 7.

```
void FEVRThreadBarrier::SynchronizeThreads()
{
#define WAIT(bar_state) ...
{ _cond_var.wait(lock, COMPARATOR((bar_state))); }
#define SET_AND_NOTIFY(bar_state) ...
{ _barrier_state = (bar_state); _cond_var.notify_all(); }
#define SEMAFOR(counter, value, bar_state) ...
{ if ((counter) == (value)) SET_AND_NOTIFY((bar_state)) else WAIT((bar_state)) }

  //unique_lock is released while calling its destructor et the end of function
  //for each thread
  std::unique_lock<std::mutex> lock(_mtx);
  if (COMPARATOR(BarrierState::e_DECR)())
    SEMAFOR(--_thrd_counter, 0                    , BarrierState::e_INCR)
  else
    SEMAFOR(++_thrd_counter, _initial_thrd_count, BarrierState::e_DECR)
}
```

Fig. 7 Synchronization barrier

For the presented functionality it is needed to include header files as follows:

#include <thread>, #include <vector>, #include <cstdlib>, #include <algorithm>, #include <memory>, #include <mutex> and #include <condition_variable>.

For a particular idea and purpose of interrelations among the different classes which compose a layer of parallel solver is used chart of Fig. 8. This shows the links that relate a direct reference to a particular object (ref.), the links created by the template parameters (templ.) and inheritance respectively (inh.).

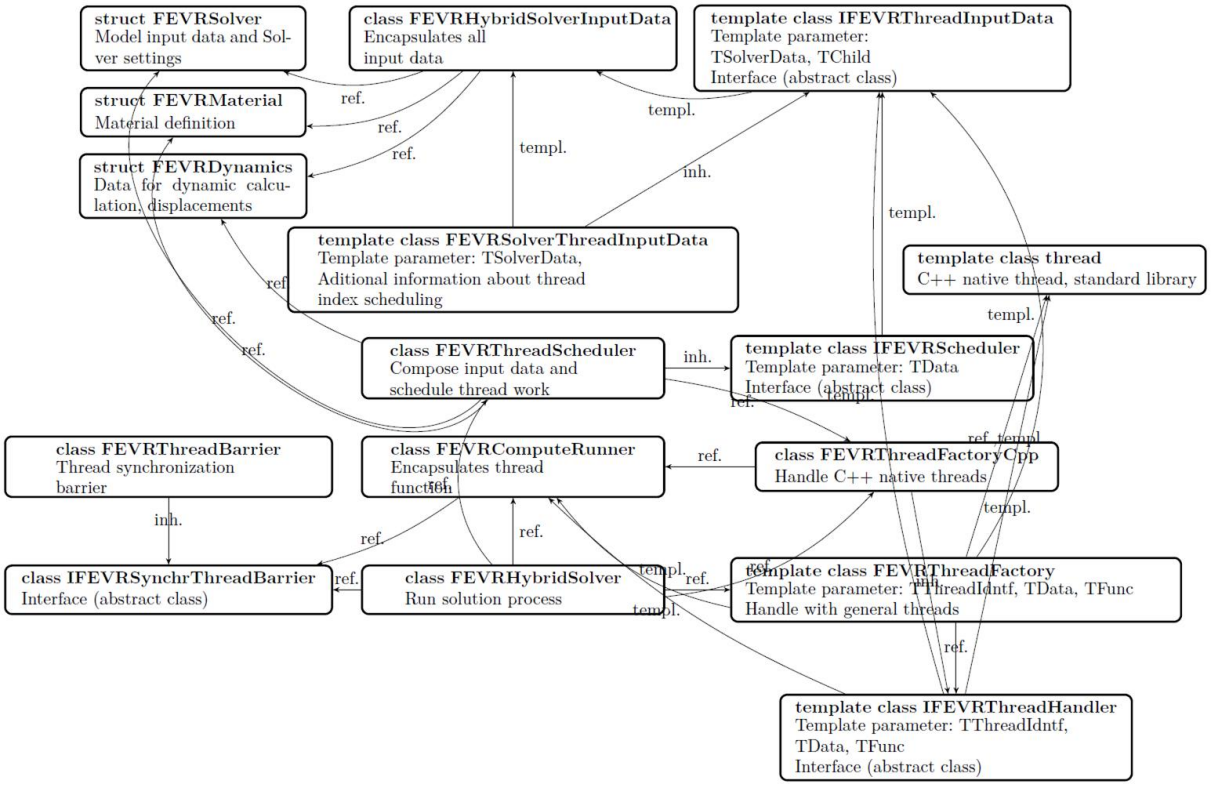

Fig.8 Class interconnections and dependencies

9 Class model of parallel solver

A test was performed on a model that consists of 5 000 finite elements (30 906 DOF) with 100 time steps. The effectiveness of the used algorithm was tested on the three different processors as follows:

- Intel Core2 Duo SU9400 - 1.40 GHz (2 Cores / 2 Threads)
- Intel Core i5-3320M Ivy Bridge - 2.6 GHz (2 Cores / 4 Threads)
- Intel Core i5-4690 3.5 GHz (4 Cores / 4 Threads)

Achieved performance (Sequential time [s] / Parallel time [s]):

- 1.986
- 1.972
- 3.826

The observed results from performance tests of the implemented algorithm has been found that only number of CPU cores affects the performance output. Hardware threads in this case are irrelevant.

CONCLUSION

The introduced approach to the potential usage of the modern form of the C++ programming language and its new standard libraries allows us to make better use of the support of parallel computations on the level of native programming language.

Functions written in C or C++ programming language designed for sequential run can be easily applied in a new C++ code, and thus, the existing computational tool could be adapted to exploit the opportunities of multicore processors.

Respective tests on different processors proved the effectiveness of the used approach.

REFERENCES

[1] A J. M. Hart: Windows System Programming (Addison-Wesley Microsoft Technology), Fourth Edition. Addison-Wesley Professional, 2015.

[2] A M. Kerrisk: The Linux Programming Interface: A Linux and UNIX System Programming Handbook. No Starch Press, 2010.

[3] A. Williams: C++ Concurrency in Action: Practical Multithreading. Manning Publications, 2010.

[4] T. Peierls, B. Goetz, J. Bloch, J. Bowbeer, D. Lea, D. Holmes: Java Concurrency in Practice. Addison-Wesley Professional, 2006.

[5] E. Agafonov. Multithreading in C# 5.0 Cookbook. Packt Publishing, 2013.

[6] S. R. Wu, L. Gu: Introduction to the Explicit Finite Element Method for Nonlinear Transient Dynamics. Wiley, 2012.

[7] V. Rek, I. Němec: Parallel Computing Procedure for Dynamic Relaxation Method on GPU Using NVIDIA's CUDA. Switzerland, Trans Tech Publications. Applied Mechanics and Materials, 2016, 821, 331-337.

[8] S. Prata. C Primer Plus, Fifth Edition. Sams Publishing, 2004.

[9] J. Har, K. K. Tamma. Advances in Computational Dynamics of Particles, Materials and Structures. Wiley. 2012.

[10] E. WV Chaves. Notes on Continuum Mechanics (Lecture Notes on Numerical Methods in Engineering and Sciences), Springer, 2013.

[11] E. O˜nate: Structural Analysis with the Finite Element Method. Linear Statics: Volume 2: Beams, Plates and Shells (Lecture Noteson Numerical Methods in Engineering and Sciences), Springer, 2009.

[12] E. O˜nate: Structural Analysis with the Finite Element Method. Linear Statics: Volume 1: Basis and Solids (Lecture Noteson Numerical Methods in Engineering and Sciences), Springer, 2013.

[13] J. Rodriguez, G. Rio, J.M. Cadou, J. Troufflard: Numerical study of dynamic relaxation with kinetic damping applied to inflatable fabric structures with extensions for 3D solid element and non-linear behavior. Elsevier, Thin-Walled Structures, 2011, 49, 1468-1474.

[14] J. Alamatian: A new formulation for fictitious mass of the Dynamic Relaxation method with kinetic damping. Elsevier, Computers & Structures, 2012, 91, 42-54.

[15] O. C. Zienkiewicz, R. L. Taylor: Finite Element Method: Volume 1, Fifth Edition, Butterworth-Heinemann, 2000.

[16] P. Staňák, J. Sládek, V. Sládek.: Analysis of Piezoelectric Semiconducting Solids by Meshless Method, In *Journal of Mechanical Engineering - Strojnícky časopis*, Vol. 65, No. 1, 2015, pp.77-92, ISSN 2450-5471