Použití konečně prvkových metod

Tomáš Pergler

Obsah prezentace

Možnosti řešení parciálních differenciálních rovnic s využitím volně dostupných C++ knihoven.

- LibMesh
- DealII
- PETSc
- Generování sítě
- Příklad - Poissonova rovnice
- Paralelní programování

LibMesh

- Využívá již hotových matematických knihoven
  - PETSc - seriové a paralelní řešení lineárních systémů
  - LASPACK - iterativní metody, multigrid, seriové řešení lineárních rovnic
  - METIS - rozdělení na nestrukturované sítě
  - SLEPc - analýza vlastních čísel
- Strukturované, nestrukturované a hybridní sítě
- Množství používaných formatů
  - libmesh format .xda,.xdr
  - Ideas Universal (.unv)
  - Sandia National Labs ExodusII (.exd)
  - AVS Unstructures UCD (.ucd)
- Rozdělení oblasti a paralelní řešení
- Lineární a nelineární systémy
- Postprocesingové nástroje - GMV, gnuplot
- Varianta nekonečných prvků

DealII

- Podobná knihovna jako LibMesh, inspirovala její tvorbu

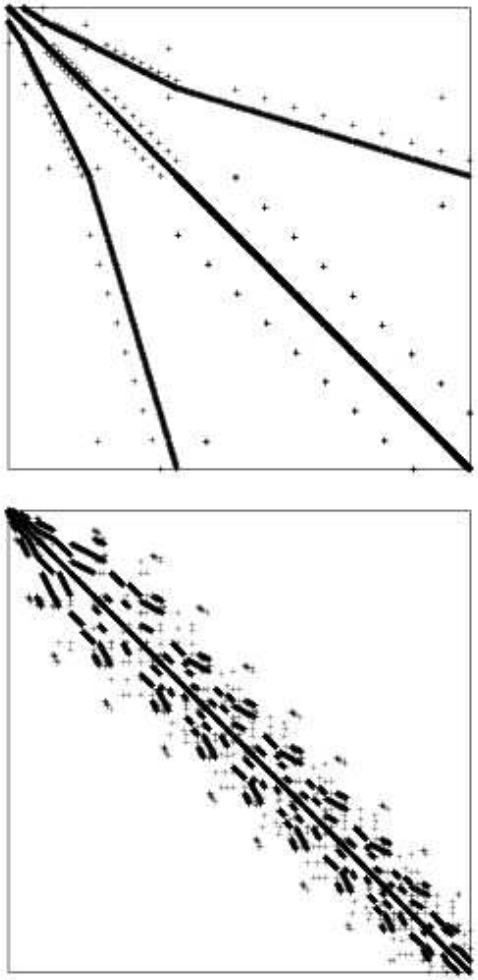
- Neobsahuje tolik typů elementů

- Neumožňuje paralelní výpočty na strojích s nesdílenou pamětí

+ Vyvíjen déle a proto obsahuje velkou škálu řešených příkladů

- Struktura obou knihoven je velice podobná a tudíž lze vytvořené kódy s jistými úpravami přenášet

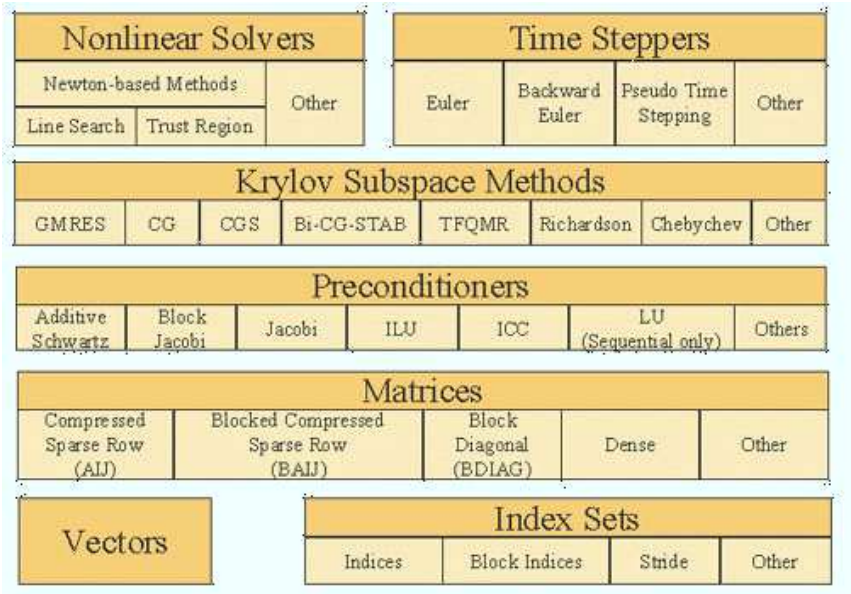
- Obrazky napravo ukazují strukturu matic obsahujících lineární soustavu vzniklou použitím FEM před a po přečíslování prvků



PETSc

- Portable, Extensible Toolkit for Scientific computation
- Poskytuje množství paralelních (i seriových) numerických řešení PDR
- Velké systémy, řídké a nelineární systemy rovnic
- Obsahuje nelineární a lineární řešič rovnic, využívá se různých Newtonových method

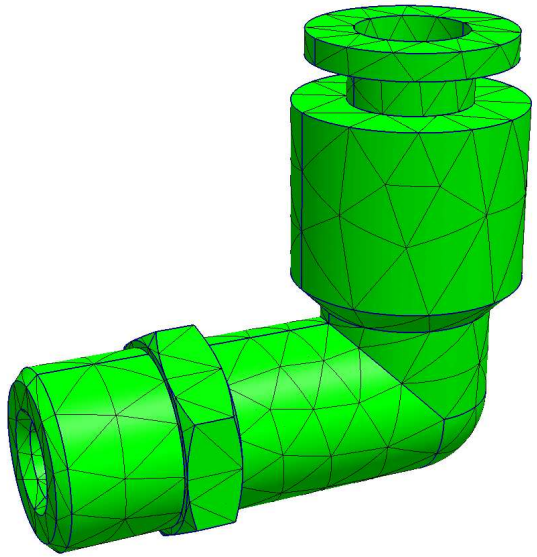
- PETSc je dostupný ve Fortranu a C/C++ a běží na většině UNIXových systémech



NETGEN

- Software na tvorbu sítí s exportem do běžně užívaných formátů [a]
- Trojúhelníky ve 2D a čtyřstěny ve 3D
- Kostka s vyříznutým válcem

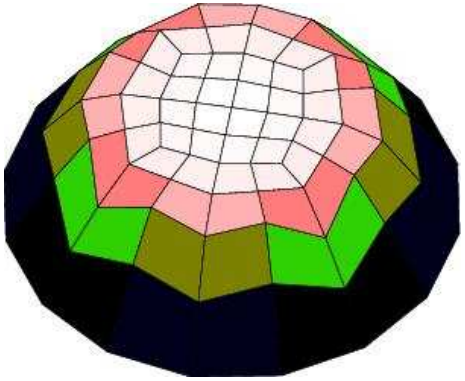
```
algebraic3d
solid cube = plane (0, 0, 0; 0, 0, -1)
   and plane (0, 0, 0; 0, -1, 0)
   and plane (0, 0, 0; -1, 0, 0)
   and plane (1, 1, 1; 0, 0, 1)
   and plane (1, 1, 1; 0, 1, 0)
   and plane (1, 1, 1; 1, 0, 0);
solid cyl = cylinder (-1, 0, 0; 0, 0, 0; 0.2);
# cut off cylinder:
solid main = cube and not cyl;

tlo main;
```

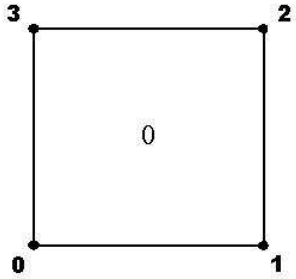


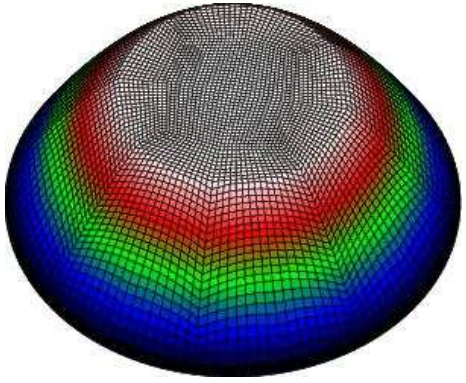
[a] http://www.andrew.cmu.edu/user/sowen/mesh.html

Formát souboru .xda

LIBM 0 # Standardní formát a počet zjemnění
1 # Počet prvků
4 # Počet uzlů
6 # Délka propojovacího vektoru
0 # Počet hraničních podmínek
65536 # Velikost řetězců (nepodstatné)
1 # Počet použitých typů prvků
1 # Typy prvků v každém bloku
1 # Počet prvků v bloku při různém zjemnění
Id String
Title String
0 1 2 3 0 -1
0. 0. 0.
1. 0. 0.
1. 1. 0.
0. 1. 0.

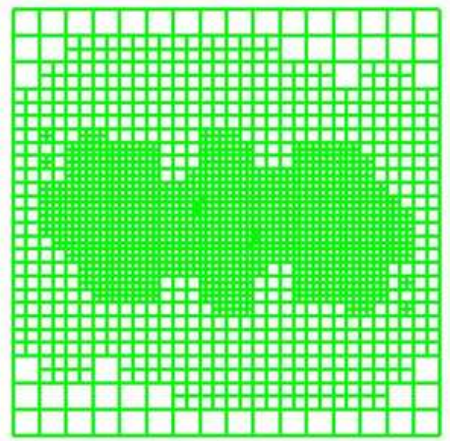


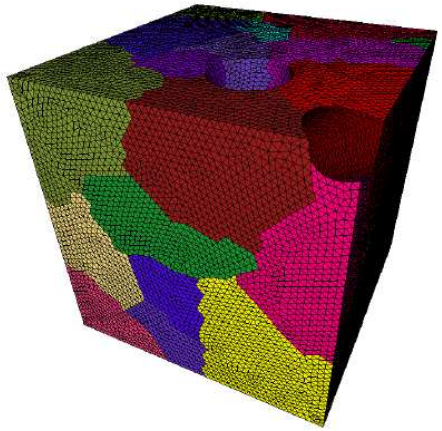


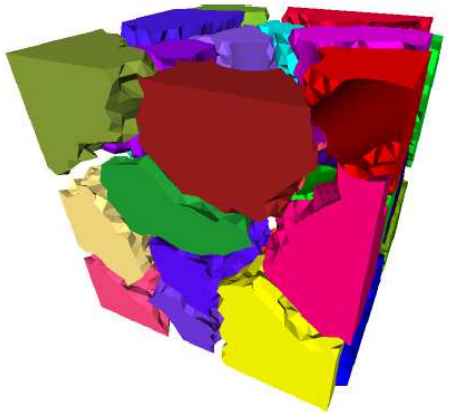


Prvky a uzly

- Trojuhelníky a čtyřúhelníky ve 2D
  - Tri3, Tri6, Quad4, ...
- Čtyřstěny, osmistěny, hranoly a jehlany ve 3D
  - Tet4, Tet10, Hex8, Hex27, ...
- Možnost použití tzv. volných uzlů
- Zjemňování elementů podle vypočtených hodnot





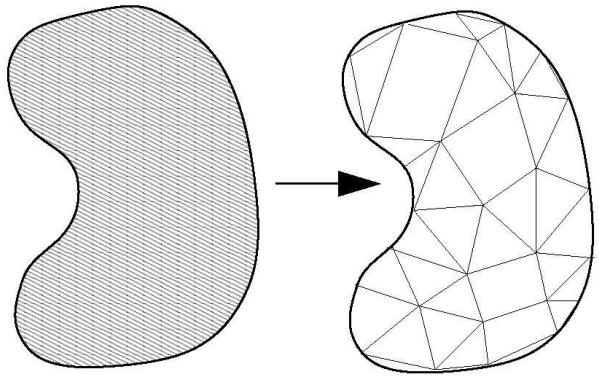


Příklad - Poissonova rovnice

- Najít u takové, že

  $-\triangle u = f = -\triangle g \qquad \text{na} \quad \Omega = [-1,1]^2$

  a $\quad u = g = \cos(\frac{\pi}{2}x)\sin(\frac{\pi}{2}y) \qquad \text{na} \quad \partial\Omega$



- Slabá formulace

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega = \int_\Omega f v \, \mathrm{d}\Omega \quad \forall v \in W^{1,2}(\Omega)$$

- Diskrétní slabá formulace

  Najít $u_h \in U_h$ takové, že

$$\sum_e \int_{\Omega_e} \nabla u_h \cdot \nabla v_h \, \mathrm{d}\Omega_e = \sum_e \int_{\Omega_e} f v_h \, \mathrm{d}\Omega_e \quad \forall v_h \in U_h$$

- Rozvineme u_h a v_h do vhodné báze $u_h = \sum_j U_j \phi_j$, $v_h = \sum_i V_i \phi_i$, kde neznámé hodnoty U_j jsou stupně volnosti

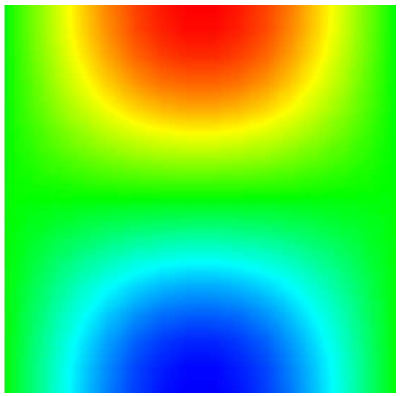
- Diskrétní problém

$$Ku = f$$

$$\text{kde} \quad K_{ij} = \int_{\Omega_e} \nabla \phi_i \cdot \nabla \phi_j \, \mathrm{d}\Omega_e, \quad f_i = \int_{\Omega_e} f \phi_i \, \mathrm{d}\Omega_e$$

Poissonova rovnice - kód

```
int main (int argc, char** argv){
    libMesh::init (argc, argv);
    {
        Mesh mesh (2);
        MeshTools::Generation::build_square (mesh, 15, 15, -1., 1., -1., 1., QUAD4);
        EquationSystems es(mesh);
        es.add_system<LinearImplicitSystem> ("Poisson");
        es.get_system("Poisson").add_variable("u", FIRST);
        es.get_system("Poisson").attach_assemble_function (assemble_poisson);
        es.init();
        es.get_system("Poisson").solve();
        GMVIO (mesh).write_equation_systems ("out.gmv", es);
    }
    return libMesh::close();

}
```



Poissonova rovnice - kompletování matice

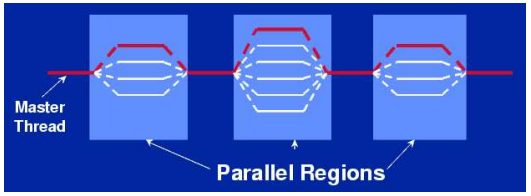
```
void assemble_poisson(EquationSystems& es, const std::string& system_name){


    const Mesh& mesh = es.get_mesh();
    const unsigned int dim = mesh.mesh_dimension();
    LinearImplicitSystem& system = es.get_system<LinearImplicitSystem> ("Poisson");
    const DofMap& dof_map = system.get_dof_map();

    FEType fe_type = dof_map.variable_type(0);
    AutoPtr<FEBase> fe (FEBase::build(dim, fe_type));
    QGauss qrule (dim, FIFTH);
    fe->attach_quadrature_rule (&qrule);

    const std::vector<Real>& JxW = fe->get_JxW();
    const std::vector<Point>& q_point = fe->get_xyz();
    const std::vector<std::vector<Real> >& phi = fe->get_phi();
    const std::vector<std::vector<RealGradient> >& dphi = fe->get_dphi();

    DenseMatrix<Number> Ke;
    DenseVector<Number> Fe;
    // Vektor obsahující globální index stupňů volnosti
    std::vector<unsigned int> dof_indices;
    MeshBase::const_element_iterator el = mesh.elements_begin();
    const MeshBase::const_element_iterator end_el = mesh.elements_end();
```

Poissonova rovnice - pokračování

```
    for ( ; el != end_el ; ++el) {
        const Elem* elem = *el;
        dof_map.dof_indices (elem, dof_indices);
        fe->reinit (elem);
        Ke.resize (dof_indices.size(), dof_indices.size());
        Fe.resize (dof_indices.size());
        for (unsigned int qp=0; qp<qrule.n_points(); qp++) {
            for (unsigned int i=0; i<phi.size(); i++)
                for (unsigned int j=0; j<phi.size(); j++)
                Ke(i,j) += JxW[qp]*(dphi[i][qp]*dphi[j][qp]);
            for (unsigned int i=0; i<phi.size(); i++)
                Fe(i) += JxW[qp]*fxy*phi[i][qp];
        }
        system.matrix->add_matrix (Ke, dof_indices);
        system.rhs->add_vector (Fe, dof_indices);


    }

}
```

Paralelní programovaní

- Rozdělení úlohy na dílčí problémy a dále pak na části, které mohou být zpracovávány současně
- Nejobecnější přístup
  Message Passing Interface - *MPI*
- Počítače se sdílenou pamětí
  OpenMP - zde není nutné obstarávat komunikaci mezi procesory, není potřeba znát technickou stránku věci
- OpenMP je poskytováno pro C/C++ a Fortran



Fork-Join Parallelism

```
#pragma omp parallel
#pragma omp for
        for (I=0;I<N;I++){
                NEAT_STUFF(I);
        }
```

Paralelní cyklus v C/C++

- Pokud neprogramujeme přímo paralelní kód, tak bysme měli alespoň dbát na používání knihoven, které jsou schopny řešit svoje úlohy paralelně. Starší knihovny nejsou ani tzv. thread-safe.