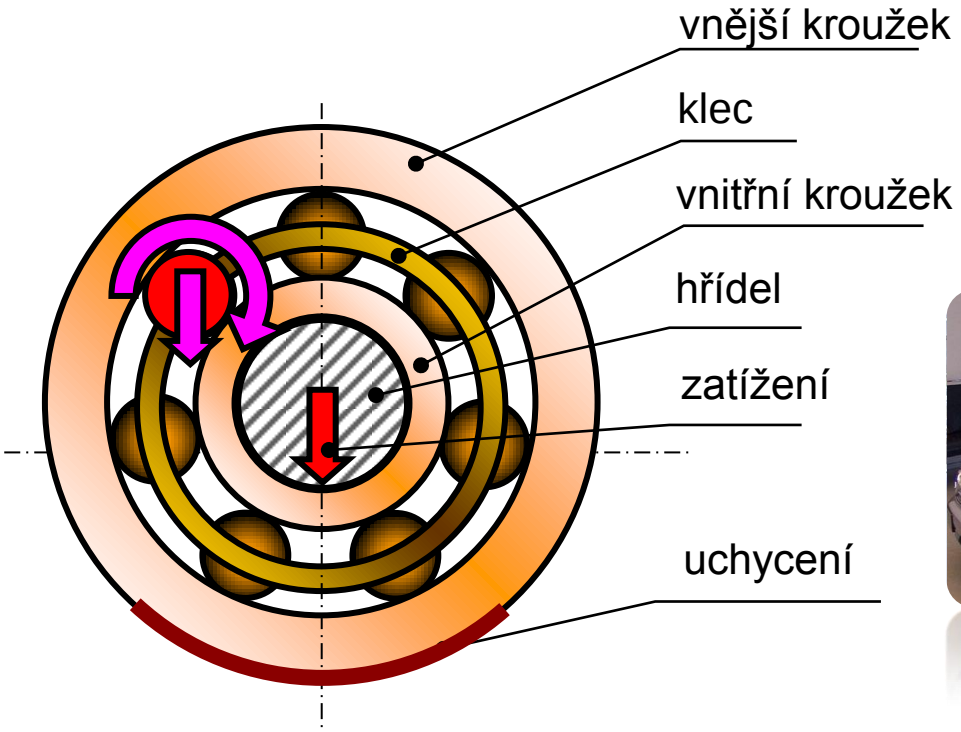
Objektově orientovaná implementace škálovatelných algoritmů pro řešení kontaktních úloh

Václav Hapla

Katedra aplikované matematiky
Fakulta elektrotechniky a informatiky
VŠB-Technická univerzita Ostrava

Na počátku je problém...



- Jaké je **rozložení mechanického napětí** v kuličkovém ložisku ?

Potřebujeme počítat?

- Máme tedy **fyzikální úlohu**.
- Rádi bychom „hezký výsledek“, třeba **3D obrázek** rozložení napětí v ložisku.
- Co takhle **změřit** mech. napětí ve skutečném namáhaném ložisku?
- Ale jak?
- Rozmístit **senzory**?
- Pohyblivé součásti!
- Zachování fyzikálních vlastností!
- Finanční a časová náročnost!

Fyzikální model

- Jedná se o **systém mnoha pružných těles**.
- Víme, že některá tělesa se vzájemně dotýkají, ale předem **nevíme, kde se dotýkají**.
- Některá tělesa jsou **pevně uchycena**, některá **nikoliv**.

Bez počítače to nepůjde…

- Úloha je dost složitá na to, abychom ji spočítali „na koleně“ jako jsme zvyklí u středoškolských úloh.
- Nezbývá nám, než si ložisko namodelovat v počítači a „nějak“ zjistit potřebné hodnoty mech. napětí.

… ani bez matematiky

- I když umíme programovat, pořád ještě **nevíme**, **co** vlastně potřebujeme **programovat**.
- Mezi fyzikální úlohou a výpočtem na počítači je ještě **několik vrstev matematiky**!

Matematické modelování

- Společně s fyziky vytvoříme **spojitý matematický model**.
- Obsahuje rovnice nebo nerovnice s integrály a derivacemi.
- Hledáme řešení, jež vyhovuje přirozeným fyzikálním podmínkám:
  - **Okrajové podmínky** (uchycení těles).
  - **Podmínky nepronikání**.

Náš spojitý model

- V našem případě **optimalizační úloha** – hledáme minimum $\mathbf{u}$ tzv. **energetického funkcionálu**

$$J(\mathbf{u}) = \min_{\mathbf{v} \in \mathcal{K}} J(\mathbf{v})$$

- $\mathbf{u}$ je **přípustné posunutí** (~ napětí) všech bodů systému, pro něž je **celková potenciální energie** (J) **minimální**.
- To odpovídá přírodě – systém v rovnováze má nejmenší možnou potenciální energii.

Diskretizace (1)

- **Počítač** ale **nerozumí spojitým** problémům a není schopen nalézt přesné řešení.
- Spojitou úlohu musíme převést na **diskrétní**.
- Představte si rozdělení na „buňky“, v celé **buňce** předpokládáme **všude stejné hodnoty napětí**.

Diskretizace (2)

- Výsledné řešení bude zatíženo jistou **chybou**.
- Tu ale lze **odhadnout** a také **ovlivňovat** podle potřeby (přesnost vs. rychlost)
- Nyní již **konečná množina hodnot** ve formě **matic** a **vektorů** (blíže počítačové realizaci)

$$\text{min.} \quad \frac{1}{2}\mathbf{u}^\top \mathbf{K}\mathbf{u} - \mathbf{f}^\top \mathbf{u}$$

$$\text{za podm.} \quad \mathbf{B}_\mathcal{I}\mathbf{u} \leq \mathbf{c}_\mathcal{I} \quad \text{a} \quad \mathbf{B}_\mathcal{E}\mathbf{u} = \mathbf{c}_\mathcal{E}$$

Dekompozice (1)

- Při hledání velmi hladkého řešení může počet „buněk“ narůst do miliónů a každá se může pohybovat mnoha směry!
- Časově náročný výpočet.
- Dobu výpočtu lze zkrátit šikovně provedenou **paralelizací**.
- K tomu je potřeba úlohu **dekomponovat**.

Dekompozice (2)

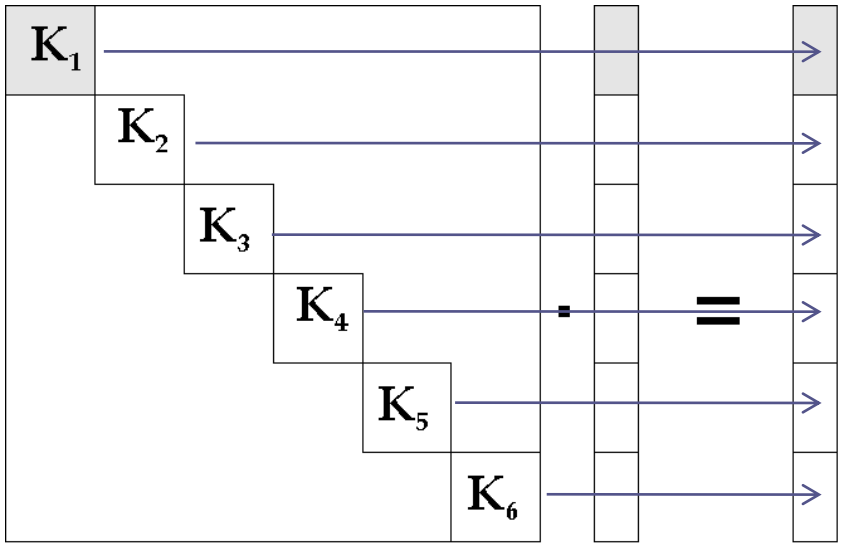
- **FETI metody** (finite element tearing and interconnecting)
- Systém těles se „roztrhá na kusy“ – **rozdělení na podoblasti**.
- Zavádějí se umělé **lepicí podmínky**, jejichž splnění zajistí návaznost řešení mezi podoblastmi.
- Každou podoblast lze pak počítat na jiném procesoru.

Dekompozice (3)

- Model auta rozdělený na podoblasti
- Zhruba stejně velké – **rovnoměrná zátěž CPU**

Dekompozice (4)

- Matice $\mathbf{K}$ se stává **blokově diagonální, bloky ~ oblasti**
- Lze paralelizovat např. její sestavení, násobení vektorem.



$$\mathbf{K} \cdot \mathbf{u} = \mathbf{K}\mathbf{u}$$

$$\text{min.} \quad \frac{1}{2}\mathbf{u}^\top \mathbf{K}\mathbf{u} - \mathbf{f}^\top \mathbf{u}$$

$$\text{za podm.} \quad \mathbf{B}_\mathcal{I}\mathbf{u} \leq \mathbf{c}_\mathcal{I} \quad \text{a} \quad \mathbf{B}_\mathcal{E}\mathbf{u} = \mathbf{c}_\mathcal{E}$$

Škálovatelnost

- Pracujeme na algoritmech, jež jsou **numericky a paralelně škálovatelné**.
- **paralelní škálovatelnost:** doba výpočtu ~ 1 / počet CPU
- **numerická škálovatelnost:** doba výpočtu ~ velikost úlohy

Příklad OO přístupu

- Na rozdíl od čísel, u matic a vektorů záleží na pořadí násobení.
- $(\mathbf{AB})\mathbf{v} = \mathbf{A}(\mathbf{Bv})$, ale z hlediska implementace velký rozdíl.
- Provést **násobení $\mathbf{AB}$** může být příliš **drahé** (čas, datové nároky).
- Potřebujeme-li $\mathbf{AB}$ jako argument algoritmu, můžeme jej implementovat jako **„virtuální matici“**

Příklad OO přístupu

```
classdef Mat_AB
    properties
        A
        B
    end

    methods
        function this = Mat_AB(A,B)
            this.A = A;
            this.B = B;
        end

        function p = mtimes(this, x)
            p = this.A * (this.B * x);
        end
    end
end
```

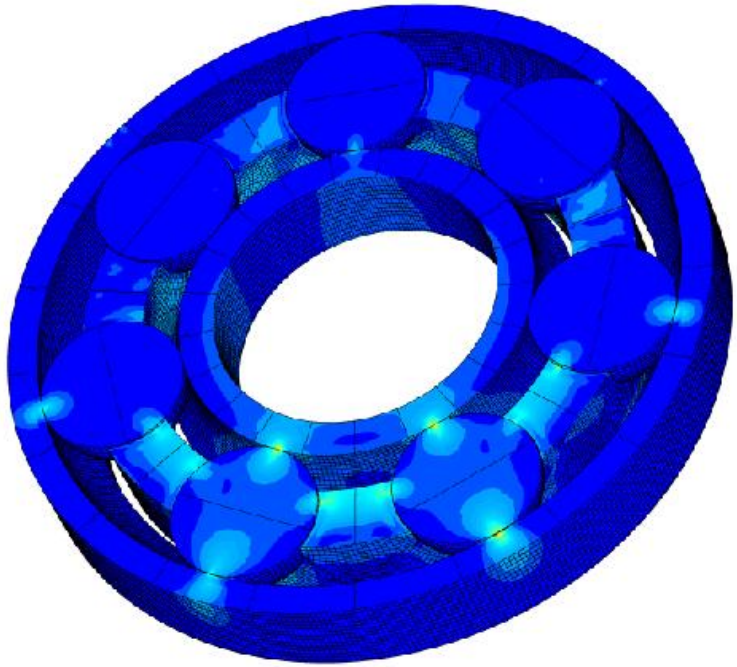
Mat_AB.m

```
>> AB = Mat_AB(A,B);
>> norm(AB*x – A*B*x) < 1e-12

ans =

     1
```

Virtuální matice **Mat_AB** umí pouze **násobit se s vektorem**.

Můžeme ale implementovat třeba i **paralelizaci** (např. naše blokově diagonální **K**).

Výsledek – rozložení napětí



Vypočteno knihovnou MatSol

Paralelní počítače

IT4Innovations
Centrum excelence

www.it4i.eu
www.prace-project.eu

Děkuji za pozornost.