Making a Common Lisp FE library high-performant

Marco Heisig*, Nicolas Neuss**

* Computer science 10 (System simulation)
Friedrich-Alexander-Universität Erlangen-Nürnberg

** Applied Mathematics 3 (Scientific computing)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Contents

- Partial Differential Equations
- The Finite Element Method
- What is Femlisp?
- Model problem
- Working towards high performance
- Outlook

Motivation of Partial Differential Equations (PDE)

- OBSERVATION: Phenomena which are characterized by **instantaneous** and **short-range** interactions in a **continuum** can be modelled by **partial differential equations (PDEs)**.
- EXAMPLES: continuum mechanics, fluid mechanics, reaction and transport, general relativity, quantum mechanics
- APPLICATIONS: Physics, chemistry, biology, economy, . . . , many engineering disciplines

Definition and Examples

Definition

A PDE is an equation for an unknown *function* $u : \Omega \to \mathbb{R}$ which has to be satisfied for all points $x \in \Omega \subset \mathbb{R}^d$ ($d \geq 2$) involving only the values of the function and its derivatives at each x.

Examples

- 2D-Diffusion: $\Delta u \equiv \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) u(x, y) = s(x, y)$
- Stokes ($d + 1$ equations): $$\begin{aligned} -\Delta u + \nabla p &= f \\ \operatorname{div} u &= 0 \end{aligned}$$
- Not a PDE: Search $u : \mathbb{R} \to \mathbb{R}^m$ with $\frac{du}{dt}(t) = f(t, u(t))$.

Difficulties when solving PDEs

- The solution of a PDE is a function defined on a continuum
  $\Rightarrow$ Often a large number of unknowns is necessary for approximating it well.
- Existence, uniqueness and regularity of solutions to PDEs is often a difficult (sometimes even an unsolved) problem.
  $\Rightarrow$ Also the discretized equations may be “ill-conditioned” and difficult to solve.
- Already the precise definition of the problem can be nontrivial (e.g. when the domain Ω is geometrically complex)

The Finite Element Method

- Mathematical theory starts from "**variational form**":
  Find $u \in V$ with $a(u, v) = f(v)$ for all $v \in V$.
- V is an (infinite-dimensional) function space adapted to the problem at hand.
- IDEA OF FEM: **Approximate** V with a space V_h made from **piecewise polynomial functions** defined on a **mesh**.
- Construct **discrete equations** by restricting the variational form to V_h.
- PROPERTIES: Flexibility, good theoretical foundation, somewhat more complex than e.g. Finite Difference Methods

Femlisp

Femlisp (FEMlisp?) is a Common Lisp FEM framework with:

- Arbitrary-dimensional meshes consisting of simplex and/or simplex-product cells (like cubes or prisms)
- Anisotropic and local mesh refinement
- Conforming FE of arbitrary order
- Geometric and algebraic multigrid
- Interactive graphics (interface to OpenDX and VTK)
- Can handle several types of PDEs:
  convection-diffusion, elasticity, Navier-Stokes, ...

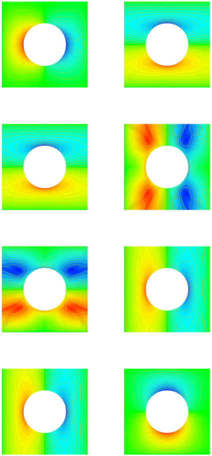
Model problem: 3D linear elasticity

- A *periodically* perforated elastic medium satisfies an effective elasticity law.
- The **effective elasticity tensor** is

$$\hat{A}_{iq}^{kr} = \int_Y A_{ij}^{kl}(y)\left(\delta_{jq}\delta_{lr} + \frac{\partial N_q^{lr}}{\partial y_j}(y)\right) dy$$

- where the **corrector** $N : Y \to \mathbb{R}^{d^3}$ satisfies

$$-\frac{\partial}{\partial y_i}\left(A_{ij}^{kl}(y)\frac{\partial N_q^{lr}}{\partial y_j}(y)\right) = \frac{\partial A_{iq}^{kr}}{\partial y_i}(y)\,.$$



Femlisp results (state from 2005-2015)

- **Test 1:** 2D, hole inlay, uniform refinement, Q^5-FE:

Cells	Unknowns	Matrix entries	Time (s)	$\hat{A}_{11}^{11}$
4	872	18.836K	3.2	1.7928477139
16	4.224K	78.780K	10.6	1.7925713781
64	17.336K	314.716K	34.3	1.7925694507
256	69.168K	1.257M	136.6	1.7925694414
1024	275.240K	5.022M	597.0	1.7925694414

- **Test 2:** 3D, hole inlay, uniform refinement, Q^5-FE:

Cells	Unknowns	Matrix entries	Time (s)	$\hat{A}_{11}^{11}$
6	22.167K	2.125M	76	2.6235177047
48	192.024K	18.571M	2373	2.6231458888

How to increase performance?

SYSTEMATIC APPROACH:

- Check if algorithm is good enough
- Check for possible use of high-performant libraries (BLAS, LAPACK, ...)
- Optimize single core performance (profiling!)
- Shared-memory parallelization (OS threads)
- Distributed-memory parallelization (MPI)

Using libraries and choice of algorithm

- The results above already used the BLAS/LAPACK libraries (before they were worse by a factor of about 2).

- They also used multigrid with a special *p*-robust smoother ("vertex-centered SSC") which is ok in 2D, but costly in 3D
$\rightsquigarrow$ Multigrid with simple block-Gauss-Seidel gives:

Cells	Unknowns	Matrix entries	Time (s)	$\hat{A}_{11}^{11}$
6	22.167K	2.12473M	19	$\underline{2.6235}177047$
48	192.024K	18.5712M	165	$\underline{2.62314}58888$

- Gauss-Seidel is not parallelizable $\rightsquigarrow$
BPX (CG, additive V-cycle, block-Jacobi smoother) gives:

Cells	Unknowns	Matrix entries	Time (s)	$\hat{A}_{11}^{11}$
6	22.167K	2.12473M	9	$\underline{2.6235}177047$
48	192.024K	18.5712M	114	$\underline{2.62314}58888$

Improvement of the serial code

- Profiling shows bottleneck in generic function `MREF`

  In Lisp, it is easy to write fast code.
  Unfortunately, it is very easy to write slow code.
  (Paul Graham, "On Lisp")

- REMEDY: Block-wise updates of the global matrix during discretization eliminates bottleneck and gives:

Cells	Unknowns	Matrix entries	Time (s)	$\hat{A}^{11}_{11}$
6	22.167K	2.125M	3.7	2.6235177047
48	192.024K	18.5712M	47	2.6231458888
384	1.520M	148.704M	378	2.6231424485

- $\rightsquigarrow$ Profiling does not show easily removable bottlenecks

Shared-memory parallelization – 1

- Defect calculation can be performed in parallel (HOWEVER: might not be completely unproblematic depending on matrix/vector data structure)
- Discretization can be performed in parallel (HOWEVER: update of the global stiffness matrix must be synchronized)
- We used a worker pool working on assembly pipelines containing assembly tasks and global matrix update tasks
- Results (on my laptop with two threads):

Cells	Unknowns	Matrix entries	Time (s)	$\hat{A}_{11}^{11}$
6	22.167K	2.125	2.7	2.6235177047
48	192.024K	18.571M	36	2.6231458888
384	1.520M	148.704M	290	2.6231424485

Shared-memory parallelization – 2

- Results on *sultana* (older workstation with large memory):

Cells	Unknowns	Matrix entries	Time (s)	$\hat{A}_{11}^{11}$
6	22.167K	2.125M	4	2.6235177047
48	192.024K	18.571M	40	2.6231458888
384	1.520M	148.704M	300	2.6231424485
3072	12.017M	1.188G	2400	2.6231424309

- Speedups on level 3 (384 cells):

Threads	1	2	3	4	5	6
Speedup	1	1.7	2.1	2.2	2.8	2.9

Distributed-memory parallelization in Common Lisp

- CL-MPI (Marco Heisig): CL interface to MPI
- LFARM (James M. Lawrence): Interactive control of the workers
- DDO -“Dynamic Distributed Objects”
  - Creation, removal and changes for distributed objects are communicated at synchronization points.
  - Distributed objects can be dropped and left to GC
  - Basic administrative data structure:
    Triple relation (*local-index*, *processor*, *distant-index*)

Distributed-memory parallelization for Femlisp

1. Starting from identical coarse meshes, parts belonging to other processors are dropped, and the interfaces are DDO-identified (distributed). Refinement of distributed interfaces remains distributed.

2. Discretization works without synchronization, because we work with inconsistent stiffness-matrix A_i and load-vector f_i.

3. BPX solving needs some synchronization ($S_{I \to C}$):
   - One-time calculation of consistent diagonal $D_c := S_{I \to C} D_i$.
   - Correction: $c_i := D_c^{-1} r_i$ followed by $c_c := S_{I \to C} c_i$ before correcting $u_c := u_c + c_c$. [1]
   - For monitoring: $r_c := S_{I \to C} r_i$ and $\|r\|_2^2 := \langle r_c, r_i \rangle$.

[1] $r_i = f_i - A_i u_c$ denotes the inconsistent residual.

Distributed-memory parallelization – Results

- Results on *sultana*

Cells	MPI workers 1	2	3
6	3.7	2.8	2.4
48	38	24	17
384	295	164	115
3072	2400	1250	840

- And using the *LiMa* cluster of the RRZE

Cells	MPI workers 2	4	6	8	12	16	24	48
48	20	13	11	9	8	8	9	10
384	100	58	45	36	27	24	23	20
3072	—	—	240	190	130	105	82	53

Current work on Femlisp

- Solving this model problem still faster and more accurate
- Thread-parallel DDO
- Load-balancing with DDO
- More functional approach to PDE solution (?)
- Applications
  - Benchmark flow problems (driven cavity, flow around a cylinder)
  - Interactive demo at “Long Night of Sciences”
  - Research on “Multiscale Finite Elements”
  - …

References

- `http://www.femlisp.org`
- M. Heisig, N. Neuss: “Distributed High Performance Computing in Common Lisp”, in Proc. 9th European Lisp Symposium (2016)
- M. Heisig, N. Neuss: “Making a Common Lisp Finite Element library high-performing — a case study” (2017, submitted)