High-performance numerical simulation on distributed heterogeneous hardware platforms

Matthias Möller
Delft University of Technology, Department of Applied Mathematics

8th July 2019, SURFsara, Amsterdam
Overview

- Design-Through-Analysis
- Overview of Spline technologies
- DTA for twin-screw compressor
- Implementation aspects
- Recent updates and collaboration plans
Motivation

DESIGN-THROUGH-ANALYSIS
Example: Airfoil design

- Early airfoil
- Later airfoil
- Clark 'Y' airfoil (Subsonic)
- Laminar flow airfoil (Subsonic)
- Circular arc airfoil (Supersonic)
- Double wedge airfoil (Supersonic)

Image gallery at nasa.gov

CFInotebook.net
Design-through-analysis cycle

1. Design

2. Simulation

3. Analysis

4. Redesign

Coefficient of Lift Distribution

Coefficient of Lift

Angle of Attack (deg)

Simulated
Experimental
DTA: 1. Design $D(\mathbf{p})$

- **Design parameters**
  
  $$\mathbf{p} = (p_1, \ldots, p_{12})$$

- **Admissible design space**
  
  $$S = [p_{11}^{\text{min}}, p_{11}^{\text{max}}] \times \ldots \times [p_{12}^{\text{min}}, p_{12}^{\text{max}}]$$
DTA: 2. Simulation

Mathematical model

Navier–Stokes Equations

$\begin{align*}
\text{Continuity:} & \quad \frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} + \frac{\partial (\rho v)}{\partial y} + \frac{\partial (\rho w)}{\partial z} = 0 \\
\text{X - Momentum:} & \quad \frac{\partial (\rho u)}{\partial t} + \frac{\partial (\rho u^2)}{\partial x} + \frac{\partial (\rho u v)}{\partial y} + \frac{\partial (\rho u w)}{\partial z} = -\frac{\partial P}{\partial x} + \frac{1}{Re} \left[ \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} \right] \\
\text{Y - Momentum:} & \quad \frac{\partial (\rho v)}{\partial t} + \frac{\partial (\rho u v)}{\partial x} + \frac{\partial (\rho v^2)}{\partial y} + \frac{\partial (\rho v w)}{\partial z} = -\frac{\partial P}{\partial y} + \frac{1}{Re} \left[ \frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} \right] \\
\text{Z - Momentum} & \quad \frac{\partial (\rho w)}{\partial t} + \frac{\partial (\rho u w)}{\partial x} + \frac{\partial (\rho v w)}{\partial y} + \frac{\partial (\rho w^2)}{\partial z} = -\frac{\partial P}{\partial z} + \frac{1}{Re} \left[ \frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} \right] \\
\text{Energy:} & \quad \frac{\partial (E_r)}{\partial t} + \frac{\partial (w E_r)}{\partial x} + \frac{\partial (w E_r)}{\partial y} + \frac{\partial (w E_r)}{\partial z} = \frac{\partial Q}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial Q}{\partial z} + \frac{1}{Re Pr} \left[ \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} \right]
\end{align*}$

Solution for one particular design and one particular angle of attach

$U = U(D(p); AoA)$

Linné FLOW Centre and SeRC, KTH, Sweden
DTA: 3. Analysis

- Cost functional

\[ C(U; D) \]
Example: Operation conditions
Design-through-analysis

1. Find a set of admissible design parameters $p$ and generate the design $D(p)$
2. Compute solutions $U(D(p); AoA)$ to the mathematical model $M(U, D(p))$
3. Evaluate the cost functional $C(U, D(p))$ for all solutions/operating conditions
4. Vary the design parameters $p$ to optimize the cost functional $C(U, D(p))$ for a wide range of operating conditions and repeat the DTA cycle at step 1
Design-through-analysis

1. Find a set of admissible design parameters $\mathbf{p}$ and generate the design $D(\mathbf{p})$
2. Compute solutions $\mathbf{U}(D(\mathbf{p}); A_0 A)$ to the mathematical model $\mathcal{M}(\mathbf{U}, D(\mathbf{p}))$
3. Evaluate the cost functional $\mathcal{C}(\mathbf{U}, D(\mathbf{p}))$ for all solutions/operating conditions
4. Vary the design parameters $\mathbf{p}$ to optimize the cost functional $\mathcal{C}(\mathbf{U}, D(\mathbf{p}))$ for a wide range of operating conditions and repeat the DTA cycle at step 1

4. \textbf{control point distribution:}

Figure 6: The B-spline curve is defined by 14 control points with different degrees of freedom. In order to increase the accuracy at the leading edge, two control points are used to define this region. These points are free to move in both x and y directions. The distribution is refined at the leading and trailing edges.

Number of design variables = 14

2.2.1. \textbf{Geometrical flexibility}

A simple test is performed with MATLAB to test the flexibility of the distributions described above. In principle, any physically realistic shape should be achievable to allow design from an initial arbitrary shape. Hence, the aim of the test is to try reproduce different existing airfoil profiles using a given distribution and evaluate the maximal geometrical difference between the two profiles.

The geometrical difference between the two curves is evaluated vertically at 100 different points not equally spaced along the chord length. Indeed the density of evaluation points is higher at the leading and trailing edges. The test is directed by the MATLAB function \texttt{lsqnonlin} developed to solve nonlinear least-squares (nonlinear data-fitting) problems. The input parameters are the coordinates of the control points (according to their degrees of freedom) and the output is the vector of geometrical differences computed at each evaluation point. The function minimizes the difference between the two curves by gradually moving the control points (see figure below).

Figure 7: Test of the validation of the control point distribution
Introduction to

SPLINE TECHNOLOGIES
Basis functions: $\hat{B}_i(\xi) \quad i = 1, \ldots, N$

Linear 'tent' basis functions

Quadratic B-spline basis functions
Curves: \( C(\xi) = \sum_{i=1}^{N} c_i \hat{B}_i(\xi) : [0,1] \rightarrow \mathbb{R}^d \)

- Linear ‘tent’ basis functions
- Quadratic B-spline basis functions

\( C^0 \) continuity

\( C^1 \) continuity
Surfaces: \( S(\xi, \eta) = \sum_{i,j=1}^{N,M} c_{i,j} \hat{B}_i(\xi) \hat{B}_j(\eta) : [0,1] \times [0,1] \to \mathbb{R}^d \)

Bilinear basis functions

Biquadratic B-spline basis functions
Matrix structure for single-patch domain

Biquadratic B-spline basis functions
Matrix structure for multi-patch domain
Isogeometric Analysis in a nutshell

\[ \int_{\tilde{\Omega}_k} \ldots |\det J_k| d\xi \]

Parametric domain \( \tilde{\Omega}_1 \)

Parametric domain \( \tilde{\Omega}_2 \)

Bijective geometry mapping

\[ \int_{\Omega} \ldots dx \]

Physical domain \( \Omega \)
Isogeometric Analysis in a nutshell

Physical domain $\Omega$

Parametric domain $\hat{\Omega}_1$

Parametric domain $\hat{\Omega}_2$

$(\xi^*, 0)$

$(\xi^*, 1)$

$(x^*, y^*)$

$(x^*, y^*)$
Advanced spline technologies: THB splines

Figure 15: L-shape: The marking parameters, the Bézier meshes and the sparsity patterns of the stiffness matrices after refinement steps for all (a)-(d) refinement strategies. The safe refinement strategies result in well graded meshes, the greedy refinement strategies in more unstructured meshes. Again, the greedy THB-spline refinement creates the stiffness matrix with the highest density and interaction.

Application of THB splines

The parametric length functional is given by:

\[ Q_l(s) = \iint_{\hat{\Sigma}} k \frac{\partial u}{\partial s} k^2 + k \frac{\partial v}{\partial s} k^2 \, du \, dv, \]

the orthogonality functional is:

\[ Q_o(s) = \iint_{\hat{\Sigma}} k \frac{\partial u}{\partial s} k \frac{\partial v}{\partial s} k \, du \, dv, \]

and the Winslow functional is:

\[ Q_w(s) = \iint_{\hat{\Sigma}} \text{tr} g \text{det} g \, du \, dv, \]

where \( g \) is the first fundamental form of the template map \( s \).

The meaning of these quality measures are explained in [11, 12].

For computational purposes, the variational template mapping problem is discretized. The discrete version of the objective function (8) then is minimized by applying an iterative procedure.

3.4 Application to design space exploration of blades

3.4.1 Design process

The spline maps solving the template mapping problem will be used as deformation fields in order to set up a stepwise construction of a volumetric representation of the air-passage of an engine. Figure 2 shows two rows of a low pressure turbine (LPT) and an example of an air-passage volume, which is highlighted in blue.
Design-Through-Analysis for

TWIN-SCREW COMPRESSORS
Long-term vision

- Develop a computer code for the efficient geometry modelling, simulation and optimization of rotary twin-screw compressors and expanders

Source: Chair of Fluidics, TU Dortmund University, DE
A challenging industrial application

**Geometry modelling**
- Counter-rotating helical rotors
- Narrow clearances (<0.4mm)
- Complex deforming fluid domain

**Multi-physics simulation**
- Compressible high-speed flow
- Thermal expansion of solids
- Extension: injection of oil, …
Co-design of geometry model and simulation pipeline

**Co-design principles**
- No topology changes (casing-to-rotor)
- Exploit block structure of matrices
- Keep design simple and extendible
- Support heterogeneous hardware
Design-Through-Analysis for twin-screw compressors

GEOMETRY MODELLING
Test cases
Test case #1: rotor-casing passage
Test case #2: separator with CUSP points
Test case #3: rotating twin rotors
Test case #4: rotating twin screws
Design-Through-Analysis for twin-screw compressors

ISOGEOMETRIC FLOW SOLVER
Compressible Euler equations

\[
\frac{\partial}{\partial t} \begin{bmatrix} \rho \\ \rho u \\ \rho E \end{bmatrix} + \nabla \cdot \begin{bmatrix} \rho u \\ \rho u \otimes u + pJ \\ (\rho E + p)u \end{bmatrix} = 0
\]

\[u = \begin{bmatrix} u_1 \\ \vdots \\ u_{d+2} \end{bmatrix}, \quad F(u) = \begin{bmatrix} f_1^1 & \cdots & f_1^d \\ \vdots & \ddots & \vdots \\ f_{d+2}^1 & \cdots & f_{d+2}^d \end{bmatrix}\]

- Equation of state for an ideal gas (isentropic index \(\gamma = 1.4\) for dry air)

\[p(\rho, e) = (\gamma - 1)\rho e, \quad \rho e = (\rho E - 0.5\rho \|u\|^2)\]

- Flux-Jacobian matrix \(A(U) = \frac{\partial F(U)}{\partial U}\), homogeneity property \(F(\lambda U) = \lambda F(U)\)
Variational formulation

- Find solution $U(\cdot, t)$ at fixed time $t$ such that for all test functions $W$

$$
\int_{\Omega} W \partial_t U - \nabla W \cdot F(U) \, dx + \int_{\Gamma} W G(U, U^*) \, ds = 0
$$

- Boundary fluxes

$$
G(U, U^*) = \begin{cases} 
[0, p\mathbf{n}, 0]^T & \text{at solid walls} \\
0.5(F(U) + F(U^*)) \cdot \mathbf{n} & \text{otherwise} \\
-0.5|A(\text{Roe}(U, U^*))| \cdot \mathbf{n} & \text{otherwise}
\end{cases}
$$
Discretization

- Find solution $U_h(\cdot, t)$ at fixed time $t$ such that for all test functions $W_h$

  $$
  \int_\Omega W_h \partial_t U_h - \nabla W_h \cdot F_h(U) \, dx + \int_\Gamma W_h G_h(U, U^*) \, ds = 0
  $$

- Fletcher’s group approximation (CMAME ‘83)

  $$
  U_h(x, t) = \sum_{j=1}^N B_j(x)U_j(t), \quad B_j = \hat{B}_j \circ x^{-1}
  $$

  $$
  F_h(U(x, t)) = \sum_{j=1}^N B_j(x)F_j(t), \quad F_j = F(U_j)
  $$
Semi-discrete formulation

\[
\begin{bmatrix}
M & \vdots & M \\
\end{bmatrix}
\begin{bmatrix}
\dot{u}_1 \\
\vdots \\
\dot{u}_{d+2}
\end{bmatrix}
= 
\begin{bmatrix}
f^1_1 & \cdots & f^d_1 \\
f^1_{d+2} & \cdots & f^d_{d+2}
\end{bmatrix}
\begin{bmatrix}
C^1 \\
C^d
\end{bmatrix}
+ 
\begin{bmatrix}
g^1_1 & \cdots & g^d_1 \\
g^1_{d+2} & \cdots & g^d_{d+2}
\end{bmatrix}
\begin{bmatrix}
S^1 \\
S^d
\end{bmatrix}
= 0
\]

- Constant coefficient matrices

\[M = \left\{ \int_\Omega B_i B_j d\mathbf{x} \right\}_{i,j=1}^{N}, \quad C^k = \left\{ \int_\Omega \partial_k (B_i) B_j d\mathbf{x} \right\}_{i,j=1}^{N}, \quad S^k = \left\{ \int_\Omega B_i B_j n d\mathbf{s} \right\}_{i,j=1}^{N}\]

are pre-assembled using Gauss quadrature and stored to efficiently form the divergence term as SpMV-operation when it is required.
From the programmer’s perspective

\[ M[\dot{u}_1 \cdots \dot{u}_{d+2}] - [C^1 \cdots C^d] \begin{bmatrix} f_1^1 & \cdots & f_{d+2}^1 \\ \vdots & \ddots & \vdots \\ f_1^d & \cdots & f_{d+2}^d \end{bmatrix} + [S^1 \cdots S^d] \begin{bmatrix} g_1^1 & \cdots & g_{d+2}^1 \\ \vdots & \ddots & \vdots \\ g_1^d & \cdots & g_{d+2}^d \end{bmatrix} = 0 \]

1 x d+2 block vector of dense vectors

1 x d block vector of sparse matrices

d x d+2 block matrix of function expressions
FDBB: Fluid dynamics building blocks

Low-level API

- **Unified function wrappers** to the core functionality of established HPC linear algebra libraries, e.g. ArrayFire, Blaze, Eigen, VexCL

- **Compile-time** block linear algebra backend with full support for
  - Dense vectors
  - Sparse matrices
  - Function expressions

Example

```cpp
vex::vector<double> x, y;
vex::sparse::matrix<double> Cx, Cy;

BlockMatrix<...,1,2> Mat(Cx, Cy);

auto Expr = make_BlockExpr<2,1>(cos(x)+y, sin(x)-y);

BlockRowVector<...,1> Vec = Mat * Expr;
```

Loops are unrolled at compile time and fused in a single compute kernel
High-level API

- C++ expression templates for
  - Variables & Riemann invariants
  - Fluxes with ‘generic’ pressure
  - Equations of state

Example

```cpp
using eos = EOS::idealGas<double,
    ratio<7,2>,
    ratio<5,2>>;

// 1x4 dim conservative state vector
using var = Variables<eos,2,Conservative>;
auto U = create<vex::vector<double>,4>(N);

// 2x4 dim inviscid flux tensor
using flx = Fluxes<var>;
auto F = flx::inviscid(U);

// Explicit solution update
U += dt * Mat * F
```
Auto-generation of device-optimized CFD kernels

double rhs_6_sum = 0;
{
    for(size_t j = 0; j < rhs_6_ell_width; ++j)
    {
        int nnz_idx = idx + j * rhs_6_ell_pitch;
        int c = rhs_6_ell_col[nnz_idx];
        if (c != (int)(-1))
        {
            int idx = c;
            rhs_6_sum += rhs_6_ell_val[nnz_idx] * ((prm_tag_2_1[idx] * prm_tag_2_1[idx]) / prm_tag_0_1[idx]) +
            ( rhs_6_x_4 * (prm_tag_0_1[idx] * ( prm_tag_3_1[idx] / prm_tag_0_1[idx] ) -
                          (5.000000000000000e-01) * ( (prm_tag_1_1[idx] * prm_tag_1_1[idx]) + (prm_tag_2_1[idx] *
                          prm_tag_2_1[idx] ) ) ) / ( prm_tag_0_1[idx] * prm_tag_0_1[idx] ) );
        } else break;
    }
    if (rhs_6_csr_ptr)
    ...

\[ \rho u \otimes u + pJ \]
## Heterogeneous HPC support

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>CUDA</th>
<th>OpenCL</th>
<th>Intel MIC</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayFire</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Armadillo</td>
<td>(X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blaze</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eigen</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTL4</td>
<td>(X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uBLAS</td>
<td>(X)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VexCL(^1)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ViennaCL</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) source code is generated and JIT-compiled; \(^2\) Maxeler DFEs
Example: Computational performance

- Intel 2xE5-2670 @2.6 GHz
- IBM Power8NVL @4.02 GHz
- IBM + NVIDIA GP100GL
Test case #1: Inviscid compressible flow, $p_{in}:p_{out} = 2:1$
Conclusions

Hardware-oriented Numerics with IGA: **co-design** of geometry and simulation
- IGA package G+Smö: [https://github.com/gismo/gismo](https://github.com/gismo/gismo)
- Open-source FDBB: [https://gitlab.com/mmoelle1/FDBB](https://gitlab.com/mmoelle1/FDBB)

Ongoing and future work
- Extension to turbulent flows and ALE formulation for rotating geometries
- Automatic compute resource scheduling and dynamic load balancing

References

Financial support by EC under GA No 672787 is acknowledged
Recent updates

- **MPI-communication and dynamic load-balancing tool framework**: *joint student or master project (UvA, TUD); topic 2*
  - Cluster inventory tool implemented with support for x86, x86_64, ppc64le, ARM, CUDA, OpenCL, SDAccel, …

- **Sparse band matrix implementation**: *student project TUD*

- **DSL-based algorithmic differentiation**: *PhD project by A. Jaeschke*
  - Gradient-based shape optimization in turbomachinery applications

- **PyG+Smo**: *hobby project of mine*
  - Python wrappers of G+Smo library based on PyBind11