Adjoint-based non-linear optimization of complex systems governed by PDEs using HPC techniques

Oscar Peredo, Mariano Vázquez, Guillaume Houzeaux, José María Cela

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

3rd Conference on Optimization Methods and Software

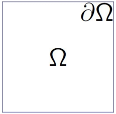
May 13-17, 2012 - Chania, Greece

Outline

Outline

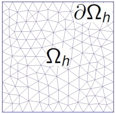
How to "solve" a PDE?

Example: Poisson's equation with boundary conditions in a domain $\Omega \subset \mathbb{R}^2$, u is unknown, f is data (source):



$$
\begin{array}{rcll}
\Delta u(x,y) & = & f(x,y) & ,(x,y) \in \Omega \\
u(x,y) & = & 0 & ,(x,y) \in \partial\Omega
\end{array}
$$

Domain discretization:



$$
\underbrace{\begin{bmatrix} \mathbf{A}^{\Omega_h} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}}_{\mathbf{A}} \mathbf{u} = \underbrace{\begin{bmatrix} \mathbf{b}^{\Omega_h} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{b}}
$$

Main task:	solve $\mathbf{Au} = \mathbf{b}$
Common issue:	better solutions $\rightarrow$ ++elements/nodes $\rightarrow$ *++execution time*
Possible solution:	**High Performance Computing techniques to solve large-scale linear systems** (domain decomposition, parallel matrix-vector ops, programming models MPI/OpenMP, ...)

Where is the optimization?

Example: Poisson's equation with boundary conditions in a domain $\Omega \subset \mathbb{R}^2$, u is unknown, $f(x, y) = d_1 x^2 + d_2 y^2$ is data (source)

$$\begin{array}{rcll} \Delta u(x,y) & = & d_1 x^2 + d_2 y^2 & , (x,y) \in \Omega \\ u(x,y) & = & 0 & , (x,y) \in \partial\Omega \end{array}$$

Linear system:

$$\mathbf{Au} = \mathbf{b}(\mathbf{d})$$

with $\mathbf{d} = (d_1, d_2)$ and dependency $\mathbf{b} := \mathbf{b}(\mathbf{d})$.

$$\text{input: } \mathbf{d} \rightarrow \boxed{\mathbf{Au}\text{=}\mathbf{b}(\mathbf{d})} \rightarrow \text{output: } \mathbf{u}$$

Main task:	find optimal values for **d** such that the PDE solution **u** reaches a defined objective (**inverse problem**)
Common issue:	several resolutions of $\mathbf{Au} = \mathbf{b}(\mathbf{d})$... *++++execution time*
Possible solution:	**gradient-based optimization methods**

Nonlinear optimization problem

- Variables: $\mathbf{d}$: **design variables** and $\mathbf{s}$: **state variables** of the system.
- Constraint function: $R(\mathbf{d}, \mathbf{s}) : \mathbb{R}^{n_d} \times \mathbb{R}^{n_s} \to \mathbb{R}^{n_s}$. Ex: $R(\mathbf{d}, \mathbf{s}) = \mathbf{A}\mathbf{s} - \mathbf{b}(\mathbf{d})$
- Cost function: $J(\mathbf{d}, \mathbf{s}) : \mathbb{R}^{n_d} \times \mathbb{R}^{n_s} \to \mathbb{R}$.
- Constrained optimization problem:

$$\min\{J(\mathbf{d}, \mathbf{s}) : (\mathbf{d}, \mathbf{s}) \in \mathbb{R}^{n_d} \times \mathbb{R}^{n_s}, R(\mathbf{d}, \mathbf{s}) = \mathbf{0}\}$$

- Unconstrained optimization problem: $\mathbf{s} := \mathbf{s}(\mathbf{d})$ and $j(\mathbf{d}) = J(\mathbf{d}, \mathbf{s}(\mathbf{d}))$,

$$\min\{j(\mathbf{d}) : \mathbf{d} \in \mathbb{R}^{n_d}\}$$

- Descent directions: $\mathbf{d}^{k+1} = \mathbf{d}^k + \alpha^k \mathbf{p}^k$, Example: $\mathbf{p}^k = -\nabla_{\mathbf{d}} j(\mathbf{d}^k)$

Main task:	calculate $\nabla_{\mathbf{d}} j(\mathbf{d}^k)$
Common issue:	each evaluation of $j(\mathbf{d} \pm \mathbf{h}_i)$ is expensive (in execution time):
	re-assembling $\mathbf{A}$ or $\mathbf{b}$, and resolution of linear system
Possible solution:	**discrete adjoint method for gradient calculation**

Outline

Discrete adjoint method

Objective: Calculate $\nabla_{\mathbf{d}}\, j(\mathbf{d}^k)$
Step 1. Chain rule applied to cost function $j(\mathbf{d})$:

$$\nabla_{\mathbf{d}}\, j(\mathbf{d}) \quad = \quad \nabla_{\mathbf{d}} J(\mathbf{d},\mathbf{s}) + \nabla_{\mathbf{s}} J(\mathbf{d},\mathbf{s}) \cdot \nabla_{\mathbf{d}}\mathbf{s}(\mathbf{d}) \tag{1}$$

Step 2. Linearization of constraint function $R(\mathbf{d},\mathbf{s}(\mathbf{d}))$ to obtain $\nabla_{\mathbf{d}}\mathbf{s}$:

$$\nabla_{\mathbf{d}} R(\mathbf{d},\mathbf{s}) + \nabla_{\mathbf{s}} R(\mathbf{d},\mathbf{s}) \cdot \nabla_{\mathbf{d}}\mathbf{s}(\mathbf{d}) = \mathbf{0} \tag{2}$$

$$\boxed{\nabla_{\mathbf{d}}\mathbf{s}(\mathbf{d}) = -[\nabla_{\mathbf{s}} R(\mathbf{d},\mathbf{s})]^{-1} \cdot \nabla_{\mathbf{d}} R(\mathbf{d},\mathbf{s})} \tag{3}$$

Step 3. KKT conditions for the constrained opt. problem to obtain $\nabla_{\mathbf{s}} J(\mathbf{d},\mathbf{s})$:

$$\nabla_{\mathbf{d}} J(\mathbf{d},\mathbf{s}) - \boldsymbol{\lambda}_{\mathbf{d}}^T \nabla_{\mathbf{d}} R(\mathbf{d},\mathbf{s}) \quad = \quad \mathbf{0} \tag{4}$$

$$\nabla_{\mathbf{s}} J(\mathbf{d},\mathbf{s}) - \boldsymbol{\lambda}_{\mathbf{s}}^T \nabla_{\mathbf{s}} R(\mathbf{d},\mathbf{s}) \quad = \quad \mathbf{0} \tag{5}$$

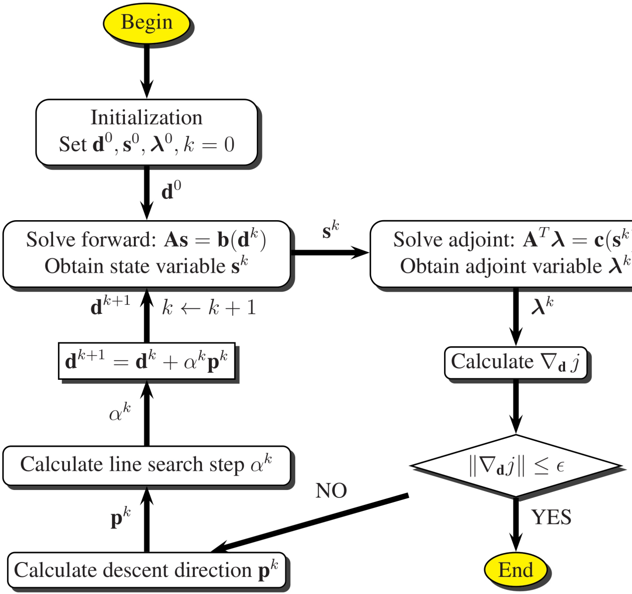
$$\boxed{\underbrace{\nabla_{\mathbf{s}} J(\mathbf{d},\mathbf{s})}_{\mathbf{c}^T} = \boldsymbol{\lambda}_{\mathbf{s}}^T \underbrace{\nabla_{\mathbf{s}} R(\mathbf{d},\mathbf{s})}_{\mathbf{A}}} \tag{6}$$

Step 4. Put (6) and (3) into (1):

$$\nabla_{\mathbf{d}}\, j(\mathbf{d}) \quad = \quad \nabla_{\mathbf{d}} J(\mathbf{d},\mathbf{s}) - \boldsymbol{\lambda}_{\mathbf{s}}^T \nabla_{\mathbf{s}} R(\mathbf{d},\mathbf{s}) [\nabla_{\mathbf{s}} R(\mathbf{d},\mathbf{s})]^{-1} \nabla_{\mathbf{d}} R(\mathbf{d},\mathbf{s})$$

$$\boxed{\nabla_{\mathbf{d}}\, j(\mathbf{d}) = \nabla_{\mathbf{d}} J(\mathbf{d},\mathbf{s}) - \boldsymbol{\lambda}_{\mathbf{s}}^T \nabla_{\mathbf{d}} R(\mathbf{d},\mathbf{s})} \tag{7}$$

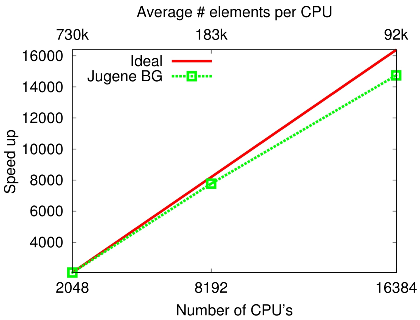
Discrete adjoint method



Outline

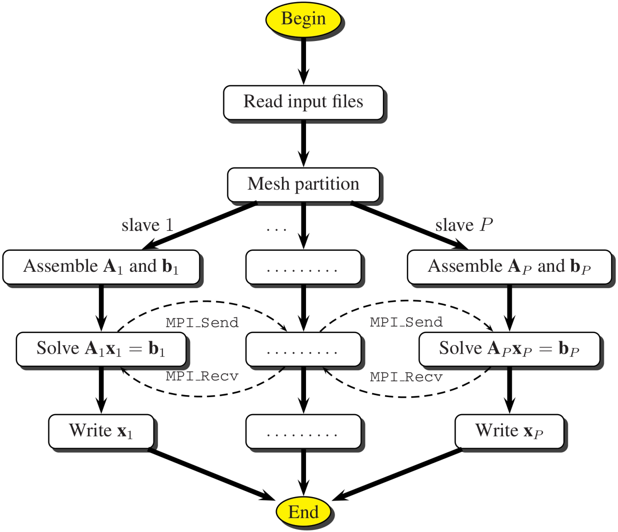
Alya system

- In-house code for High Performance Computational Mechanics
  - (In)Compressible flows
  - Thermal flows
  - Non-linear Solid mechanics
  - ...
- Parallel code from scratch (Fortran90+MPI+OpenMP) implementing Finite Element method.
- Structured software design:
  - Kernel (*the core*)
  - Modules (*the physics*)
  - Services (*the toolbox*)
- Scalability is a requirement for any new component of the system.



Benchmark: mesh of 1.6 billion tetrahedra, incompressible flow on an aneurism geometry

Alya system



Begin
Read input files
Mesh partition
slave 1
...
slave P
Assemble $\mathbf{A}_1$ and $\mathbf{b}_1$
....
Assemble $\mathbf{A}_P$ and $\mathbf{b}_P$
MPI_Send
MPI_Send
Solve $\mathbf{A}_1\mathbf{x}_1 = \mathbf{b}_1$
....
Solve $\mathbf{A}_P\mathbf{x}_P = \mathbf{b}_P$
MPI_Recv
MPI_Recv
Write $\mathbf{x}_1$
....
Write $\mathbf{x}_P$
End

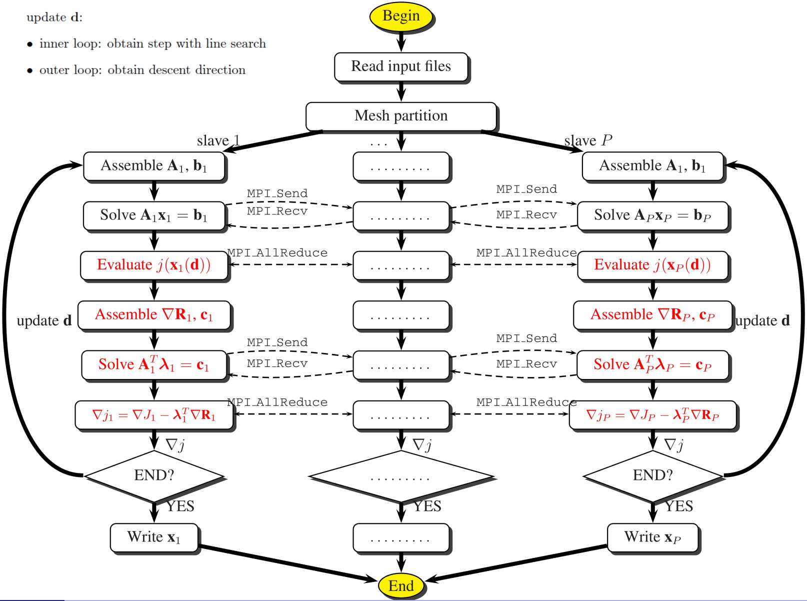
Alya system

How can we exploit the structure of this code to implement the discrete adjoint method, preserving the time execution scalability?

Alya system + discrete adjoint

update **d**:

- inner loop: obtain step with line search
- outer loop: obtain descent direction



Outline

Test example

The test example problem is:

$$\begin{array}{lll} \text{minimize}_{\mathbf{d}} & \frac{1}{2}\int_{\Omega}(u(x,y)-u^{obs}(x,y))^2dxdy & \\ \text{subject to} & L(u(x,y)) = f(\mathbf{d},x,y) & (x,y)\in\Omega \\ & u(x,y) = u^{obs}(x,y) & (x,y)\in\partial\Omega \end{array}$$

with

$$L(u) = \rho c_p \vec{v}\cdot\nabla u - \nabla\cdot(\kappa\nabla u) + s\,u$$

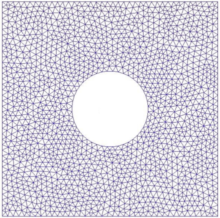
$$f(\mathbf{d},x,y) = \sum_{i=1}^{n_d}(d_i - d_i^{target})^2\,p_i(x,y)$$

the stationary convection-diffusion linear (elliptic) operator L and the source f defined with $p_i \in C(\partial\Omega)\cup L^2(\Omega)$ functions that are independent of $\mathbf{d}$.

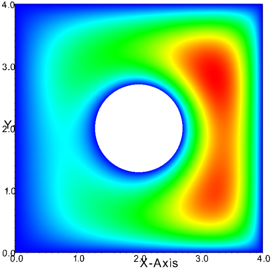
Theoretical solution: $\boxed{(u(\mathbf{d}^*),\mathbf{d}^*) = (u^{obs},\mathbf{d}^{target})}$ (weak maximum principle)

Test example

- 2D-Domain, u_{obs} constant
- # elements: small(330K), medium(1.3M), large(3.7M)
- # design variables: 1, 5, 10, 50
- # processes (CPUs) 1, 2, 4, 8, 16, 32, 64, 128, 256, 512
- Distributed-memory supercomputer MareNostrum (10240 CPUs,2.3GHz)
- Iterative method: GMRES (1000 fixed iterations)
- Descent direction method: steepest descent (30 fixed outer-iterations)

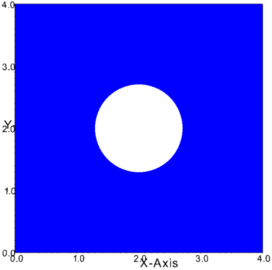


Possible initial state using $u_{obs} = 100$



$u_{\max} = 2471$, $u_{\min} = 100$

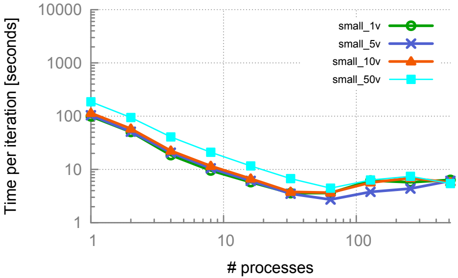
Final state using $u_{obs} = 100$

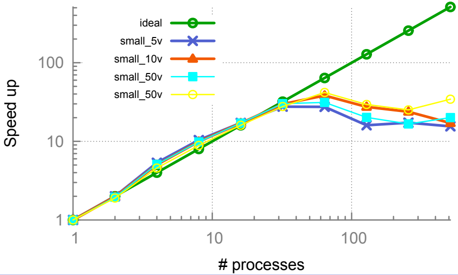


$u_{\max} = 100$, $u_{\min} = 100$

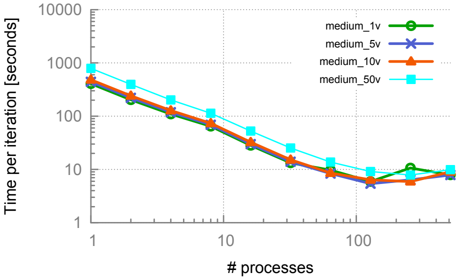
Outline

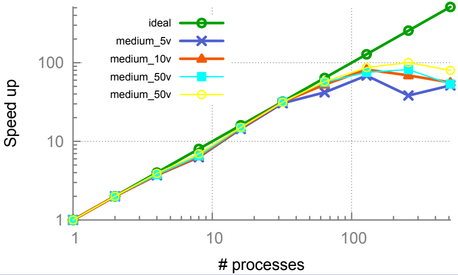
330K elements (small)



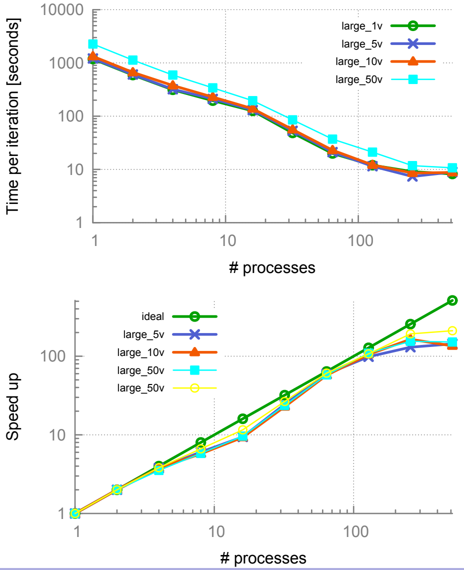


1.3M elements (medium)





3.7M elements (large)



Outline

Conclusions

- Code structure is preserved ”with minimal modifications” (still not automatic differentiation of $\nabla_s J$, $\nabla_d J$ or $\nabla_d R$).
- Execution time scalability is preserved.
- Several design variables can be optimized at *cheap* computational cost.
- Maximum number of design variables is limited by the computer’s physical memory (in this case, the RAM of each compute node).

Outline

Future work

- This work is in the context of a final project for a master's degree.
- Handling external constraints of the design variables (constrained optimization methods: SQP, barrier/penalization, others...).
- Transient and nonlinear problems (addition of a temporal term $\frac{\partial u}{\partial t}$ in the PDE, example: Navier-Stokes equation).
- Explore new computer architectures with this algorithm (GPUs, Cell B/E, energy-efficient new processors (`http://www.montblanc-project.eu`), ...)
- Industrial applications: optimal shape design (aeronautics), inverse problems in geophysics (petroleum), optimal distribution of eolic parks (renewable energy), ...

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thanks for your attention!

Contact: `oscar.peredo [at] bsc.es`

`http://www.bsc.es`