Making parallel PDE software by object-oriented programming

Xing Cai

Simula Research Laboratory, Norway

Department of Informatics, University of Oslo

May 17, 2007

Outline

List of Topics

Motivation

- Context: parallelize an existing serial solver of partial differential equations (PDEs)
- If the serial solver already uses standard numerial libraries, parallelization may be relatively easy
- Otherwise not easy, need extensive parallelization effort
- Solution: a general and flexible parallelization framework

Motivation

- Context: parallelize an existing serial solver of partial differential equations (PDEs)
- If the serial solver already uses standard numerial libraries, parallelization may be relatively easy
- Otherwise not easy, need extensive parallelization effort
- Solution: a general and flexible parallelization framework
  - extensive reuses of serial PDE software
  - simple programming effort by the user
  - possibility of hybrid features and different linear solvers

Motivation

- Context: parallelize an existing serial solver of partial differential equations (PDEs)
- If the serial solver already uses standard numerial libraries, parallelization may be relatively easy
- Otherwise not easy, need extensive parallelization effort
- Solution: a general and flexible parallelization framework
  - extensive reuse of serial PDE software
  - simple programming effort by the user
  - possibility of hybrid features in different local areas

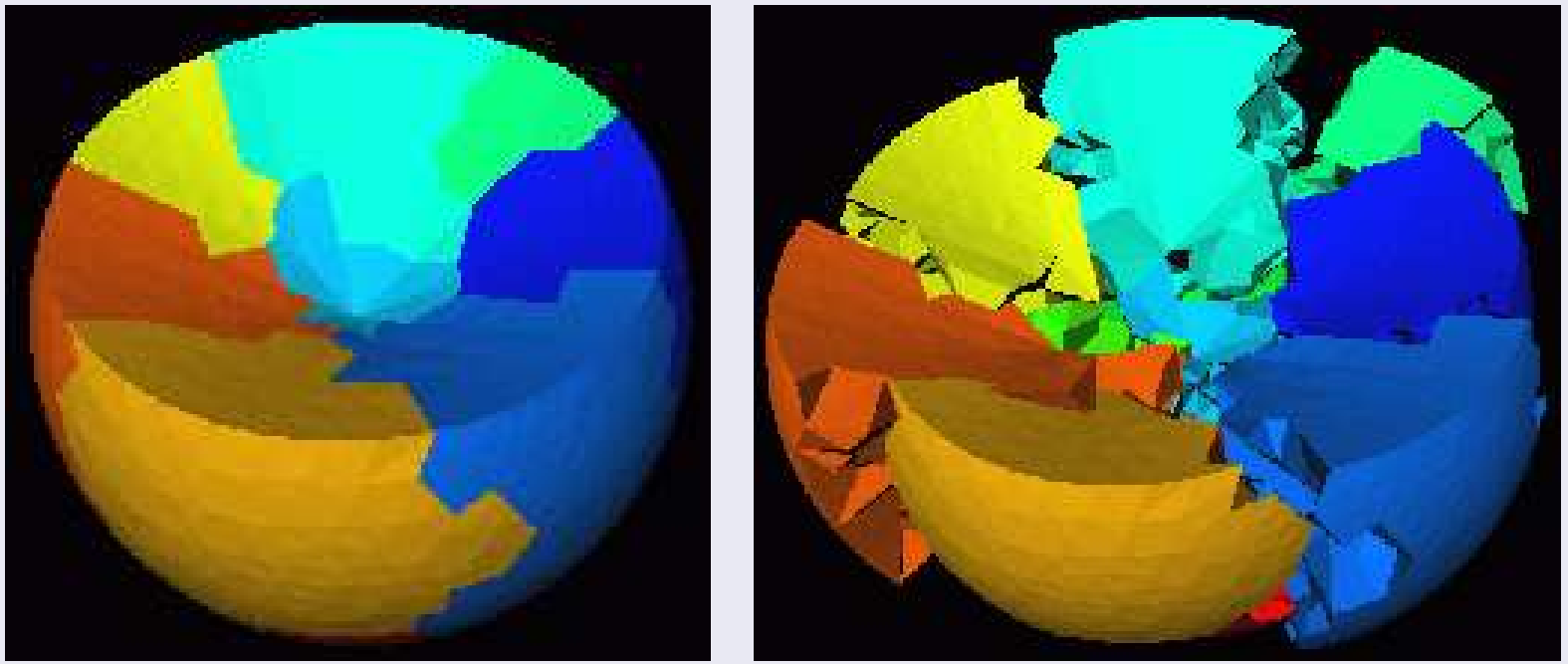
Motivation

- Context: parallelize an existing serial solver of partial differential equations (PDEs)
- If the serial solver already uses standard numerial libraries, parallelization may be relatively easy
- Otherwise not easy, need extensive parallelization effort
- Solution: a general and flexible parallelization framework
  - extensive reuse of serial PDE software
  - simple programming effort by the user
  - possibility of hybrid features in different local areas

Motivation

- Context: parallelize an existing serial solver of partial differential equations (PDEs)
- If the serial solver already uses standard numerial libraries, parallelization may be relatively easy
- Otherwise not easy, need extensive parallelization effort
- Solution: a general and flexible parallelization framework
  - extensive reuse of serial PDE software
  - simple programming effort by the user
  - possibility of hybrid features in different local areas

Basic idea: Parallelization via subdomains



- Domain decomposition: one global solution domain is divided into many subdomains
- Each subdomain: (relatively) independent working unit
- Collaboration between the subdomains: communication

Parallelization via subdomains

- The global computational mesh is divided into a set of subdomain meshes
- One processor gets one subdomain mesh
- Local discretization on each processor, totally independent of each other
- All global matrices/vectors are represented collectively by physically distributed subdomain matrices/vectors
- Parallel solution of linear systems: often readily achievable by standard parallel numerical libraries (PETSc, Trilinos, PSPARALIB, hypre)
  - parallel Krylov iterative solvers
  - parallel "black-box" algebraic preconditioners

Some difficult situations

- The serial PDE solver may be a legacy code
  - old-style programming
  - lots of low-level and tightly coupled loops
  - standard parallel libraries are not applicable
- The serial PDE solver may use a special preconditioner
  - standard parallel algebraic preconditioners are not good enough
  - good parallel numerical performance relies on a special parallel preconditioner
- Remedy: allow a user-controllable parallelization!
  - additive Schwarz methods as the numerical foundation
  - object-oriented programming as enabling technique

List of Topics

Additive Schwarz

$$\Omega = \cup_{s=1}^{P} \Omega_s$$

- The global domain is decomposed into a set of subdomains with overlap
- Solution of $\mathcal{L}_\Omega(u) = f_\Omega$ is found as $u^0, u^1, \ldots, u^i$

$$\mathcal{L}_{\Omega_s}(u_s^i) = f_{\Omega_s}^i \quad 1 \leq s \leq P \quad \Rightarrow \quad u^i = \cup_{s=1}^{P} u_s^i$$

- A global iteration among all subdomains
  - During each iteration a subdomain independently updates its local solution
  - Exchange of local solutions between neighboring subdomains at end of each iteration

More on additive Schwarz

- Simple algorithmic structure (no special interface problems)
  - subdomain solver, communication and convergence control are basic building blocks
  - can reuse existing serial PDE solver as the subdomain solver
- Applicable both as stand-alone solver or preconditioner
  - Originally well-known as a parallel numerical strategy for solving large linear systems
  - We apply DD at "software level" (not at "linear-algebra level")
- No global matrices/vectors exist, all represented by the collection of subdomain matrices/vectors
- Neighboring subdomain meshes may be non-matching and/or of different types

Schwarz methods can thus work as the numerical foundation of parallelization

List of Topics

A generic library of Schwarz methods

Schwarz methods: a general approach to solving PDEs in parallel, therefore a **generic** library can be programmed

- Object-oriented programming is well suited
- Generic components: subdomain solvers and a global administrator
- `class SubdomainSolver`: generic interface of a subdomain solver, only declaration of standard functions, no implementation
- `class Administrator`: implementation of generic functions for invoking communication and checking global convergence

Parallelization of a serial PDE solver

- Assume `class MySolver` as an existing serial PDE solver
- New implementation work task 1:
  `class My_SubdSolver : public SubdomainSolver, public MySolver`
  - Double inheritance
  - Implement the generic functions of `SubdomainSolver` by calling/extending functions of `MySolver`
  - Mostly code reuse, little new programming
- New implementation work task 2:
  `class My_Administrator : public Administrator`
  - Extend `Administrator` to handle problem-specific details
  - Mostly "cut and paste", little new programming

List of Topics

Example 1: simulating ocean wave propagation

Boussinesq wave equations

$$\begin{aligned}
\frac{\partial \eta}{\partial t} + \nabla \cdot (H + \alpha\eta)\nabla\phi + \epsilon H \left(\frac{1}{6}\frac{\partial \eta}{\partial t} - \frac{1}{3}\nabla H \cdot \nabla\phi \right) \nabla H &= 0 \\
\frac{\partial \phi}{\partial t} + \frac{\alpha}{2}\nabla\phi \cdot \nabla\phi + \eta - \frac{\epsilon}{2} H \nabla \cdot \left(H \nabla \frac{\partial \phi}{\partial t} \right) + \frac{\epsilon}{6} H^2 \nabla^2 \frac{\partial \phi}{\partial t} &= 0
\end{aligned}$$

- Wave propagation with nonlinear and dispersive effects
- Useful for studying tsunamis
- Ocean-scale simulations requires "smart computing"
  - some local regions: advanced numerics, high mesh resolution
  - remaining areas: simple numerics, low mesh resolution

Starting point for parallelization

- Legacy F77 code:
  - finite difference discretization
  - only applicable to rectangular domains
  - very efficient code
- Modern C++ Boussinesq code
  - finite element discretization
  - flexible with repect to unstructured meshes, adaptivity, iterative linear solvers
  - slower than the F77 code
- Difficult to parallelize both codes directly
  - especially, the F77 code has many levels of nested loops
- Objective: a hybrid parallel Boussinesq solver
  - we want to use both codes in the parallelization

Parallelization

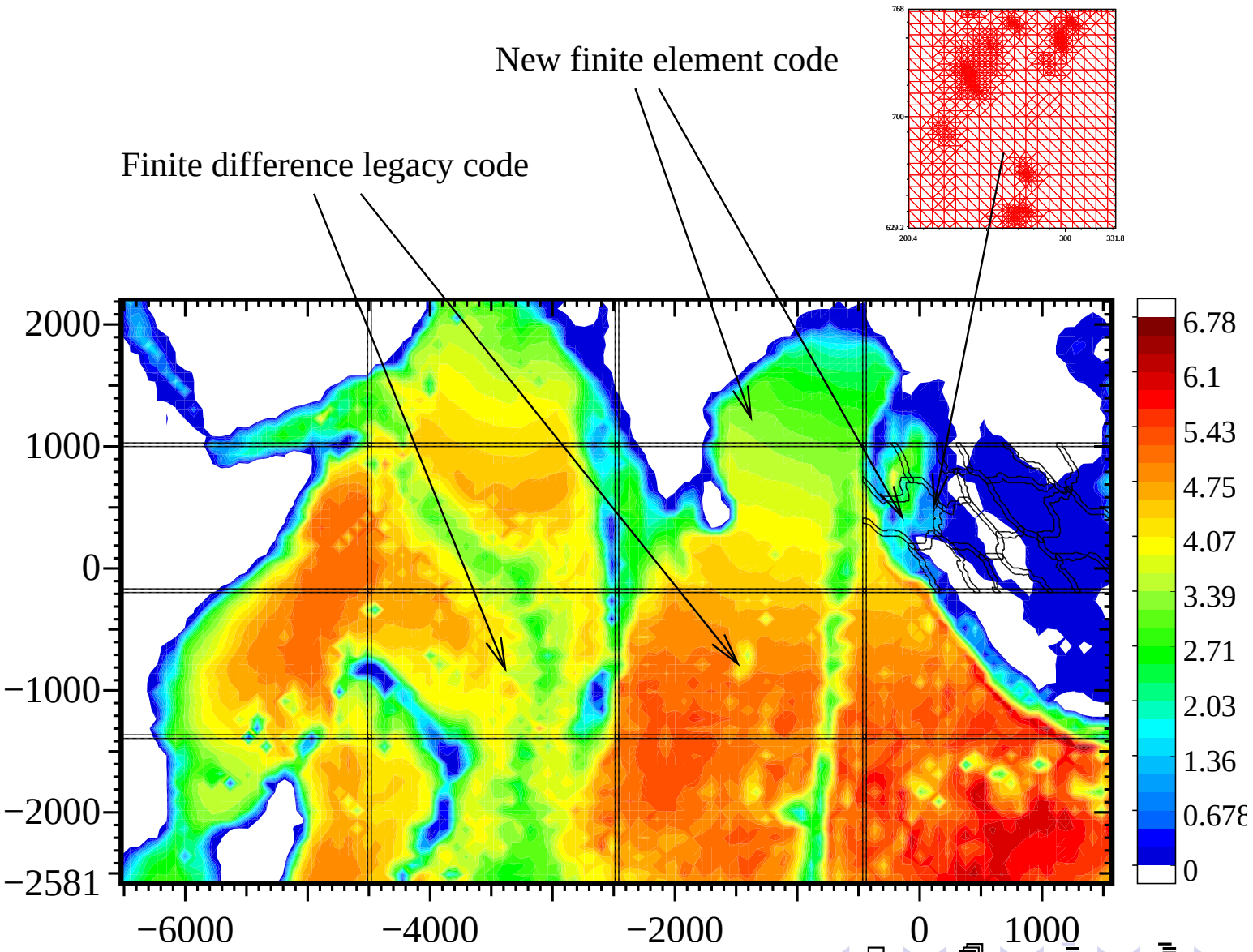
- Wrappers of two subdomain Boussinesq solvers
  - `class SubdomainBQFEMSolver : public Boussinesq, public SubdomainSolver`
  - `class SubdomainBQFDMSolver : public SubdomainSolver` (calling F77 subroutines internally)
- Implementation of a simple "controller":
  - `HybridBQSolver : public Administrator`
  - containing details specific for handling Boussinesq equations by additive Schwarz
- Total programming effort is rather limited, due to use of the generic Schwarz library

Some scalability measurements

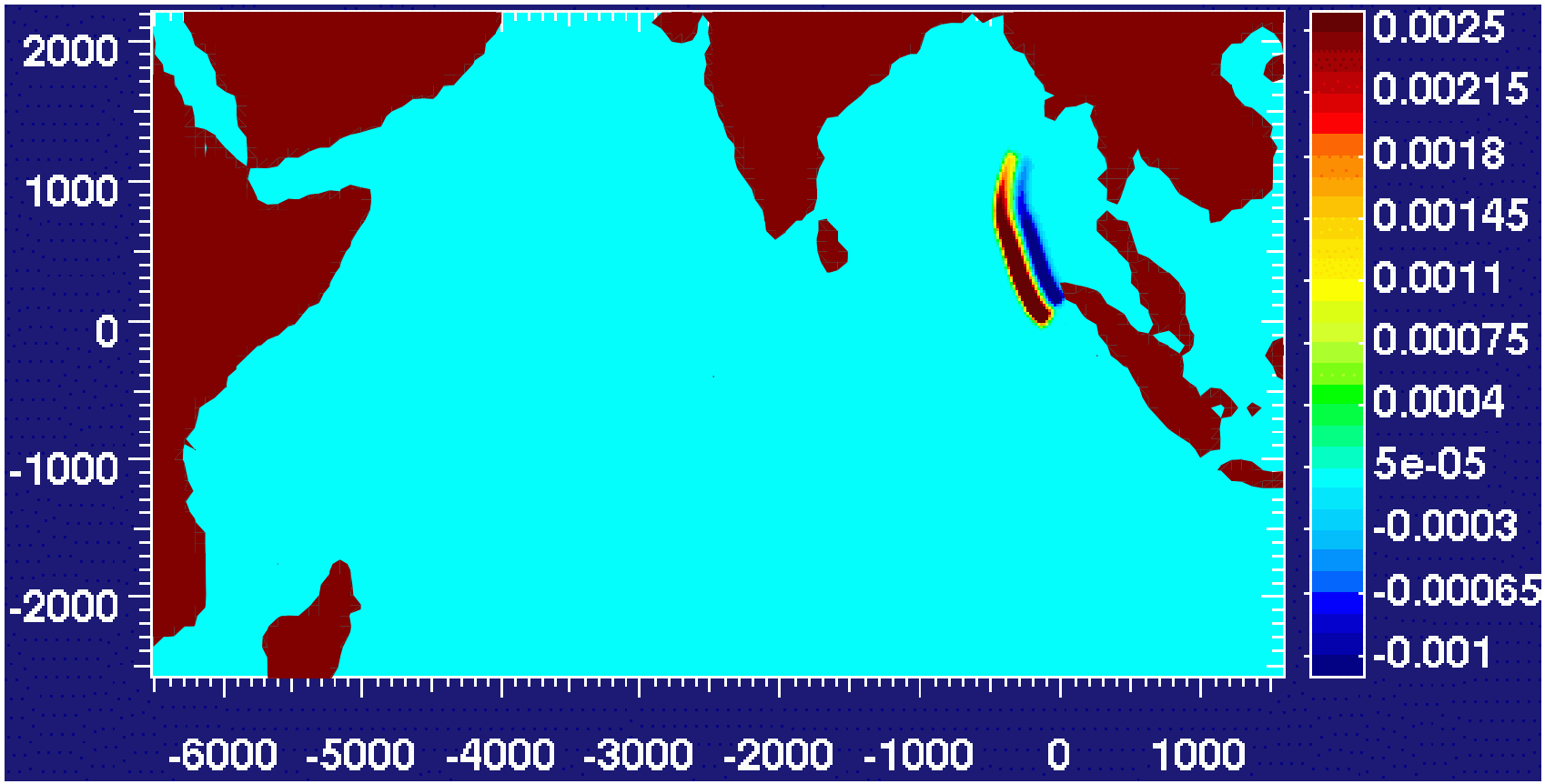
- Simple test case:
  - use of C++ code in every subdomain
  - good load balance
- Fixed global problem size: 11200×960
- Wall-time measurements on IBM SP of 160 Schwarz iterations

# subds	Wall-time (s)
16	881.91
32	458.86
64	243.37
128	128.91
256	70.84

Simulation of the Indian Ocean Tsunami

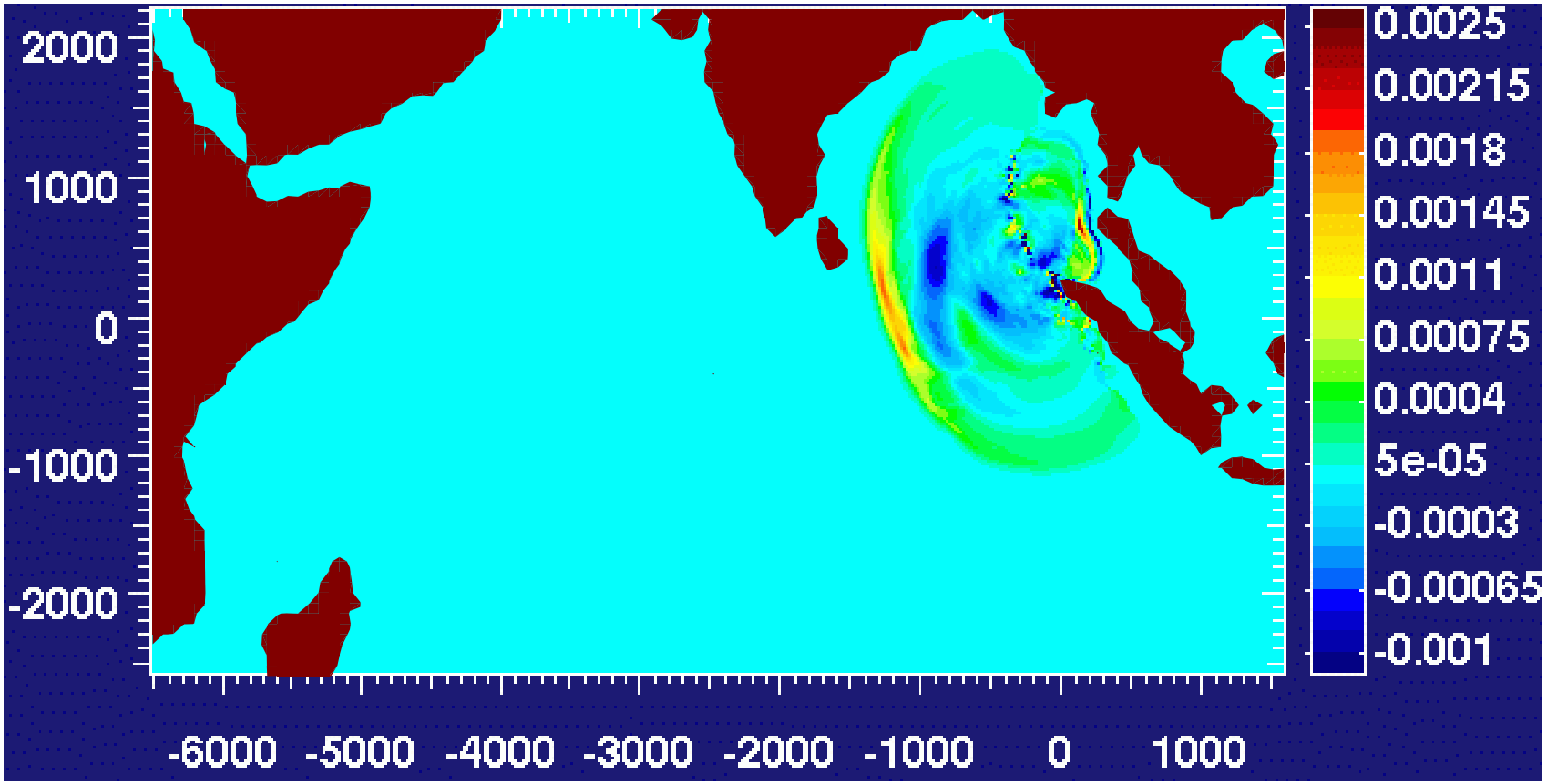


Preliminary parallel simulation



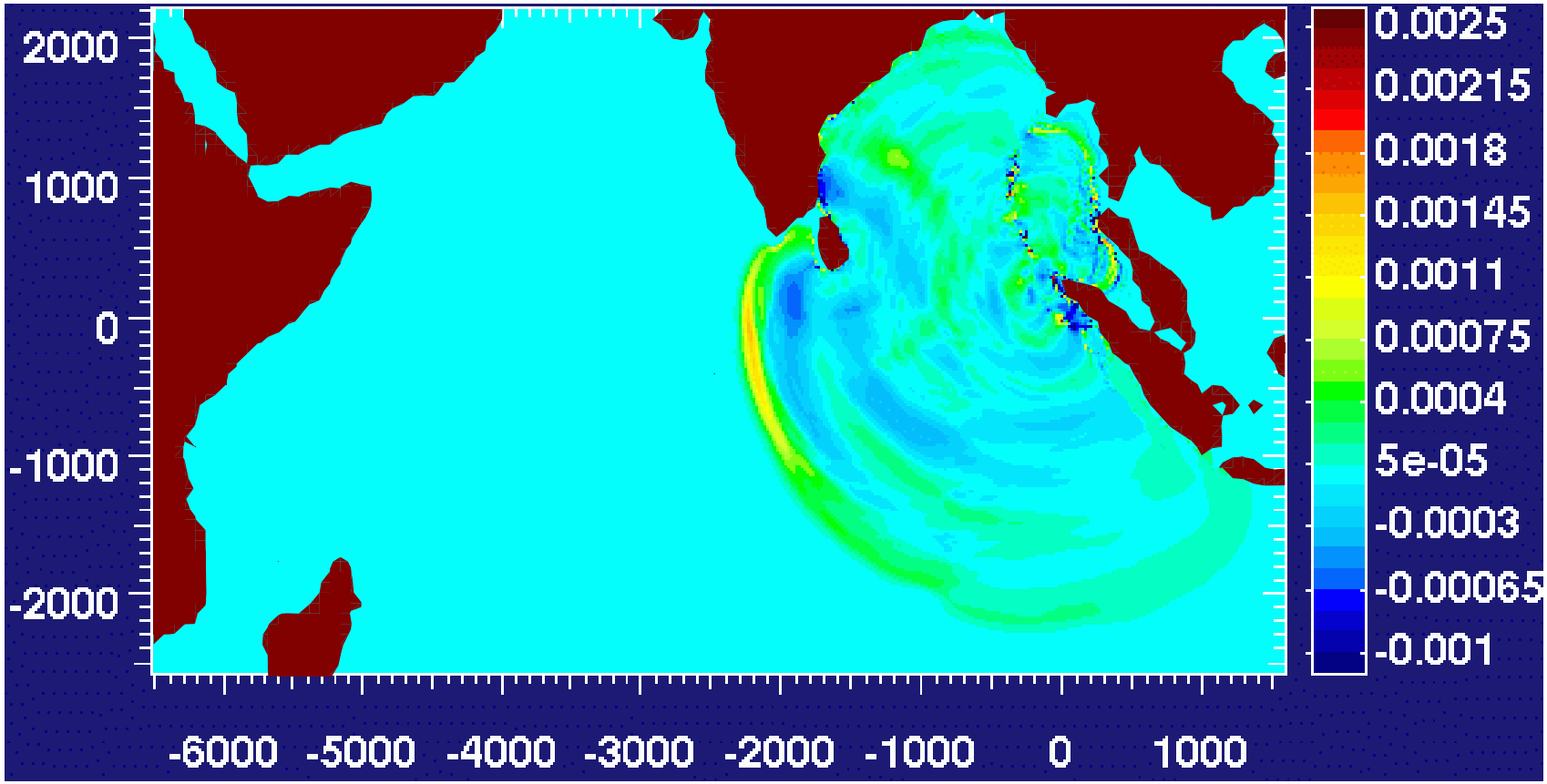
Initial wave elevation after the earthquake

Coarse-mesh simulation snapshot 1



After 1.4 hours

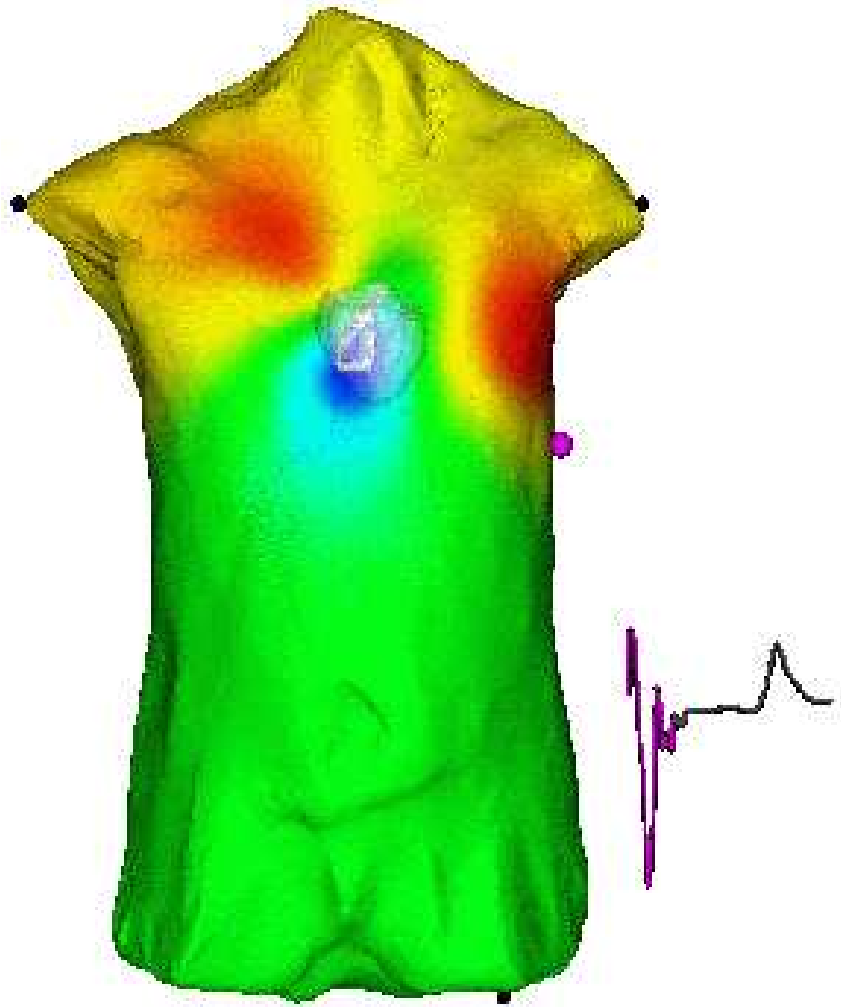
Coarse-mesh simulation snapshot 2



After 2.8 hours

Example 2: simulating ECG

- Ionic current $\Rightarrow$ heart muscle contraction $\Rightarrow$ pumping heart
- Electrical activity inside heart is measurable by electrocardiogram (ECG)
- Numerical simulation - diagnostic future tool (?)



Mathematical model

The BiDomain Model in the heart—intracellular space and extracellular space

$$\begin{aligned} \chi C_{\mathrm{m}} \frac{\partial v}{\partial t} + \chi I_{\mathrm{ion}}(v, \mathbf{s}) &= \nabla \cdot (M_i \nabla v) + \nabla \cdot (M_i \nabla u_e) \\ 0 &= \nabla \cdot (M_i \nabla v) + \nabla \cdot ((M_i + M_e) \nabla u_e) \end{aligned}$$

- Transmembrane potential: $v = u_i - u_e$
- Extracellular potential: u_e
- Ionic current $I_{\mathrm{ion}}(v, \mathbf{s})$ relies on ODEs in H

$$\frac{\mathrm{d}\mathbf{s}}{\mathrm{d}t} = \mathbf{F}(v, \mathbf{s})$$

Computational challenges

- Two coupled PDEs in H; a system of ODEs at every point
- Realistic 3D geometries (heart and torso)
- Need high resolution in space and time
  - Desired spatial resolution: 0.2mm $\sim 50 \times 10^6$ mesh points
  - Desired temporal resolution: 0.1ms $\sim$ 10000 time steps
  - Example: # degrees of freedom $= (2 + 31) \times 50 \times 10^6$ for H!

Numerical strategy

- Operator splitting $\Rightarrow$ ODE part + PDE part
- Solve the ODE system, point by point
- Solve the two PDEs simultaneously:

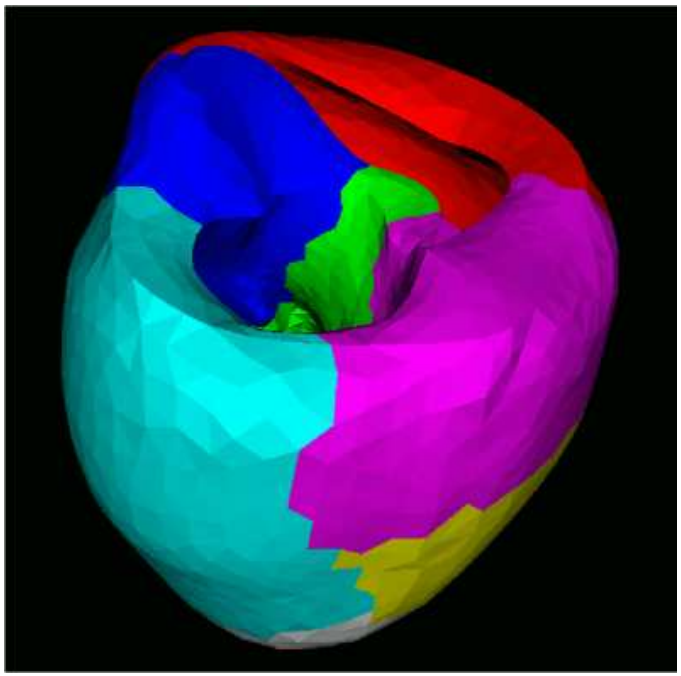
$$\begin{aligned} \chi C_{\mathrm{m}} \frac{v^l - \tilde{v}^{l-1}}{\Delta t} + \chi I_{\mathrm{ion}} &= \nabla \cdot (M_i \nabla v^l) + \nabla \cdot (M_i \nabla u_e^l) \\ 0 &= \nabla \cdot (M_i \nabla v^l) + \nabla \cdot ((M_i + M_e) \nabla u_e^l) \end{aligned}$$

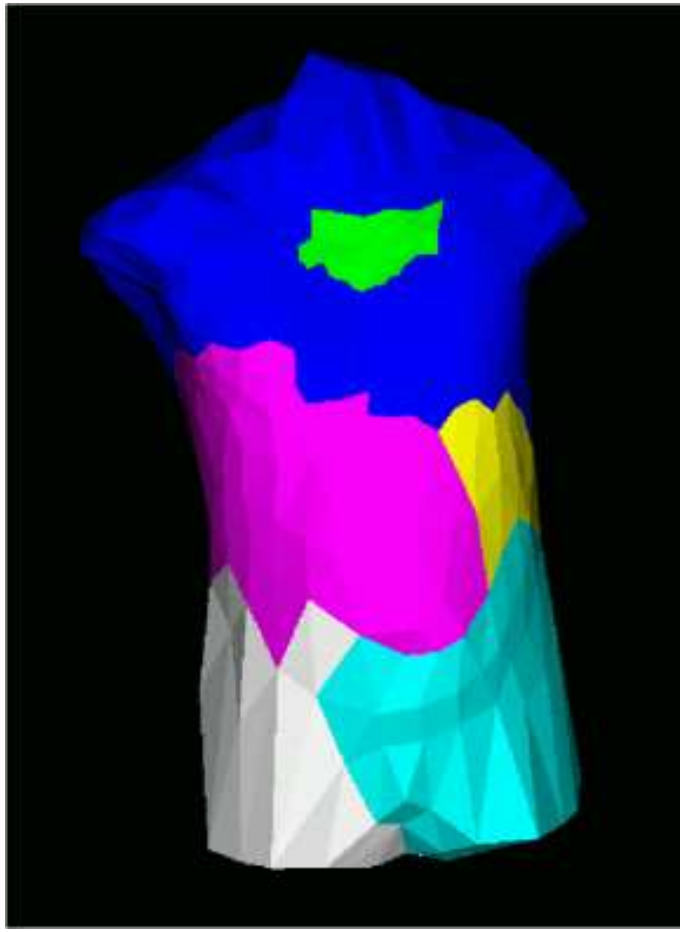
- 2×2 block linear system:

$$\begin{bmatrix} \mathbf{I} + \theta \Delta t \mathbf{A}_{\mathrm{v}} & \theta \Delta t \mathbf{A}_{\mathrm{v}} \\ \theta \Delta t \mathbf{A}_{\mathrm{v}} & \theta \Delta t \mathbf{A}_{\mathrm{u}} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

Parallelization

A special parallel block-preconditioner by additive Schwarz





Parallel ECG simulations

- Parallel programming effort: 10% of the original serial programming effort
- Additive Schwarz (with coarse grid correction) $\Rightarrow$ # CG iterations independent of # mesh points and P
- On a Linux cluster: # tetrahedra up to 34,485,752, # mesh points up to 5,762,729
- On an SGI Origin 3800 system: # tetrahedra up to 252,143,960; # mesh points up to 42,073,915

Summary

- Subdomains give rise to a natural way of parallelizing PDE solvers
- Additive Schwarz may be useful if
  - efficient parallel preconditioners are desired, and/or
  - high-level parallelization of legacy PDE code is desired, and/or
  - a parallel hybrid PDE solver is desired
- Most of the parallelization work is generic, can thus be implemented as libraries
- Languages like C++ (or Python) may greatly simplify the user programming effort