Projet DEDALES

Décomposition de domaines algébriques et géométriques pour les écoulements souterrains

M. Kern

8 décembre 2016

Outline

Outline

Overall objectives

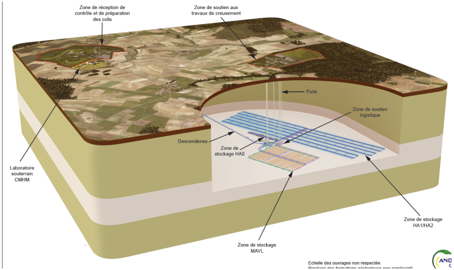
Develop high performance code for
the simulation of complex flow in the subsurface

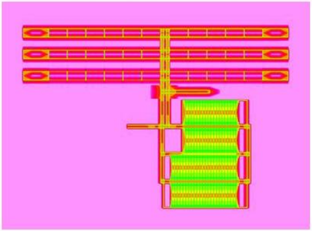
Application challenge

High performance simulation around a nuclear waste disposal site

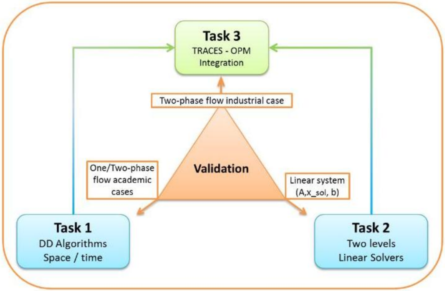
Scienctific challenge

Bridge the gap between hierarchical physical models and hierarchichal computer architectures





Organization – objectives



- Demonstrate the potential of domain decomposition methods for exploiting the upcoming hierarchical and heterogeneous architectures
- Formulate space-time DD methods for two-phase flows (non-linearity, discontinuous capillary pressure)
- Develop hybrid (MPI + OpenMP) solver over a runtime system

Participants

- Serena (ex-Pomdapi), Inria Paris
- LAGA, Université Paris Nord, CNRS
- HiePACS, Inria Bordeaux Sud–Ouest
- Andra
- Maison de la Simulation (CEA, CNRS, Inria, UVSQP, UPS)

Skills in analyzing DD methods, porous media flows, parallel linear algebra, high performance computing, storage simulation

Outline

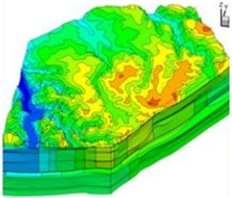
One phase liquid phase

Flow equations

$$q = -\frac{K}{\mu}(\nabla p - \rho g) \qquad \text{Darcy's law}$$

$$\nabla \cdot q = 0 \qquad \text{mass conservation}$$

- p pressure [kg/m/s^2]
- K permeability tensor [m^2] (heterogeneous, anisotropic)
- ρ density [kg/m^3]



- q Darcy velocity [m/s]
- μ dynamic viscosity [kg/m/s]
- g gravity [m/s^2]

Advection–diffusion equation

$$\omega \partial_t c - \underbrace{\text{div}(\mathbf{D}\,\text{grad}\, c)}_{\text{dispersion}} + \underbrace{\text{div}(\mathbf{q}c)}_{\text{advection}} = f$$

- c : concentration [mol/m^3]
- ω : porosity [–]
- **D** diffusion – dispersion tensor [m^2/s] (can be anisotropic)
- **q** Darcy velocity [m/s]

Two–phase immiscible flow

$$\partial_t(\omega \rho_\alpha S_\alpha) + \operatorname{div}(\rho_\alpha u_\alpha) = q_\alpha$$

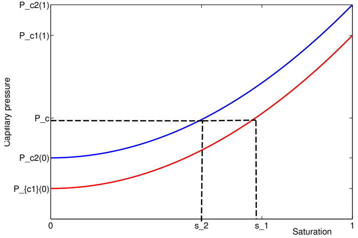
$$u_\alpha = -\frac{k_{r\alpha}(S_\alpha)}{\mu_\alpha} K(\nabla p_\alpha - \rho_\alpha g)$$

$$S_n + S_w = 1$$

$$p_n - p_w = P_c(S_w)$$

Phase $\alpha = w$ water, n gas.
$P_c(S_w)$ increasing function on $[0,1]$

Specific difficulty : discontinuous capillary pressure



- ω porosity [-]
- S_α phase saturation [-]
- u_α phase velocity [m/s]
- $k_{r\alpha}$ relative permeability [-]
- K permeability [m^2]
- p_α : phase pressure [$kg/m/s^2$]
- ρ_α phase density [kg/m^3]
- μ_α viscosity [kg/m/s]

Outline

Flow and transport simulations at Andra

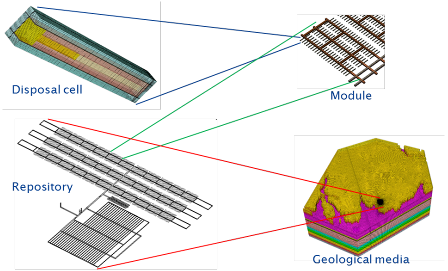
Widely varying scales

- From meter (canister) to (10s of) kilometers (geologic basin)
- Half lives : From years (Tritium 12 years) to millions of years (Iodine 16 000 000 years)
- Permeability varies over 7 orders of magnitude
- Peclet varies from 0.01 (diffusion dominated) to 1000 (advection dominated)

Traces software

Transport RéActif de Contaminant dans les Eaux Souterraines

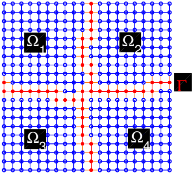
- Saturated and unsaturated transport, reactive transport module
- Mixed-hybrid finite element, discontinuous Galerkin
- Linear solvers : NSPCG, Hypre, MaPHyS

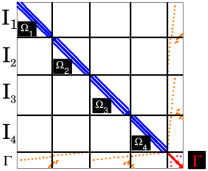


MaPHyS : a hybrid linear solver

Robust scalable parallel hybrid direct/iterative linear solvers

- Developed by Hiepacs teams, Inria Bordeaux
- Exploit the efficiency and robustness of the sparse direct solvers
- Take advantage of the natural scalable parallel implementation of iterative solvers
- Extend domain decomposition ideas to algebraic setting
- Partition the problem into subdomains, subgraphs
- Use a direct solver on the subdomains
- Robust preconditioned iterative solver





Algebraic view

Decomposition $\mathscr{A} = \begin{pmatrix} \mathscr{A}_{II} & \mathscr{A}_{I\Gamma} \\ \mathscr{A}_{\Gamma I} & \mathscr{A}_{\Gamma\Gamma} \end{pmatrix}$

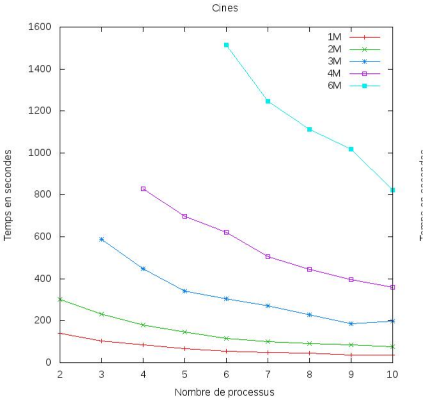
Schur compleemnt
$\mathscr{S} = \mathscr{A}_{\Gamma\Gamma} - \mathscr{A}_{\Gamma I}\mathscr{A}_{II}^{-1}\mathscr{A}_{I\Gamma}$

Improvements to Traces due to MaPHyS

- Consolidation and validation of parallel version of TRACES
- Work to improve time synchronization between subdomains
- Validation by comparison with one processor's simulations

- Test case from 1 to 6 million elements (up to 18M DOFs)
- Flow test case : 1 subdomain per node
- Preliminary results promising, extend to larger (30 M cells), more complex test cases, optimize code usage

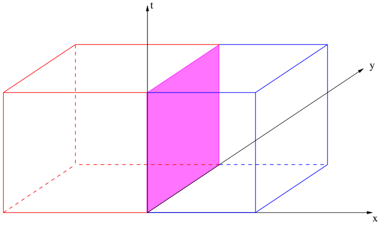
Stage M2 R. Ziara



Outline

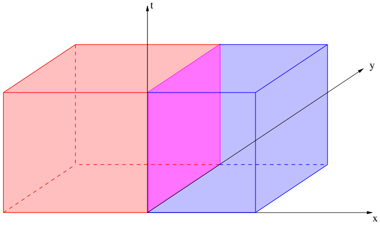
Space–time domain decomposition

Space-time domain decomposition



Space–time domain decomposition

Space-time domain decomposition

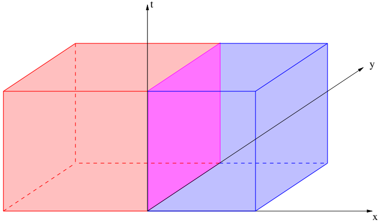


- Solve time-dependent problems in the subdomains
- Exchange information through the space-time interface
- Enable local discretizations both in space and in time

  $\longrightarrow$ local time stepping

Space–time domain decomposition

Space-time domain decomposition



- Solve time-dependent problems in the subdomains
- Exchange information through the space-time interface
- Enable local discretizations both in space and in time

  $\longrightarrow$ local time stepping

Simplified non-linear degenerate diffusion model

$$\omega \partial_t S - \Delta \phi(S) = 0 \quad \text{in } \Omega \times [0, T]$$

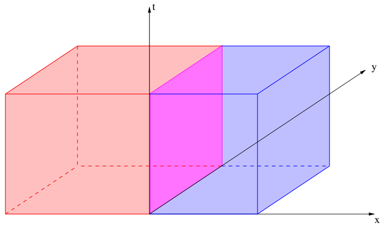
Natural transmission conditions

Continuity of capillary pressure $P_{c1}(S_1) = P_{c2}(S_2)$ on Γ

Continuity of the flux $\nabla \phi_1(S_1).n_1 = -\nabla \phi_2(S_2).n_2$ on Γ

Space–time domain decomposition

Space-time domain decomposition



- Solve time-dependent problems in the subdomains
- Exchange information through the space-time interface
- Enable local discretizations both in space and in time

  $\longrightarrow$ local time stepping

Simplified non-linear degenerate diffusion model

$$\omega \partial_t S - \Delta \phi(S) = 0 \quad \text{in } \Omega \times [0, T]$$

Equivalent transmission conditions

- $\nabla \phi_1(S_1).n_1 + \beta_1 P_{c1}(S_1) = -\nabla \phi_2(S_2).n_2 + \beta_1 P_{c2}(S_2)$
- $\nabla \phi_2(S_2).n_2 + \beta_2 P_{c2}(S_2) = -\nabla \phi_1(S_1).n_1 + \beta_2 P_{c1}(S_1)$

Non-linear Schwarz algorithm

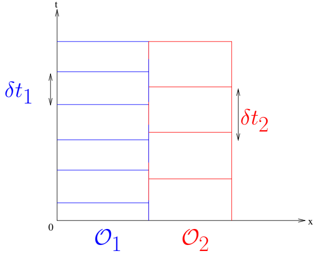
Schwarz algorithm

Given S_i^0, iterate for $k = 0, \dots$
Solve for S_i^{k+1}, $i = 1, 2, j = 3 - i$

$$\omega \partial_t S_i^{k+1} - \Delta \phi_i(S_i^{k+1}) = 0 \quad \text{in } \Omega_i \times [0, T]$$
$$\nabla \phi_i(S_i^{k+1}).n_i + \beta_i P_{c\,i}(S_i^{k+1}) = -\nabla \phi_j(S_j^k).n_j + \beta_i P_{c\,j}(S_j^k) \quad \text{on } \Gamma \times [0, T],$$

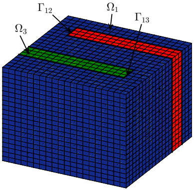
(β_1, β_2) are free parameters chosen to accelerate convergence

- Basic ingredient : subdomain solver with Robin bc.
- Discretization : extension to Robin bc of cell centered FV scheme by Enchéry, Eymard, Michel (2006).
- Different time steps in the subdomains
- Implemented with Matlab Reservoir Simulation Toolbox (Lie et al. (14))

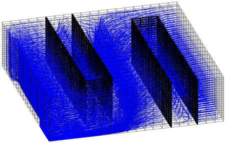


E. Ahmed, C. Japhet, M. Kern (in preparation)

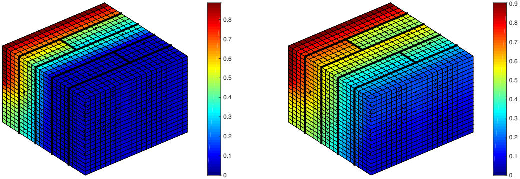
Example : a model with three rock type



Geometry



Streamlines



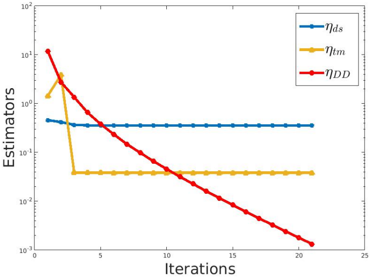
Saturation t=5000, t=2000

Stopping criteria through a posteriroi error estimates (Cemracs 2016)

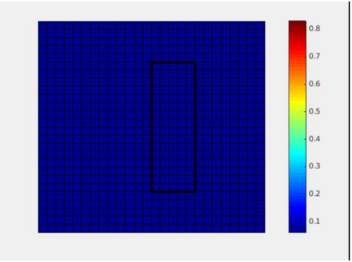
Goal

Stop DD iterations as soon as discretization error is reached

Develop fully computable error estimator with guaranteed bound (no implicit constant), based on potential and flux reconstruction.
Allows separation of space, time, and iteration errors (S. Ali Hassan's PhD, M. Vohralík, C. Japhet)



Example : lens with 2 rock types



Outline

Motivation : Coarse Correction for MaPHyS

Need for Coarse Correction

- Good scalability of the direct part ☺
- The size and condition number of the iterative problem increases with the number of subdomains ☹

A proved robust coarse space for a larger class of methods

- Generalized Abstract Schwarz (GAS) methods
  - Neumann-Neumann, Additive Schwarz, Additive Schwarz on the Schur (MaPHyS), ...
- Only works in the SPD case, with distributed input

Two implementations

- Python prototype, providing a framework for distributed GAS methods
- (partially) integrated in MaPHyS 0.9.4

Two level precondtioner : aS   Step by step

Step 1 : Domain Decomposition

- $\mathscr{A} = \sum_{i=1}^{N} \mathbf{R}_i^T \mathscr{A}_i \mathbf{R}_i$

Step 2 : Factorization

- Computation of $\mathscr{A}_{\mathscr{I}_i\mathscr{I}_i}^{-1}$ and $\mathscr{S}_i = \mathscr{A}_{\Gamma_i\Gamma_i} - \mathscr{A}_{\Gamma_i\mathscr{I}_i}\mathscr{A}_{\mathscr{I}_i\mathscr{I}_i}^{-1}\mathscr{A}_{\mathscr{I}_i\Gamma_i}$

Step 3 : Preconditioner Setup

- $\mathscr{M}_{aS} \quad = \quad \sum_{i=1}^{N} \mathbf{R}_{\Gamma_i}^T \left(\mathbf{R}_{\Gamma_i}\mathscr{S}\mathbf{R}_{\Gamma_i}^T\right)^{-1} \mathbf{R}_{\Gamma_i}$

Step 4 : Solve

- on Γ : *Krylov method* $\quad \mathscr{S} x_\Gamma = f \quad$ preconditioned with $\mathscr{M}_{aS}$
- on $\mathscr{I}$: *Direct method* $\quad x_{\mathscr{I}_i} = \mathscr{A}_{\mathscr{I}_i\mathscr{I}_i}^{-1}\left(b_{\mathscr{I}_i} - \mathscr{A}_{\mathscr{I}_i\Gamma_i} x_{\Gamma_i}\right)$

Two level precondtioner : aS, 2 Step by step

Step 1 : Domain Decomposition (Application level)

- $\mathscr{A} = \sum_{i=1}^{N} \mathbf{R}_i^T \mathscr{A}_i \mathbf{R}_i$

Step 2 : Factorization

- Computation of $\mathscr{A}_{\mathscr{I}_i\mathscr{I}_i}^{-1}$ and $\mathscr{S}_i = \mathscr{A}_{\Gamma_i\Gamma_i} - \mathscr{A}_{\Gamma_i\mathscr{I}_i}\mathscr{A}_{\mathscr{I}_i\mathscr{I}_i}^{-1}\mathscr{A}_{\mathscr{I}_i\Gamma_i}$

Step 3 : Preconditioner Setup

- $\mathscr{M}_{aS,2} = \mathscr{M}_0 + \sum_{i=1}^{N} \mathbf{R}_{\Gamma_i}^T \left(\mathbf{R}_{\Gamma_i}\mathscr{S}\mathbf{R}_{\Gamma_i}^T\right)^{-1} \mathbf{R}_{\Gamma_i}$

Step 4 : Solve

- on Γ : *Krylov method* $\mathscr{S} x_\Gamma = f$ preconditioned with $\mathscr{M}_{aS,2}$
- on $\mathscr{I}$: *Direct method* $x_{\mathscr{I}_i} = \mathscr{A}_{\mathscr{I}_i\mathscr{I}_i}^{-1}\left(b_{\mathscr{I}_i} - \mathscr{A}_{\mathscr{I}_i\Gamma_i} x_{\Gamma_i}\right)$

Coarse space for GAS

2-level method needed to keep the number of iterations independent of # cores.

Two-level abstract Schwarz

Coarse space V_0

Coarse solve $\mathscr{M}_0 = V_0 (V_0^T \mathscr{S} V_0)^\dagger V_0^T$

Proj. onto coarse space $\mathscr{P}_0 = \mathscr{M}_0 \mathscr{S}$

Two-level AS : $\mathscr{M}_D = \mathscr{M}_0 + (I - \mathscr{P}_0)\mathscr{M}_1(I - \mathscr{P}_0)$, $\mathscr{M}_1$ 1 level preconditioner
Generalized AS : replace $\mathscr{S}$ by approximation $\hat{\mathscr{S}}$.

Extend GENEO (Spillane at al.) to GAS

1. Solve locally generalized eigenvalue problems for λ and η above threshold α and β

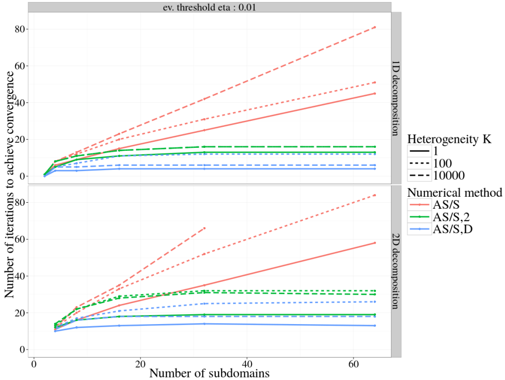
$$\hat{\mathscr{S}}_i p = \lambda \tilde{\mathscr{S}}_i^{\mathrm{NN}} p \quad \text{and} \quad \tilde{\mathscr{S}}_i^{\mathrm{AS}} p = \eta \hat{\mathscr{S}}_i p$$

2. Assemble resulting coarse space : $V_0 = \sum_{i=1}^N \mathscr{R}_\Gamma^T V_i^0$

Condition number bounded independently of N and coefficients

3D Test problem : heterogeneous diffusion

- Alternating conductivity layers of 3 elements (ratio K between layers)
- Python / MPI implementation



Ph D Thesis L. Poirel, in progress

E. Agullo, L. Giraud, L Poirel *Robust coarse spaces for Abstract Schwarz preconditioners via generalized eigenproblems*, hal-01399203, Nov. 2016

The number of iterations is stabilized independently of K and N

Outline

Sparse direct solvers

Problem : solve $Ax = b$

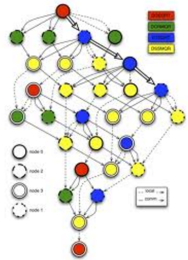
- Cholesky : factorize $A = LL^T$ (symmetric pattern $(A + A^T)$ for LU)
- Solve $Ly = b$, and $L^T x = y$

Sparse Direct Solvers : PaStiX approach

- Inria HiePACS team
- Supernodal method, no pivoting
- Order unknowns to minimize the fill-in
- Compute a symbolic factorization to build L structure
- Factorize the matrix in place on L structure
- Solve the system with forward and backward triangular solves

Advantages of using a task-based runtime system

- Several computing kernels can be associated with the task (C, OpenCL, NVIDIA CUDA)
- Execute the task graph on the available resources
- Address the whole computing units and the whole potential parallelisms
- Insulate the algorithm from the architecture and data distribution
- Automatic handling of data transfers
- Finer parallelism handling

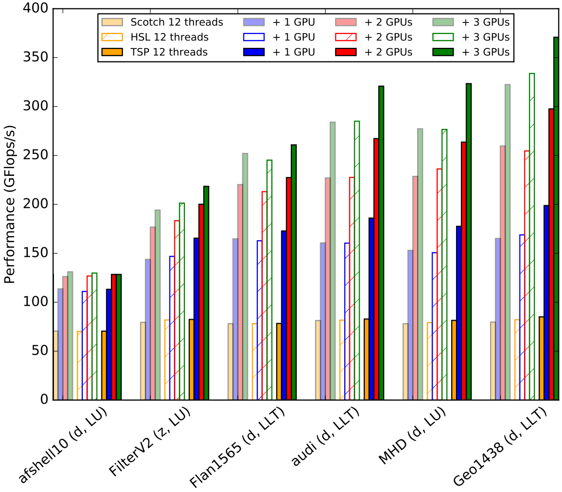


Tasks in Parsec

PaRSEC

- ICL – University of Tennessee, Knoxville
- **Parameterized Task Graph**
- Multiple kernels on the accelerators
- Scheduling strategy based on static performance model
- GPU multi-stream enabled

Performance on Fermi GPU architecture, various test matrices



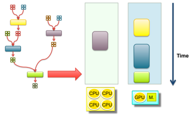
- ≈ 100GFlops speedup per GPU

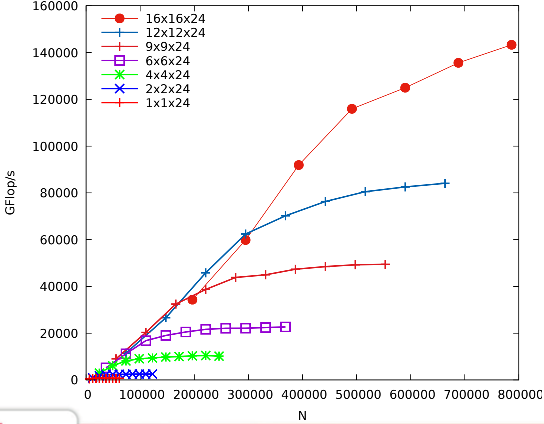
Runs on Curie/Occigen with the Chameleon library

Sequential Task Flow (STF) design of *dense linear algebra* tiles algorithms (derived from PLASMA) on top of runtime systems

Runtime systems

- **QUARK**
- **StarPU**
- PaRSEC
- OmpSS





DPOTRF performance on Occigen (up to 6 000 cores)

Outline

Vectorizing the assembly in finite element computations

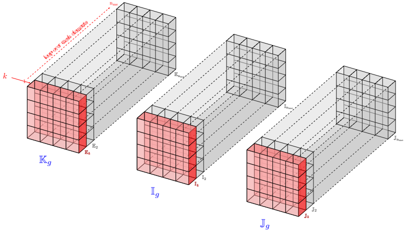
Use of the sparse function of the vector language (triplet)

```
M <-- sparse(Ig, Jg, Kg, n, nq)
```

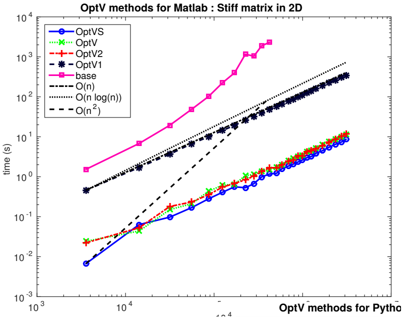
OptFEMP1

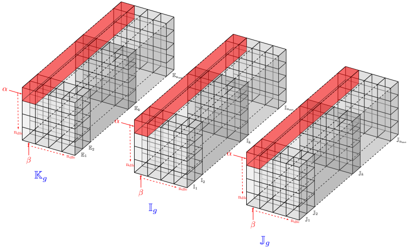
- Optimized assembly of given matrices in vector languages with a P 1-Lagrange finite element method
- Works for interpreted/vector languages (Matlab/Octave and Python)
- Multidimensional (2D, 3D, ...) codes.
- Used in python version of MaPHyS prototype, 2-level DD solver (in progress)

F. Cuvelier, C. Japhet, G. Scarella, *An efficient way to assemble finite element matrices in vector languages*, Bit Numer Math (2016) 56 : 833. doi :10.1007/s10543-015-0587-4.

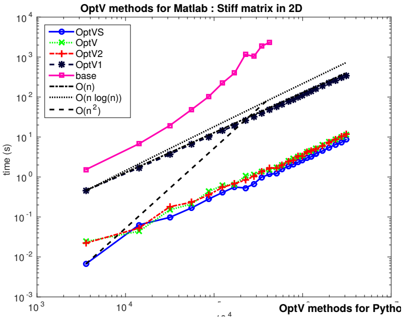


(usual) assembly : loop over mesh elements





Vectorized assembly : loop over local degrees of freedom



Conclusion – Perspectives

- Progress in solver integration in production code
- Space–time geometric DD for non-linear model
- Robust coarse space for algebraic DD
- Direct solver over runtime system, high efficicency
- Efficient building block for finite element in vector languages

Conclusion – Perspectives

- Progress in solver integration in production code
- Space–time geometric DD for non-linear model
- Robust coarse space for algebraic DD
- Direct solver over runtime system, high efficicency
- Efficient building block for finite element in vector languages

Perspectives for last year

- Domain decomposition with 2-level parallelism
  - Implement (geometric) DD approach with a parallel subdomain solver
  - Code for subdomain solver : Compass (Univ. Nice, BRGM, ANR Charms)
  - Add python wrapper for outer Schwarz iterations
- Validate developments on realistic test cases