Introduction au développement dans l'environnement de programmation PETSc

Marc MEDALE

École Polytechnique Universitaire de Marseille
Département de Mécanique Énergétique
et
IUSTI, UMR 6595 CNRS - Université Aix-Marseille

e-mail : Marc.Medale@polytech.univ-mrs.fr

Plan de la présentation

- Analyse préalable au développement
- Choix stratégiques de développement ;
- Principaux concepts de PETSc
- Implémentation dans PETSc ;
- Quelques conclusions.

Analyse préalable au développement

- Dispose-t-on d'un code de calcul et d'un ordinateur susceptible de l'exécuter convenablement ?
  - Ressources informatiques requises (CPUs, RAM) ;
  - Extensibilité en calcul parallèle (scalability) ;
  - Prix du calcul (compatibilité modèles-algorithmes-ressources) ;
- Intérêts du 'High Performance Computing' :
  - à taille donnée, résoudre plus vite ;
  - à durée donnée, résoudre un plus gros problème ;
  - à taille et durée données, résoudre avec moins de ressources (moins cher).

Environnement de calcul H.P.C.

- Les infrastructures sont là (IDRIS, CINES, CCRT, méso-grilles, etc.)
- Les ordinateurs de calcul scientifique sont remplacés environ tous les 5 ans :
- Les codes de calcul HPC ont une durée de vie limitée :
  - les paradigmes de programmation ;
  - les architectures matérielles (multi-coeur, accès mémoire, etc.)
- La portabilité devient une priorité, l'optimisation poussée un luxe
- Il existe des bibliothèques gratuites « open-sources », portables, performantes, développées et maintenues …

Dans quel environnement développer ?

Quelques questions clé à se poser avant de se lancer :

- Identification de la communauté concernée ;
  - Nombre de personnes impliquées dans le développement et l'utilisation, background informatique et numérique ;
  - Moyens de maintenance du code (subversion, etc.) et support aux utilisateurs (documentation automatique) ;
  - Proximité de domaines physiques et méthodes numériques pressenties ;
- Nature de l'espace collaboratif ?
  - Plateforme générique de développement ;
  - Librairies scientifiques ;
  - Code spécialisé.

Quelques critères de choix d'une librairie de calcul scientifique

- Open source, gratuite
- Portable
- Performante
- Nombreuses fonctionnalités
- Ouverte : interfacable vers d'autres bibliothèques
- Accessible : facile à comprendre et à utiliser (documentation, exemples, templates, etc.)
- Pérenne : maintenue et développée sur le moyen terme (5 à 10 ans)

Choix stratégiques dans le contexte du H.P.C.

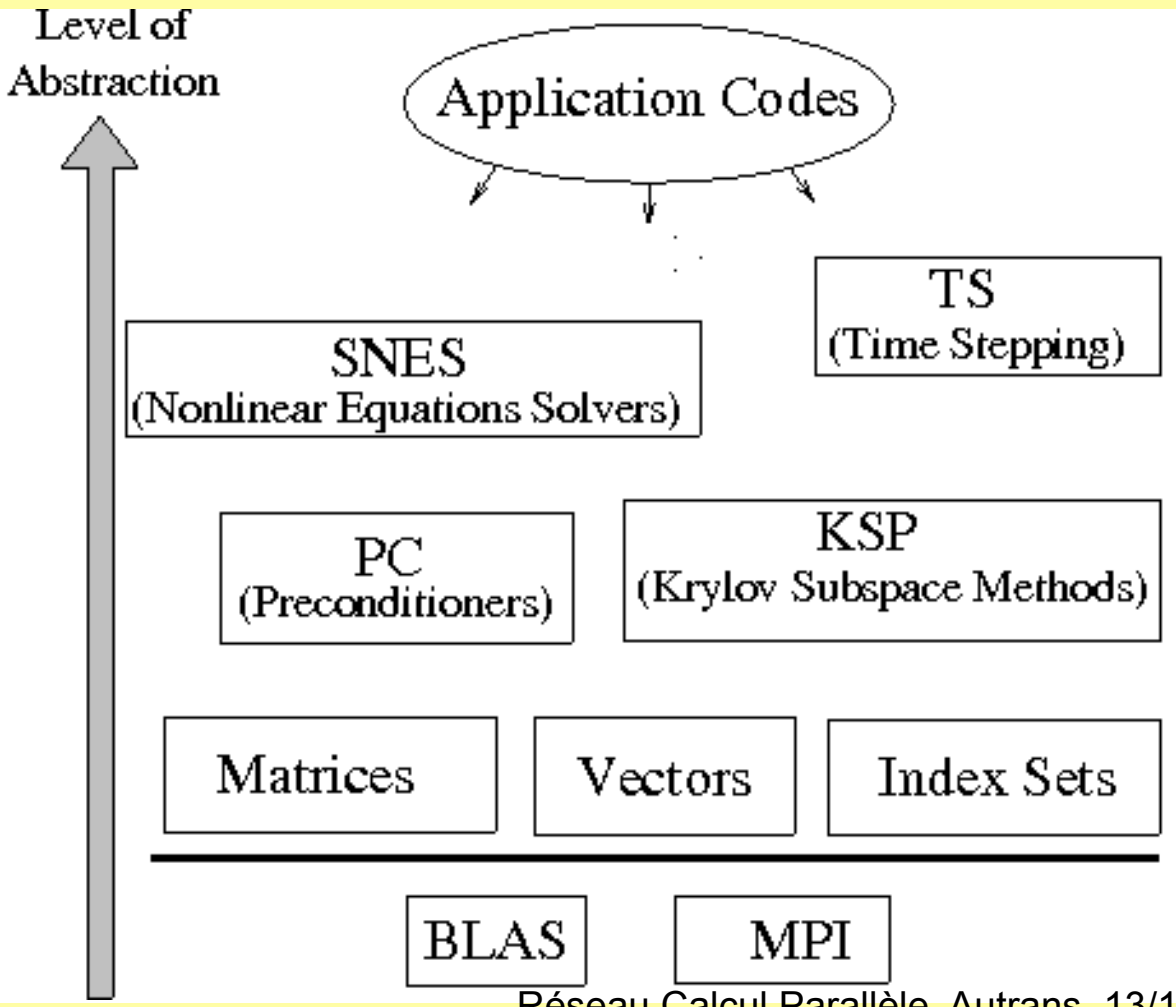
- Analyse fonctionnelle du code ;
  - **Dévelop. centrés sur les spécificités de nos modèles ;**
  - **Sous-traitance des parties génériques (PETSc, BLAS, LAPACK, MUMPS, MPI, etc.) ;**
- Développement dans un environnement de programmation orienté objets ;
- Adéquation modèles - algorithmes - plates-formes ;
- Calculateurs parallèles à hautes performances ;
  - Super-calculateurs GENCI (IDRIS, CINES, CCRT)
  - En local (meso-grille, clusters).

PETSc (http://www.mcs.anl.gov/petsc)

Portable Extensible Toolkit Scientific computations

(Argone National Laboratory)

- Download
- Features
  - Component Details
  - Diagram
  - GPUs
  - Threads
- Documentation
- Applications/Publications
- Miscellaneous
- External Software
- Developers Site



Les principaux objects de PETSc

PETSc components provide the functionality required for many parallel solutions of PDEs.

`Vec`

Provides the vector operations required for setting up and solving large-scale linear and nonlinear problems. Includes easy-to-use parallel scatter and gather operations, as well as special-purpose code for handling ghost points for regular data structures.

`Mat`

A large suite of data structures and code for the manipulation of parallel sparse matrices. Includes four different parallel matrix data structures, each appropriate for a different class of problems.

`PC`

A collection of sequential and parallel preconditioners, including (sequential) ILU(k), LU, and (both sequential and parallel) block Jacobi, overlapping additive Schwarz methods and structured MG using DMMG.

`KSP`

Parallel implementations of many popular Krylov subspace iterative methods, including GMRES, CG, CGS, Bi-CG-Stab, two variants of TFQMR, CR, and LSQR. All are coded so that they are immediately usable with any preconditioners and any matrix data structures, including matrix-free methods.

`SNES`

Data-structure-neutral implementations of Newton-like methods for nonlinear systems. Includes both line search and trust region techniques with a single interface. Employs by default the above data structures and linear solvers. Users can set custom monitoring routines, convergence criteria, etc.

`TS`

Code for the time evolution of solutions of PDEs. In addition, provides pseudo-transient continuation techniques for computing steady-state solutions.

Les principaux objects de PETSc

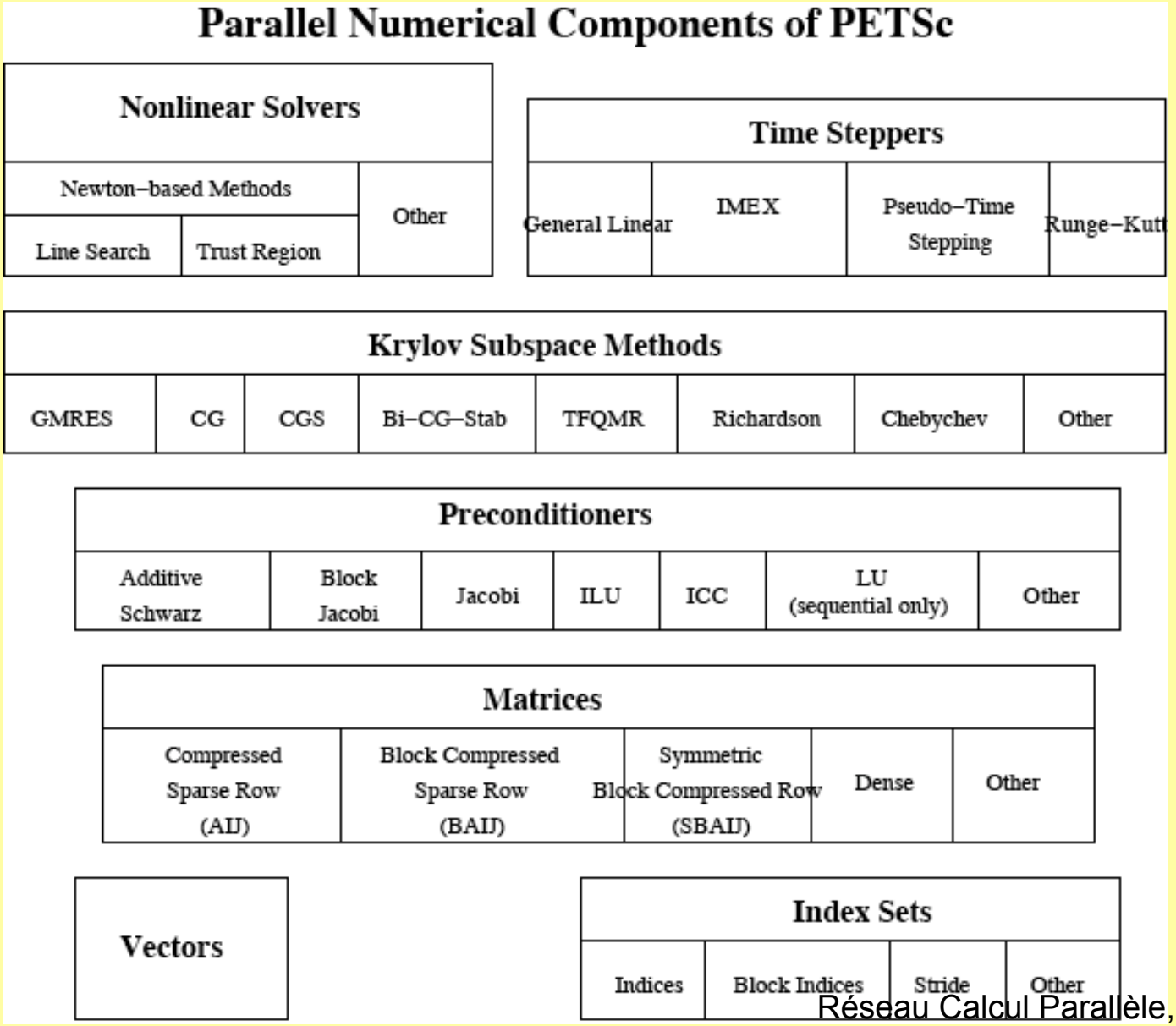
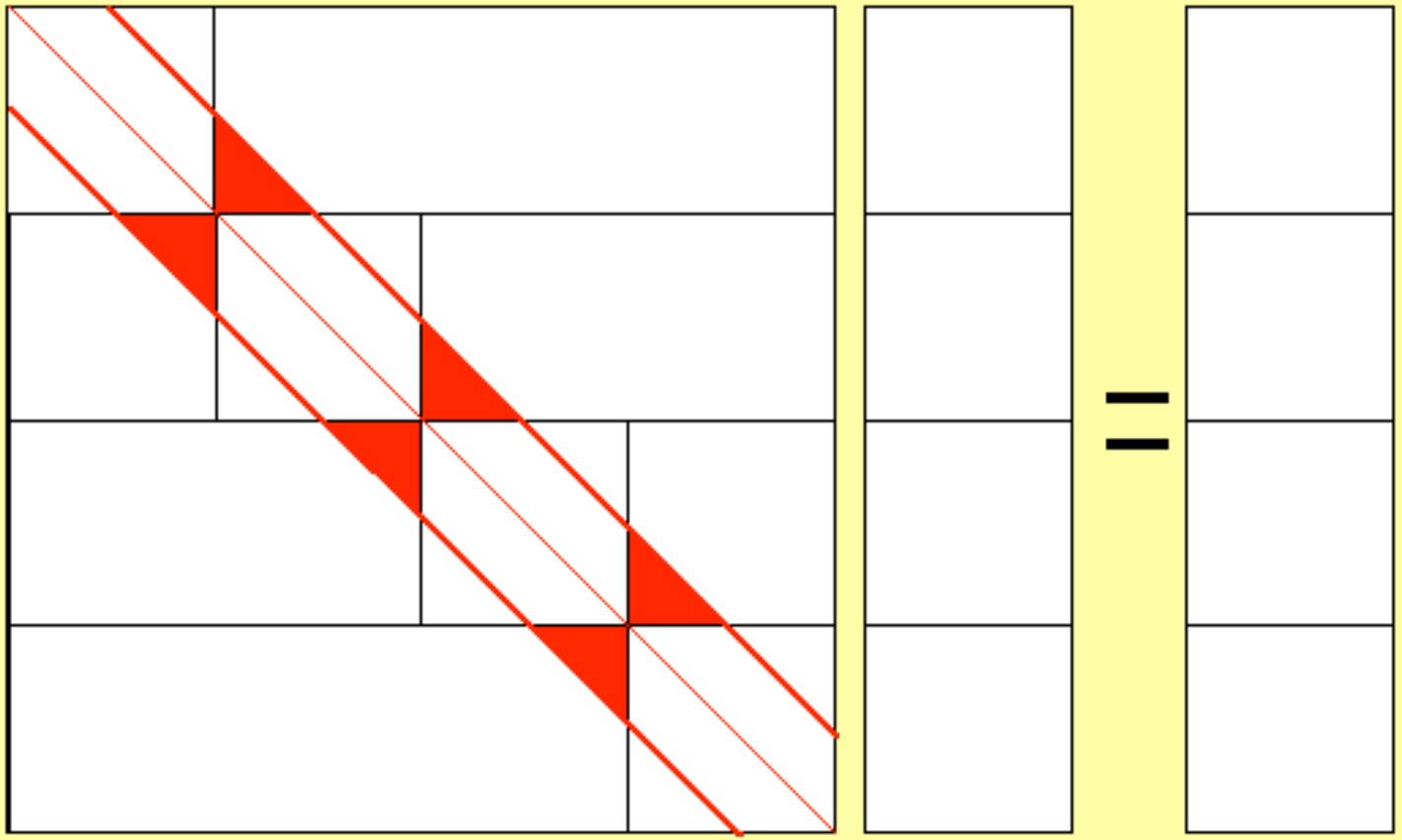


Figure3: NumericalLibraries ofPETSc

Principes de résolution // syst. algébriq.

Solver itératif => nombreux produits matrice-vecteur



Opérations de base sur des vecteurs

Function Name	Operation
VecAXPY (Vec y,PetscScalar a,Vec x);	$y = y + a\ x$
VecAYPX (Vec y,PetscScalar a,Vec x);	$y = x + a\ y$
VecWAXPY (Vec w,PetscScalar a,Vec x, Vec y);	$w = a\ x + y$
VecAXPBY (Vec y,PetscScalar a,PetscScalar b,Vec x);	$y = a\ x + b\ y$
VecScale(Vec x, PetscScalar a);	$x = a\ x$
VecDot(Vec x, Vec y, PetscScalar *r);	$r = \bar{x}^0\ y$
VecTDot(Vec x, Vec y, PetscScalar *r);	$r = x^0\ y$
VecNorm(Vec x,NormType type, PetscReal *r);	$r = \|\|x\|\|_{type}$
VecSum(Vec x, PetscScalar *r);	$r = \sum_i x_i$
VecCopy(Vec x, Vec y);	$y = x$
VecSwap(Vec x, Vec y);	$y = x$ while $x = y$
VecPointwiseMult(Vec w,Vec x, Vec y);	$w_i = x_i\ y_i$
VecPointwiseDivide(Vec w,Vec x,Vec y);	$w_i = x_i / y_i$
VecMDot(Vec x,intn,Vec y[], PetscScalar *r);	$r[i] = \bar{x}^0\ y[i]$
VecMTDot(Vec x,intn,Vec y[], PetscScalar *r);	$r[i] = x^0\ y[i]$
VecMAXPY (Vec y,intn, PetscScalar *a, Vec x[]);	$y = y + \sum_i a_i\ x[i]$
VecMax(Vec x,int*idx, PetscReal *r);	$r = \max x_i$
VecMin(Vec x,int*idx, PetscReal *r);	$r = \min x_i$
VecAbs(Vec x);	$x_i = \|x_i\|$
VecReciprocal(Vec x);	$x_i = 1/x_i$
VecShift(Vec x, PetscScalar s);	$x_i = s + x_i$
VecSet(Vec x,PetscScalar alpha);	$x_i = \alpha$

Opérations de base sur des matrices

Function Name	Operation
MatAXPY(Mat Y, PetscScalar a, Mat X, MatStructure);	$Y = Y + a X$
MatMult(Mat A, Vec x, Vec y);	$y = A x$
MatMultAdd(Mat A, Vec x, Vec y, Vec z);	$z = y + A x$
MatMultTranspose(Mat A, Vec x, Vec y);	$y = A^T x$
MatMultTransposeAdd(Mat A, Vec x, Vec y, Vec z);	$z = y + A^T x$
MatNorm(Mat A, NormType type, double*r);	$r = \|\|A\|\|_{type}$
MatDiagonalScale(Mat A, Vec l, Vec r);	$A = diag(l) A diag(r)$
MatScale(Mat A, PetscScalar a);	$A = a A$
MatConvert(Mat A, MatType type, Mat *B);	$B = A$
MatCopy(Mat A, Mat B, MatStructure);	$B = A$
MatGetDiagonal(Mat A, Vec x);	$x = diag(A)$
MatTranspose(Mat A, MatReuse, Mat* B);	$B = A^T$
MatZeroEntries(Mat A);	$A = 0$
MatShift(Mat Y, PetscScalar a);	$Y = Y + a I$

Algorithmes itératifs de résolution

Method	KSPType	Options Database Name
Richardson	KSPRICHARDSON	richardson
Chebychev	KSPCHEBYCHEV	chebychev
Conjugate Gradient [12]	KSPCG	cg
BiConjugate Gradient	KSPBICG	bicg
Generalized Minimal Residual [16]	KSPGMRES	gmres
Flexible Generalized Minimal Residual	KSPFGMRES	fgmres
Deflated Generalized Minimal Residual	KSPDGMRES	dgmres
Generalized Conjugate Residual	KSPGCR	gcr
BiCGSTAB [19]	KSPBCGS	bcgs
Conjugate Gradient Squared [18]	KSPCGS	cgs
Transpose-Free Quasi-Minimal Residual (1) [8]	KSPTFQMR	tfqmr
Transpose-Free Quasi-Minimal Residual (2)	KSPTCQMR	tcqmr
Conjugate Residual	KSPCR	cr
Least Squares Method	KSPLSQR	lsqr
Shell for no KSP method	KSPPREONLY	preonly

Techniques de préconditionnement

Method	PCType	Options DatabaseName
Jacobi	PCJACOBI	jacobi
BlockJacobi	PCBJACOBI	bjacobi
SOR(andSSOR)	PCSOR	sor
SORwith Eisenstattrick	PCEISENSTAT	eisenstat
IncompleteCholesky	PCICC	icc
IncompleteLU	PCILU	ilu
AdditiveSchwarz	PCASM	asm
Linearsolver	PCKSP	ksp
Combinationofpreconditioners	PCCOMPOSITE	composite
LU	PCLU	lu
Cholesky	PCCHOLESKY	cholesky
Nopreconditioning	PCNONE	none
Shellforuser-definedPC	PCSHELL	shell

Interfaces vers des solveurs directs

MatType	PCType	MatSolverPackage		Package (-pc_factor_mat_solver_package)
s e q a i j l u		MATSOLVERESSL		essl
s e q a i j l u		MATSOLVERLUSOL		lusol
s e q a i j l u		MATSOLVERMATLAB		matlab
a i j l u		MATSOLVERMUMPS		mumps
a i j c h o l e sky				
s b a i jcholesky				
mpidenselu		MATSOLVERPLAPACK		plapack
mpidensecholesky				
a i j l u		MATSOLVERSPOOLES		spooles
s b a i jcholesky				
s e q a i j l u		MATSOLVERSUPERLU		superlu
a i j l u		MATSOLVERSUPERLU	_DIST	superlu_dist
s e q a i j l u		MATSOLVERUMFPACK		umfpack

Principaux avantages de PETSc

- Portable, bien programmé, largement testé
- Open-sources, développé et maintenu (DOE)
- Très bien documenté, équipe de développement et d'assistance (≈ 10 personnes)
- Performances
  - Superposition des phases de com. et calculs ;
  - Choix du moment opportun pour communiquer ;
  - Optimisation des communications répétées ;
  - Regroupement des données à communiquer avant de les envoyer ;

Principaux avantages de PETSc

- Facilité de programmation (3 niveaux de prog.)
  - Gestion d'objets // de haut niveau (mat., vect., solver, SNES, TS, …)
  - Généricité (Prog. Orientée Objets) ;
- Multiples options à l'exécution
  - particularisation du type de stockage
  - choix de la technique de préconditionnement
  - choix du type de solveur (itératif, direct)
  - monitoring des objets, de la convergence des l'algo, etc.
  - profiling

Code de recherche développé dans PETSc

- Écoulements incompressibles, dilatables et faiblement compressibles, avec transferts thermiques ;
- Équations de Navier-Stokes couplées à l'équation de l'énergie ;
- Approximations de Boussinesq et dilatable (faible nombre de Mach) ;
- Recherche de solutions :
  - Stationnaires (lorsqu'elles existent) ;
  - Instationnaires ;
  - Études de stabilité.

Modèles numériques développés

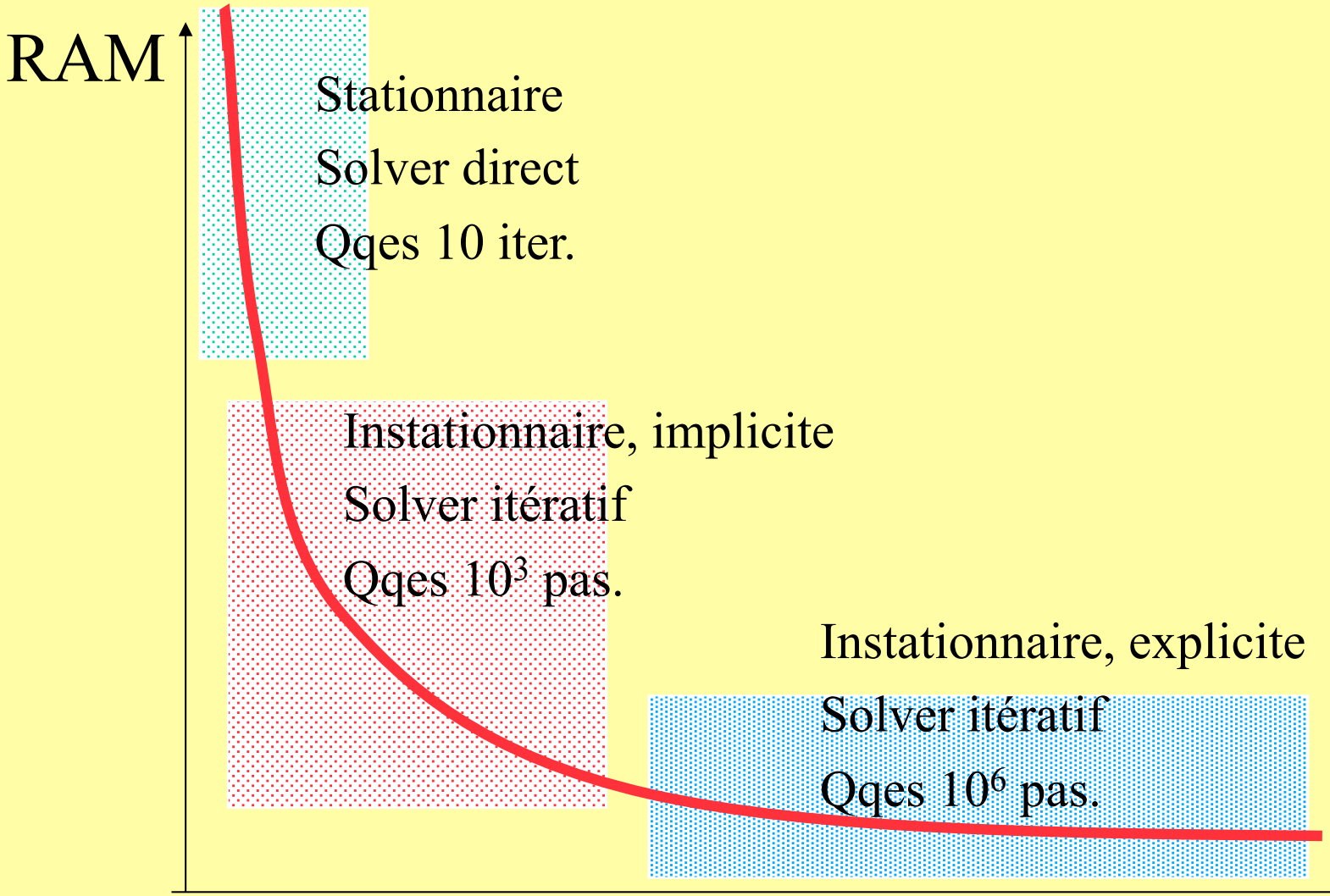
A) Calcul de solutions stationnaires

- Formulation couplée (vites.-pres.-temp.) ;
- Newton-Raphson + ‘cubic line search’ ;
- Solveur direct parallèle LU (MUMPS) ;
- Méthodes de continuation (NR-ALP, MAN).

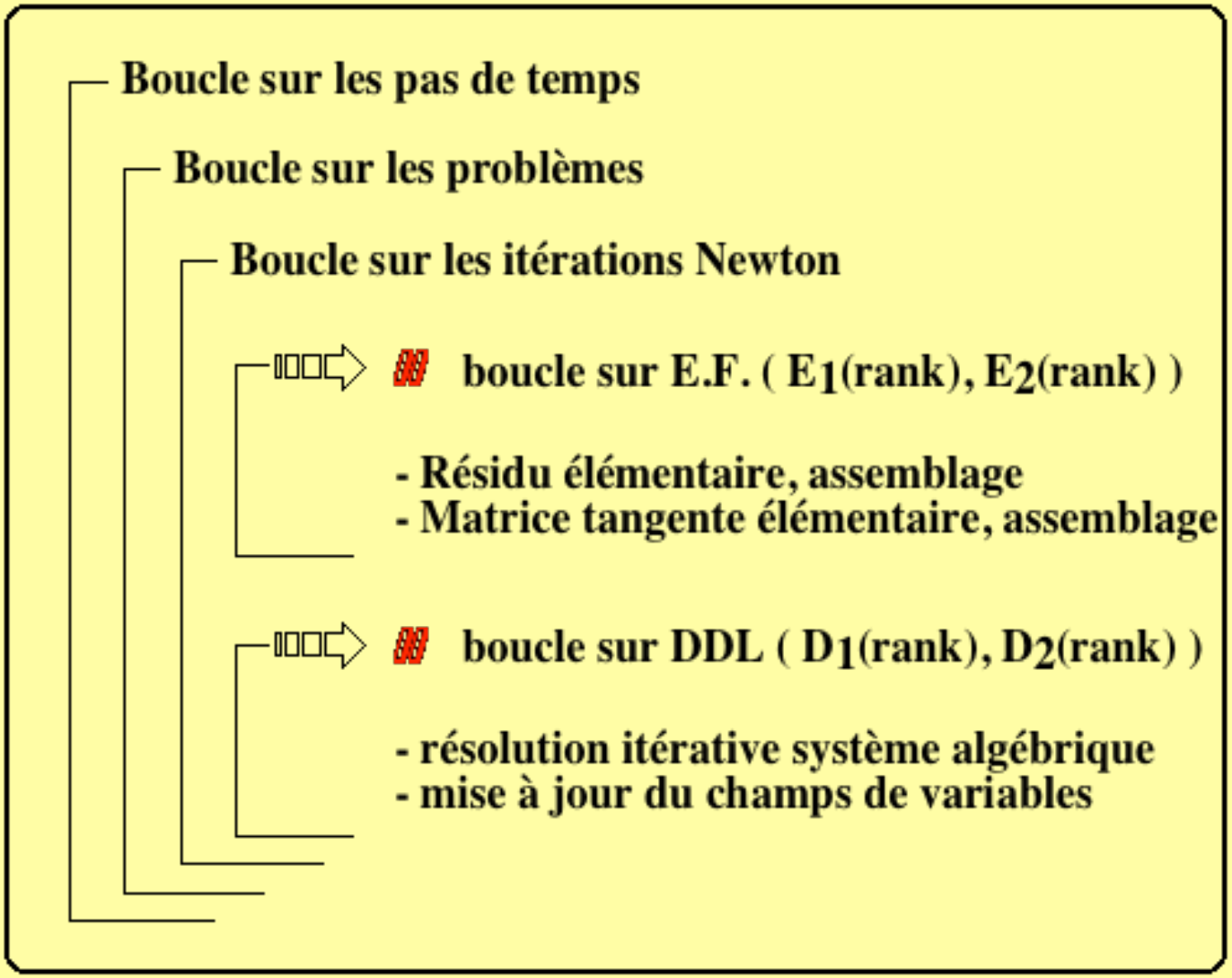
B) Calcul de solutions instationnaires

- Formulation ‘segregated’ (vites.-pres.-temp.) ;
- Méthode de Projection Incrémentale ;
- Newton-Raphson + ‘cubic line search’ ;
- Solveurs itératifs parallèles (BCGS + ASM + ILU) ;
- Schéma d’Euler semi-implicite (BDF2).

Typologie de résolution syst. algébrique



Algorithme du programme principal



Algorithmes itératifs utilisés dans Petsc :

- **système non-linéaire : Newton-Raphson + cubic line search**
- **système linéaire : BCGS + (ASM ou SSOR)**

Formulations éléments finis

Heat transfer

$$\int_{\Omega}\left[\rho C_P\left(\frac{\partial T}{\partial t}+\vec{u}.\vec{\nabla T}\right)\delta T+k\left(\vec{\nabla T}.\vec{\nabla \delta T}\right)\right]dv=-\int_{\Omega}\rho L_F\left(\frac{\partial f}{\partial t}+\vec{u}.\vec{\nabla} f\right)\delta T\,dv+\int_{\partial\Omega}k\frac{\partial T}{\partial n}\delta T\,ds$$

+ Dirichlet type B.C.

Incompressible fluid flow

* Momentum :

$$\int_{\Omega}\left[\rho\left(\frac{\vec{u}^{k+1}-\vec{u}^{k}}{\Delta t}+\left(\vec{u}^{k+1}.\vec{\nabla}\right)\vec{u}^{k+1}\right)\delta\vec{u}+\mu\left(\vec{\nabla}\vec{u}^{k+1}:\vec{\nabla}\delta\vec{u}\right)\right]dv=$$

$$\int_{\Omega}\delta\vec{u}\left[\vec{f}^{k+1}-\vec{\nabla}\left(2\,p^{k}-p^{k-1}\right)\right]dv+\int_{\partial\Omega}\delta\vec{u}\left(\bar{\bar{\sigma}}_{v}^{k+1}.\vec{n}\right)ds$$

+ Dirichlet type B.C.

* Projection step :

$$\int_{\Omega}\vec{\nabla}\left(p^{k+1}-p^{k}\right).\vec{\nabla\delta p}\,dv=-\frac{\rho}{\Delta t}\int_{\Omega}\delta p\left(\vec{\nabla}.\vec{u}^{k+1}\right)dv$$

(+ Dirichlet type B.C.)

Conclusions

- Développement de code dans un contexte H.P.C. et implémentation dans un env. de prog. de haut niveau (PETSc) ;
- Analyse poussée du problème et de l'environnement de programmation ;
- Choix de formulations et algorithmes adaptés ;
- Compatibilité modèles-algorithmes-ressources ;
- Sous-traitance des parties génériques (com., solveurs, etc.) à des librairies spécialisées et H.P.C.
- Pérennité des développements : portabilité, performances, facilité de rajouter des modèles et méthodes ;
- Compromis temps de développement, d'exécution, de maintenance ;

TP N°1 : introduction à l'utilisation

Concepts : mettre en pratique les fonctionnalités au travers d'exemples élémentaires de PETSc

- Préparation : copie de mon répertoire sur votre compte, compilation, etc.

```
cp –r /home/medale/PETSc/ .
cd TP_intro/
make ex2 (ou make ex2f)
```

- Lancer l'éxécution par défault

```
time mpirun -n 2 ./ex2
```

- Affichage des caractéristiques du solveur utilisé, de la convergence, etc.

```
time mpirun -n 2 ./ex2 -n 100 -m 100 -ksp_view
time mpirun -n 2 ./ex2 -n 100 -m 100 -ksp_monitor
time mpirun -n 2 ./ex2 -n 100 -m 100 -ksp_monitor –log_summary
```

- Comparaison des performances des diverses combinaisons (préconditionneur-solveur itératif) pour n=m=100, 1000 et #NB_PROCS.

```
time mpirun -n 2 ./ex2 -n 100 -m 100 -ksp_monitor –ksp_type bcgs –pc_type
   asm –ksp_atol 1e-15 –ksp_rtol 1e-12 –ksp_max_it 1000
```

TP N°2 : solveur non-linéaire

Concepts : mettre en pratique les fonctionnalités au travers d'exemples élémentaires de PETSc

- Préparation : copie de mon répertoire sur votre compte, compilation, etc.

```
cp –r /home/medale/PETSc/ .
cd TP_non_lineaire/
make ex5 (ou make ex5f)
```

- Lancer l'éxécution par défault

```
time mpirun -n 2 ./ex5
```

- Affichage des caractéristiques du solveur utilisé, de la convergence, etc

```
time mpirun -n 2 ./ex5 -snes_view
time mpirun -n 2 ./ex5 -snes_monitor
time mpirun -n 2 ./ex5 -snes_monitor –log_summary
```

- Comparaison des performances des divers paramètres de résolution et #NB_PROCS.

```
time mpirun -n 2 ./ex5 -snes_monitor -snes_max_it 10 -snes_atol 1e-9 -snes_rtol 1e-6 -snes_stol 1e-6 -
    da_grid_x 10 -da_grid_y 10
```