Calcul Parallèle

Zhang Minghe et Diyae-eddine Farid

M2 Analyse modélisation et simulation

Plan

1) Norme 2 et Norme L2 :
    * Code séquentiel
    * Code parallèle
2) Validation du code Parallèle : Convergence de la norme L2 ?
3) Gradient conjugué :
    * Code séquentiel
    * Code parallèle
    * Validation
4) Convergence des méthodes de Jacobi et du gradient conjugué sur un benchmark
5) Étude la scalabilité faible et la scalabilité forte du code parallélisé sur le cluster cholesky
6) Comparaison de la performance parallèle de la méthode de Jacobi et de Gradient conjugué

Norme du résidu (Séquentiel)

```
// Update residual and iterator

if((it % 10) == 0){
  residuNorm = 0;
  Au = A*u;
  exchangeAddInterfMPI(Au, mesh);
  res = b-Au;  //residual vector
  residuNorm = res.dot(res);
  residuNorm = sqrt(residuNorm);  //norm of residual

  if(myRank == 0){
    printf("\r   %i %e", it, residuNorm);
  }
}
```

Norme L2 (Séquentiel)

$$\|v\|^2_{L^2(\Omega)} = \int_\Omega |v|^2 \, d\Omega \approx \mathbf{v}^\top \mathbf{M} \mathbf{v}.$$

```
void normL2(SpMatrix& M,Vector& v,Mesh& mesh)
{
  printf("   -> final L2 error: %e\n",sqrt(v.dot(M*v)));
}
```

Validation du code séquentiel

Dans cette partie on prend $\alpha = 1$ et $u(x, y) = \cos(2\pi x)\cos(3\pi y)$, $h = 0.05$, tol $= 10^{-6}$, maxit $= 10000$

$$f(x, y) = (1 + 13\pi^2)\cos(2\pi x)\cos(3\pi y)$$

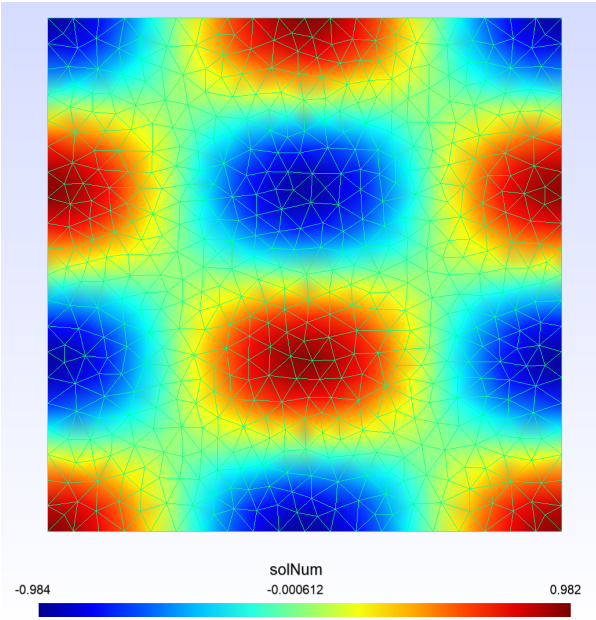


Figure: Solution approchée

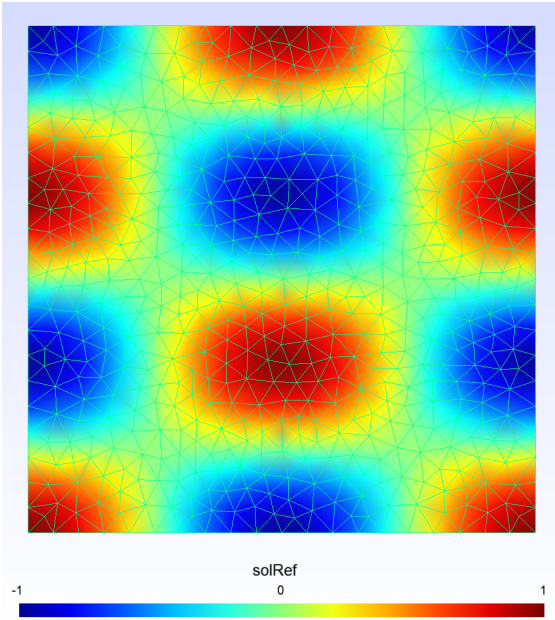


Figure: Solution Exacte

Validation du code séquentiel

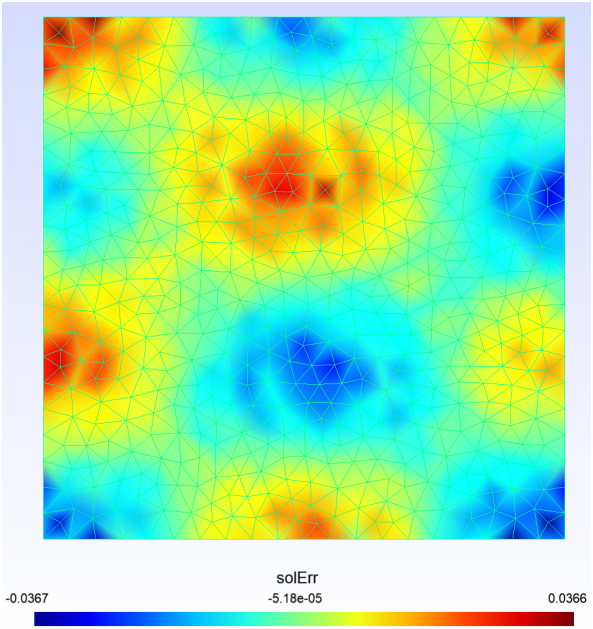


Figure: error

```
== read mesh file
   -> 568 nodes
   -> 1054 triangles
== build local numbering and list of nodes for MPI communications
== build linear system
== jacobi
   -> final iteration: 7721 (prescribed max: 10000)
   -> final residual: 9.981090e-07 (prescribed tol: 1.000000e-06)
   -> final L2 error: 1.041902e-02
```

Gradient Conjugué (séquentiel)

```
while (residuNorm > tol && it < maxit) {
  Vector Ap = A*p ;
  exchangeAddInterfMPI(Ap, mesh);
  alpha = r.dot(p)/Ap.dot(p);
  u = u + alpha*p;
  r = r - alpha*Ap;
  Vector Ar = A*r;
  exchangeAddInterfMPI(Ar, mesh);
  beta = -(Ar.dot(p))/(Ap.dot(p));
  p = r + beta*p;

  // Update residual and iterator
  if((it % 10) == 0){
    residuNorm = 0;
    Au=A*u;
    exchangeAddInterfMPI(Au, mesh);
    res = b-Au;
    residuNorm = res.dot(res);
    residuNorm=sqrt(residuNorm);
    if(myRank == 0){
      printf("\r   %i %e", it, residuNorm);
    }
  }
  it++;
}
```

Gradient Conjugué (séquentiel)

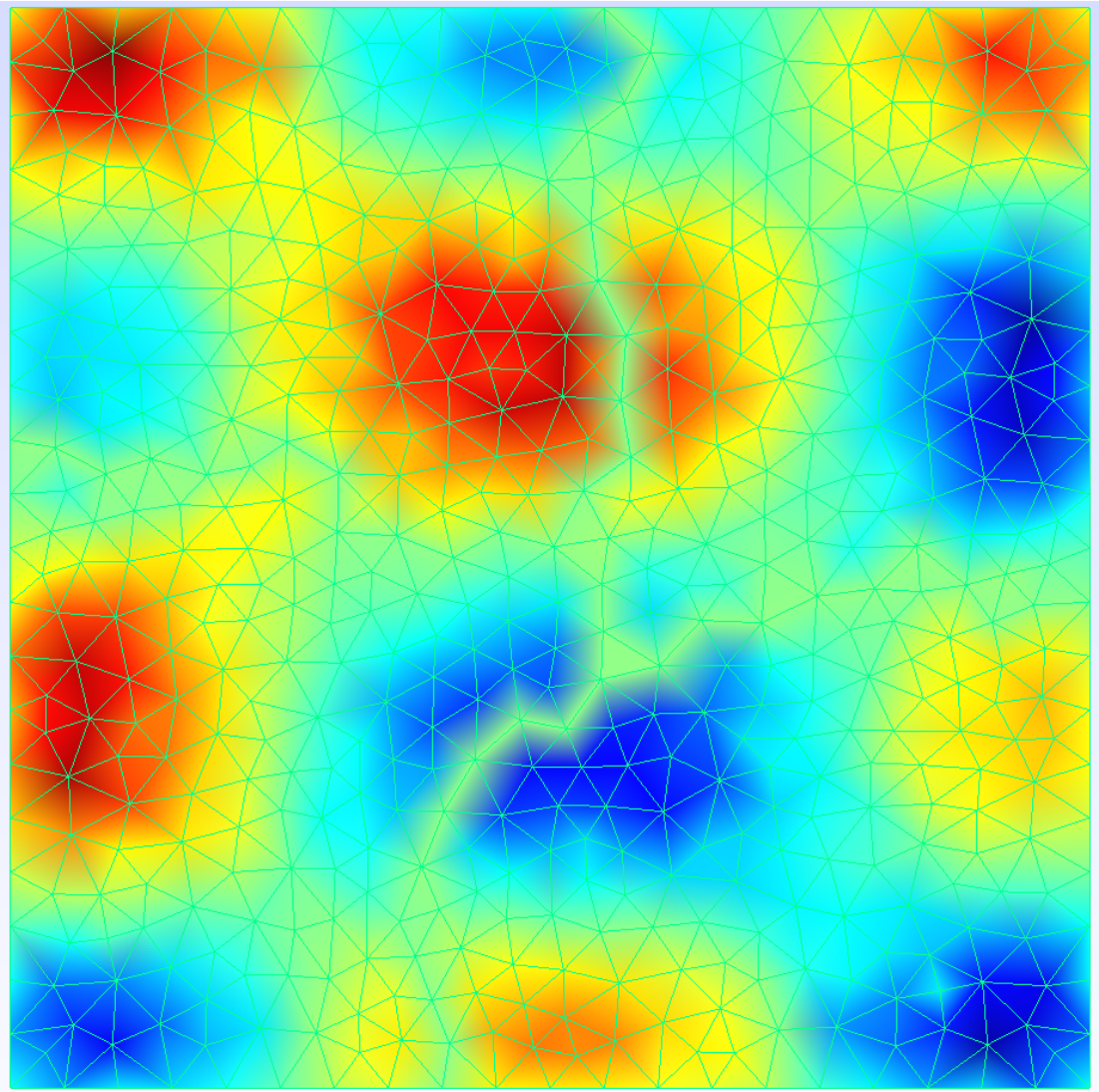
```
== read mesh file
   -> 568 nodes
   -> 1054 triangles
== build local numbering and list of nodes for MPI communications
== build linear system
== Conjugate gradient
   -> final iteration: 111 (prescribed max: 10000)
   -> final residual: 5.838040e-07 (prescribed tol: 1.000000e-06)
   -> final L2 error: 1.041930e-02
```

Parallélisation

Dans le programme, le produit Matrice*vecteur est bien parallélisé, il nous reste la parallélisation du produit scalaire.
Idée:

- Trouver le numéro des noeuds sur l'interface qui est calculé $\geq$ 3 fois (au plus 3 fois dans notre cas)
- Laisser tous les noeuds sur l'interface qui est calculé 2 fois
- Calculer le produit scalaire sur l'interface
- Remplir la valeur 0 sur l'interface
- Calculer le produit scalaire des vecteurs qui ont 0 sur l'interface puis additionner la valeur du produit scalaire sur l'interface qui divise par 2
- Additionner la valeur de chaque processus

Exemple



Norme du résidu parallélisée

```
if((it % 10) == 0){
  residuNorm = 0;
  Au = A*u;
  exchangeAddInterfMPI(Au, mesh);
  res = b-Au;
  double r = para_ps(res,res,mesh);
  MPI_Reduce(&r, &residuNorm, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

  residuNorm = sqrt(residuNorm);

  if(myRank == 0){
    printf("\r   %i %e", it, residuNorm);
  }
}
```

Norme L2 parallélisée

```
void normL2(SpMatrix& M,Vector& v,Mesh& mesh)
{
  double  r = 0;
  Vector Mv = M*v;
  exchangeAddInterfMPI(Mv,mesh);
  double rr = para_ps(v,Mv,mesh);
  MPI_Reduce(&rr, &r, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
  if (myRank == 0){
    printf("    -> final L2 error: %e\n",sqrt(r));
  }
}
```

Validation du code parallèle (Jacobi)

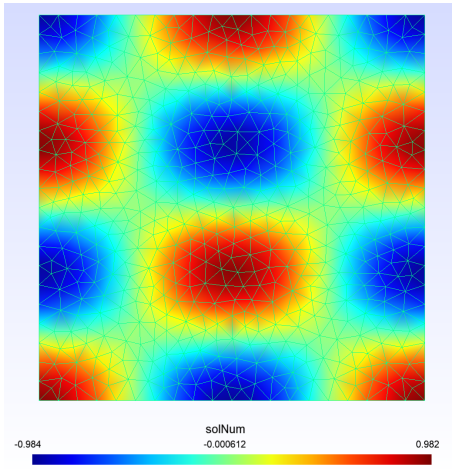


Figure: approchée

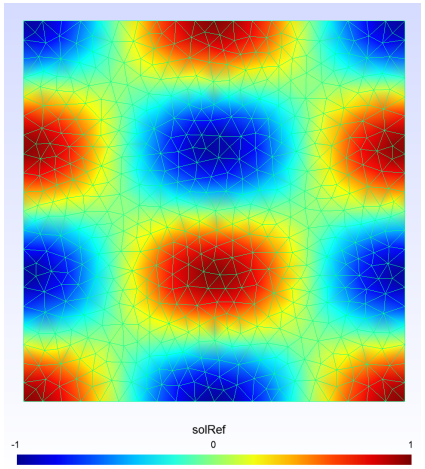


Figure: Exacte

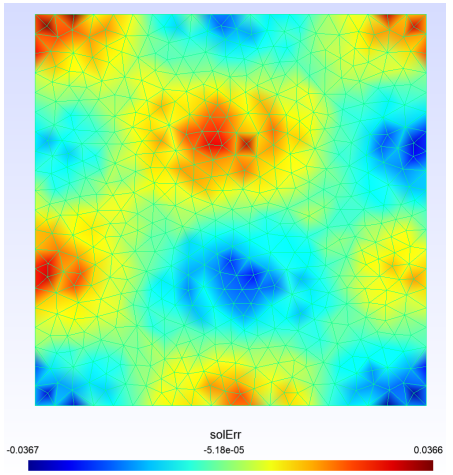


Figure: Error

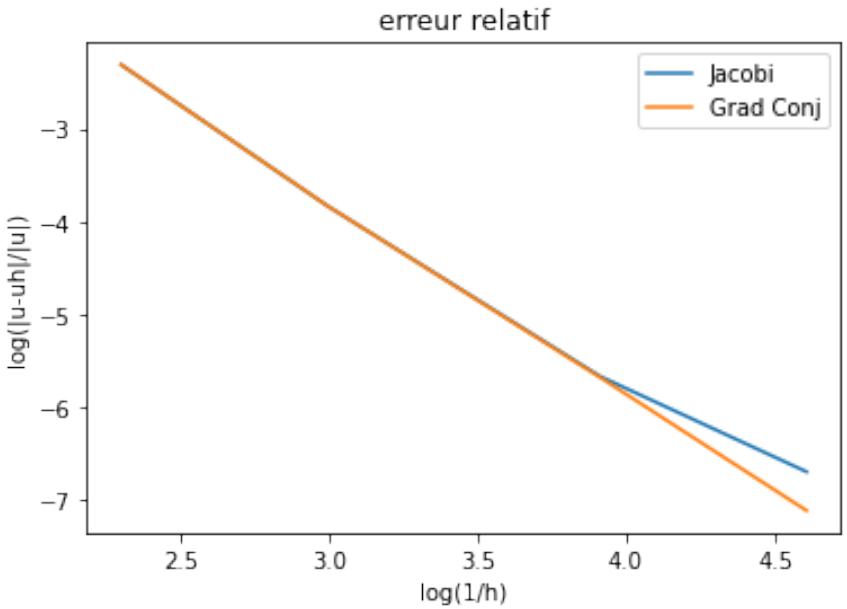
```
-> final iteration: 10000 (prescribed max: 10000)
-> final residual: 3.096742e-07 (prescribed tol: 1.000000e-06)
-> final L2 error: 1.042444e-02
```

Convergence

l'erreur relative

$$\log(\frac{1}{h}) \mapsto \log(\frac{\|u - u_h\|_{L^2}}{\|u\|_{L^2}})$$

pour $h = 0.1, 0.05, 0.02, 0.01$



Convergence

Gradient Conjugué:

h	0.1	0.05	0.02	0.01
Time (s)	0.0071	0.049	0.69	4.437
Erreur L2	4.204e-02	1.04e-02	1.69e-03	4.05e-04
Erreur relatif	9.92e-02	2.166e-02	3.445e-03	1.232e-03
Nb itération	61	111	271	441

Jacobi:

h	0.1	0.05	0.02	0.01
Time (s)	0.78	3.12	18.96	52.8
Erreur L2	4.204e-02	1.04e-02	1.712e-03	6.15e-04
Erreur relatif	9.92e-02	2.166e-02	3.406e-03	8.113e-04
Nb itération	3231	7721	10000+	10000+

Scalabilité

Jacobi:

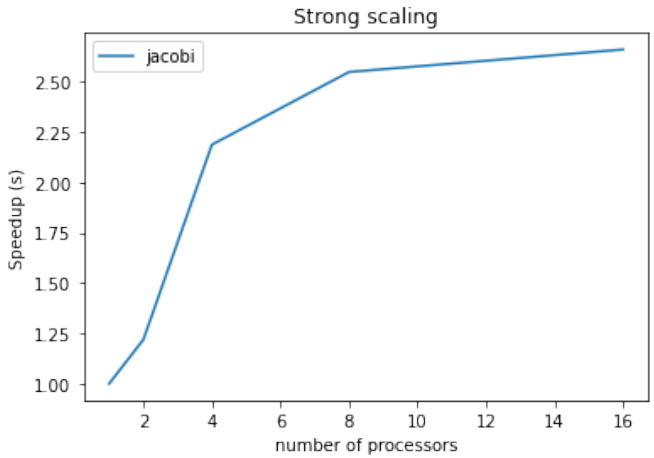


Figure: Scalabilité forte

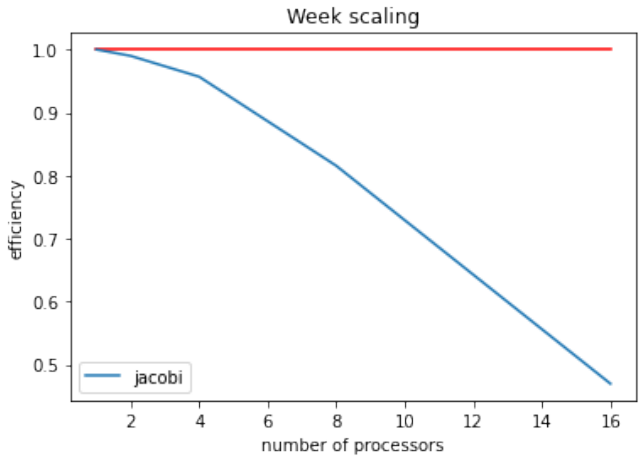


Figure: Scalabilité faible