Contribution à la conception à base de composants logiciels d'applications scientifiques parallèles.

5 novembre 2012

Vincent Pichon
LIP/Avalon - EDF/SINETICS/I2A
École Normale Supérieure de Lyon

Encadrants : Christian Pérez et André Ribes

Plan

Contexte

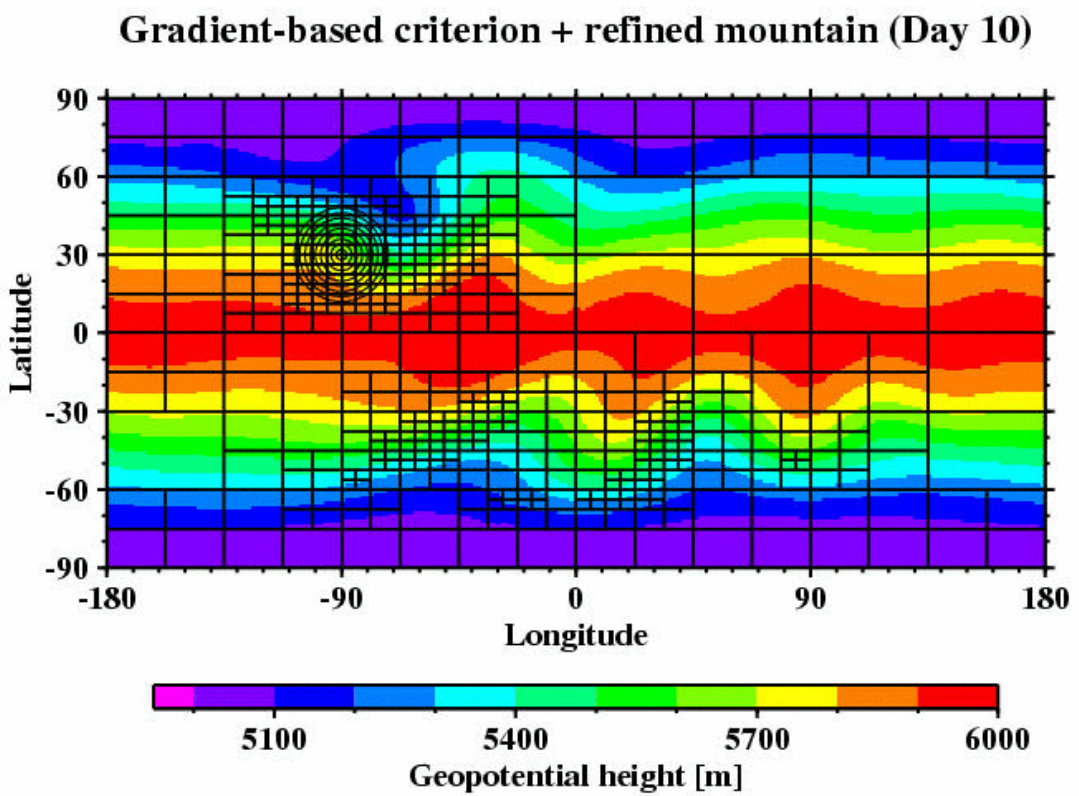
Applications scientifiques

- Simulations numériques
- Parallèles

Structures

- Couplage
- Dynamique

Complexité croissante



Ressources

Grappes de serveurs

- Topologie réseau simple, homogène
- GPU

Supercalculateurs

- Large échelle, topologie réseau, GPU

Grilles de calcul

- Fédération de ressources, hétérogène

Cloud computing

- Coût, élasticité des ressources

Modèles de programmation

Paradigmes

- Multithread (Pthread, OpenMP)
- GPU (CUDA, OpenAcc)
- Passage de message (PVM, MPI)
- Appel de méthode distante (GridRPC, CORBA)
- Passage de documents (Web Services)

Complexité

- Aucun modèle ne propose de solution adéquate
- Combinaison MPI + OpenMP ... OpenAcc
- Durée de vie d'un code ∼30 ans, d'une ressource ∼3 ans
- Portage régulier d'une ressource à l'autre
- Maintenance et évolution complexe

Modèles de composition

McIlroy, Mass Produced Software Component, 1968

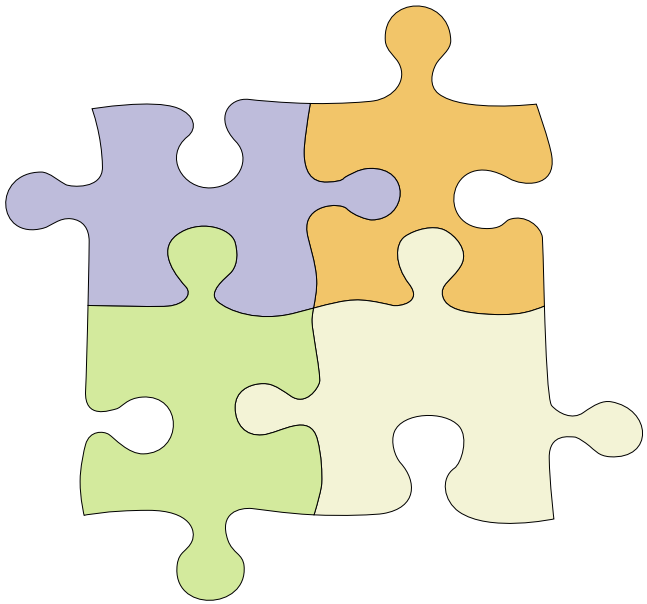
Composant

- Unité binaire indépendante déployable
- Offre des services
- Déclare dépendances de services et contexte

Assemblage

- Primitif et composite

Spatiale ou/et temporelle



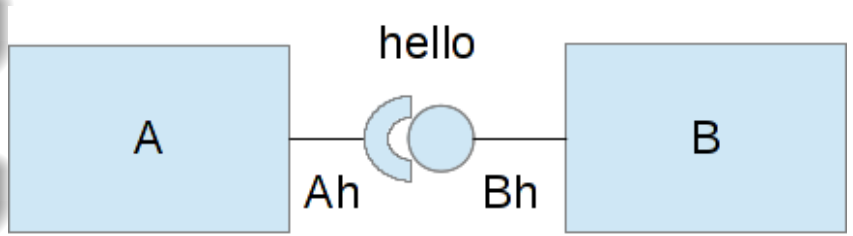
Composition spatiale

Évolution POO

Port - interfaces

Construction par composition

- Corba Component Model (CCM)
- Common Component Architecture (CCA)
- Fractal
- Grid Component Model (GCM)
- Distributed Software Component (DSC)




```
interface hello {
   void say_hello();
}
component B {
   provides hello Bh;
}
component A {
   uses hello Ah;
}

Implémentation:
pah = get_Ah_port();
pah->say_hello();
```

Composition temporelle

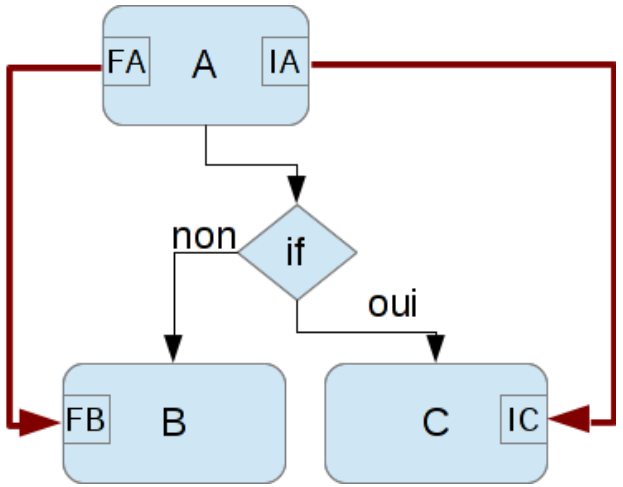
Flôt de données

- Port in out

Flôt de contrôle

- Dépendance temporelle
- Structures de contrôle

Kepler, Triana, MOTEUR, ASSIST, YACS



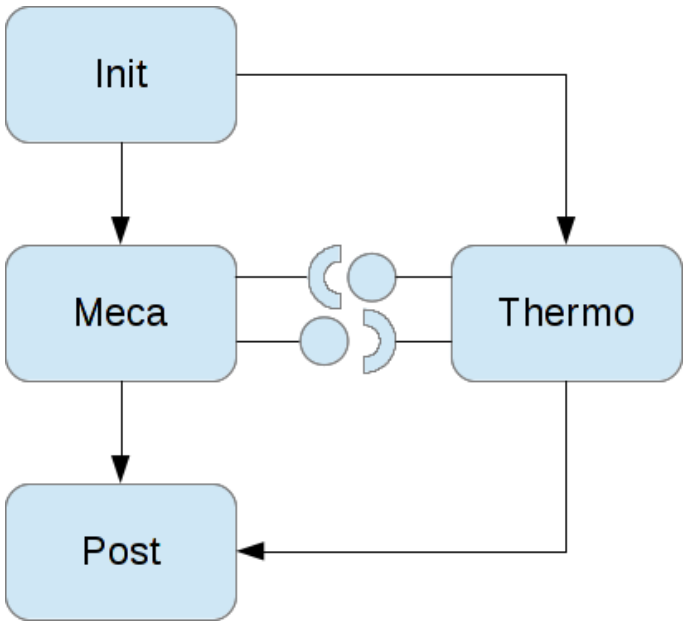

```
component A {
   dataout float FA;
}

component B {
   datain float FB;
}
```

Composition spatiale et temporelle

Dépendance spatiale et temporelle

ULCM, SALOME



Plan

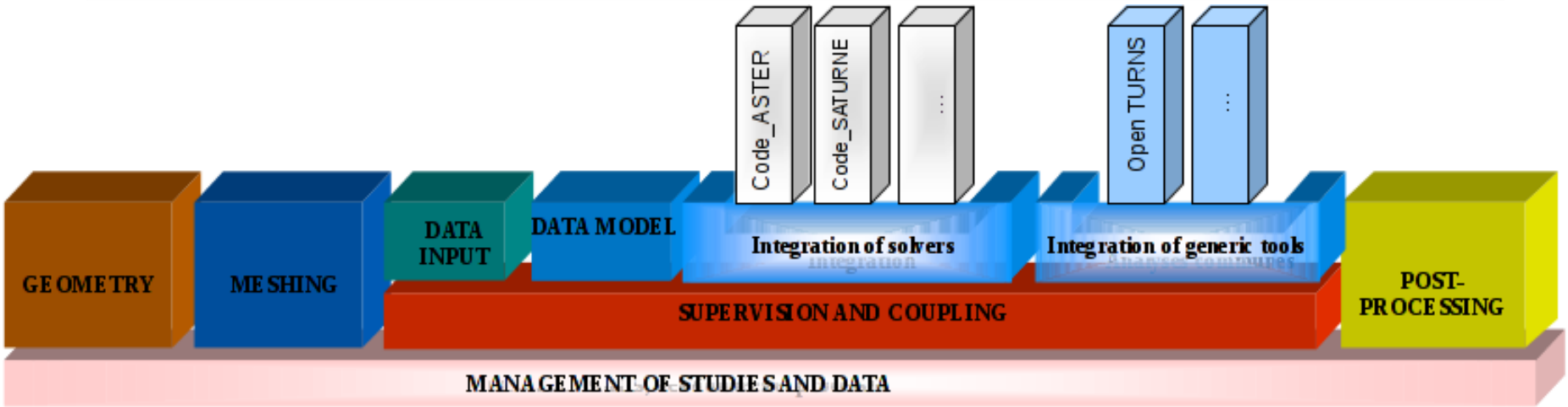
Plateforme SALOME

Plateforme Open Source d'intégration de solveurs numériques.

Développée par EDF et le CEA

Composition spatiale : modèle de composant DSC

Composition temporelle : modèle de workflow YACS



Distributed Software Component (DSC)

Module Kernel

Objet SALOMÉ

- Objet CORBA
- Cycle de vie dans SALOMÉ : Fabrique
- Blibliothèque dynamique
- Conteneur

Composant SALOMÉ

- Ports uses/provides CORBA

Service

- Fonctionnalité d'un code, méthode d'un object SALOMÉ

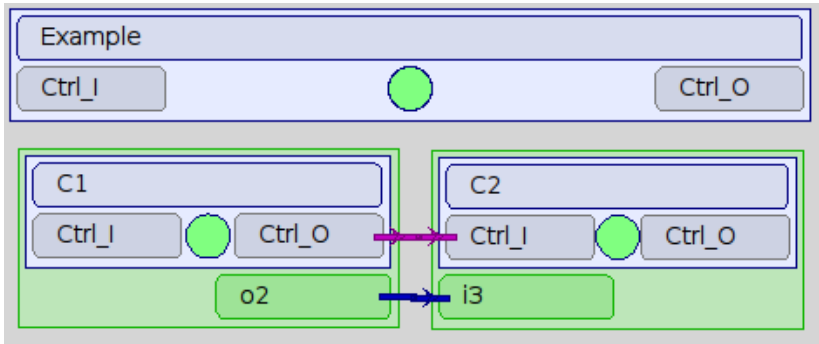
Yet Another Coupling System (YACS)

Modèle

- Workflow : schéma de couplage
- Noeud
- Structures de contrôle
- Dataflow, datastream

Superviseur

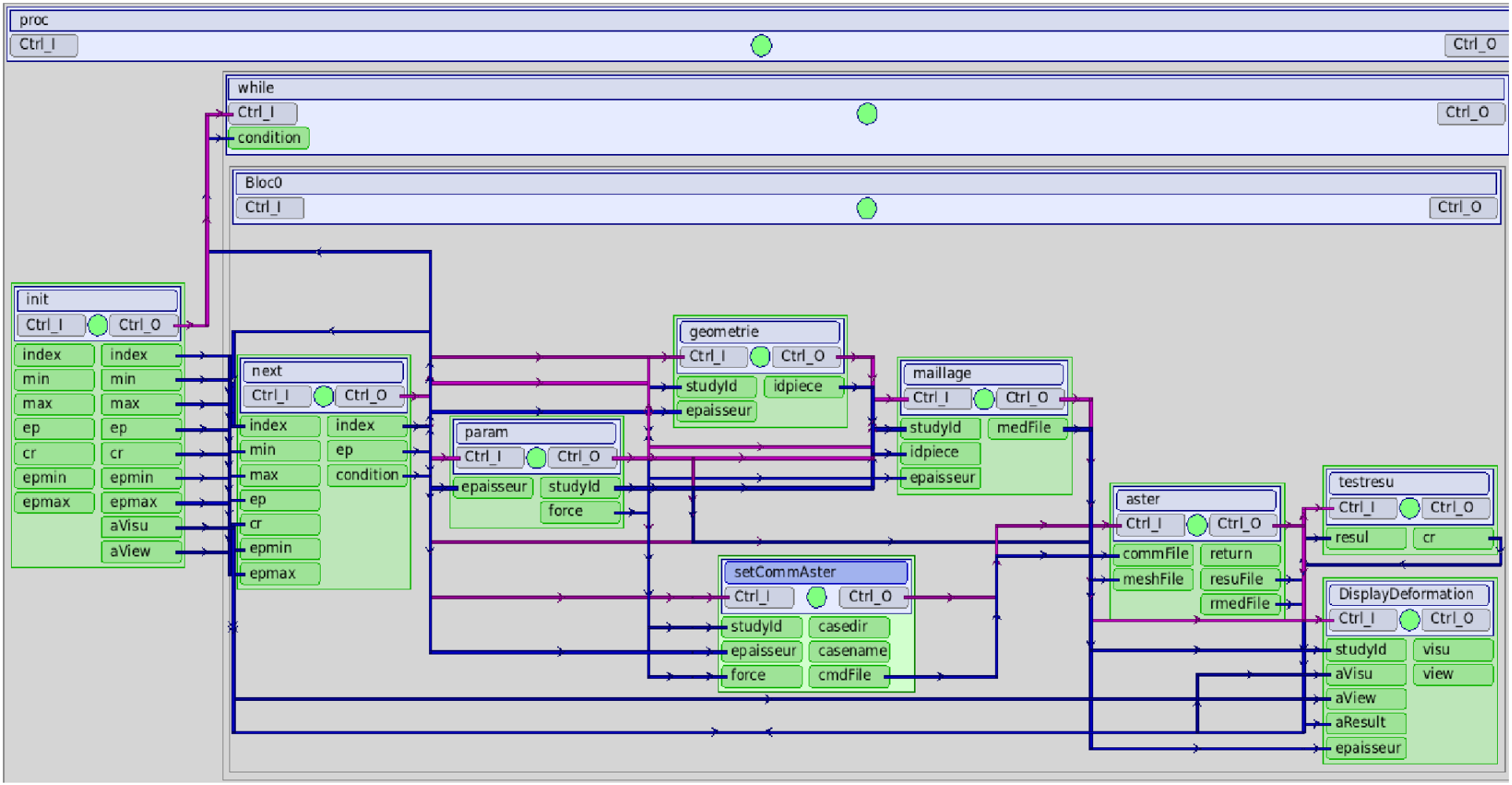
- Exécution et contrôle
- GUI
- API Python




```
interface Composant1: Engine::Component
{
  void C1(out string o2);
}

interface Composant2: Engine::Component
{
  void C2(int string i3);
}
```

Exemple de schéma YACS



Problèmatique

SALOMÉ

- Utilisée en production
- Nombreuses applications
- Étendre le modèle

Méthode

- Exemple représentatif d'une classe.
- Abstraction : décomposition de domaine.
- Dynamicité : raffinement de maillage adaptatif

Décomposition de domaine

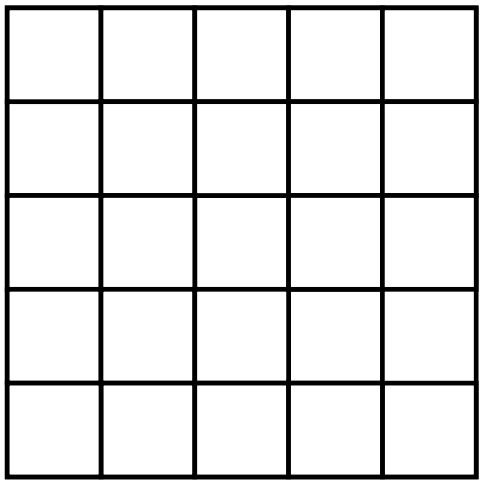
Décomposition de domaine

Simulation numérique

Précision

- Finesse de maillage

Découpage du domaine



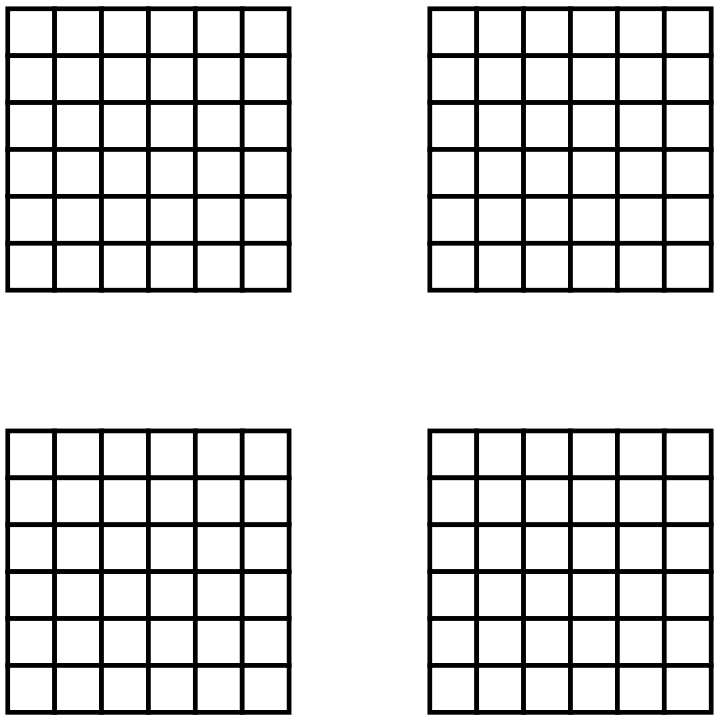
Décomposition de domaine

Simulation numérique

Pécision

- Finesse de maillage

Découpage du domaine

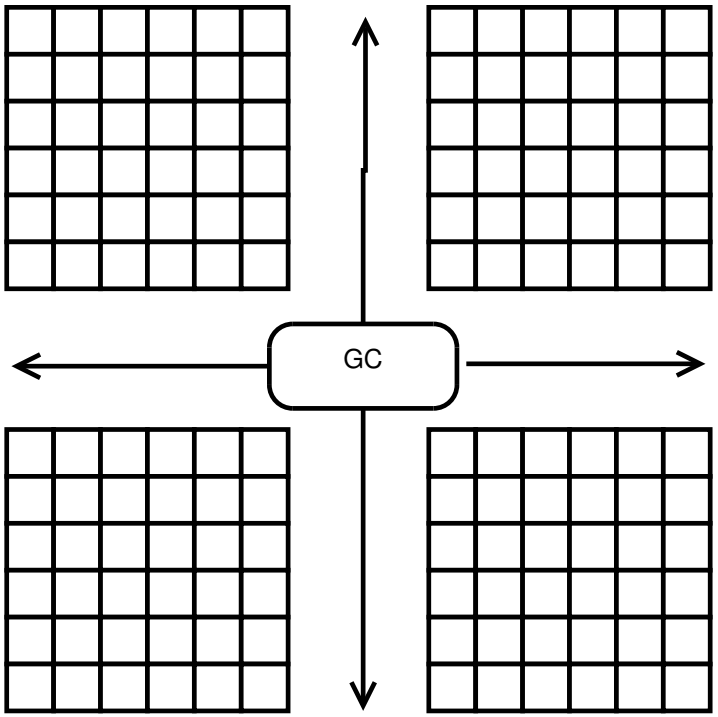


Finite Element Tearing and Interconnect (FETI)

Farhat et Roux, 1991

Algorithme

- Découpage du domaine de simulation
- Étape de calcul
- Étape de résolution du problème aux interfaces



Implémentation de FETI

Code_ASTER

- Simulation thermo-mécaniques
- 1,500,000 lignes (FORTRAN, Python, MPI)

Implementation dans Code_ASTER

- Plusieurs années
- Tâche complexe

Implementation avec SALOMÉ

- Pas de modification d'ASTER
- Algorithme FETI dans un schéma de calcul YACS
- Plusieurs mois
- Performances identiques
- Reconstruction de l'application pour chaque cas

Complexité de programmation

Paramètre : nb de sous-domaines

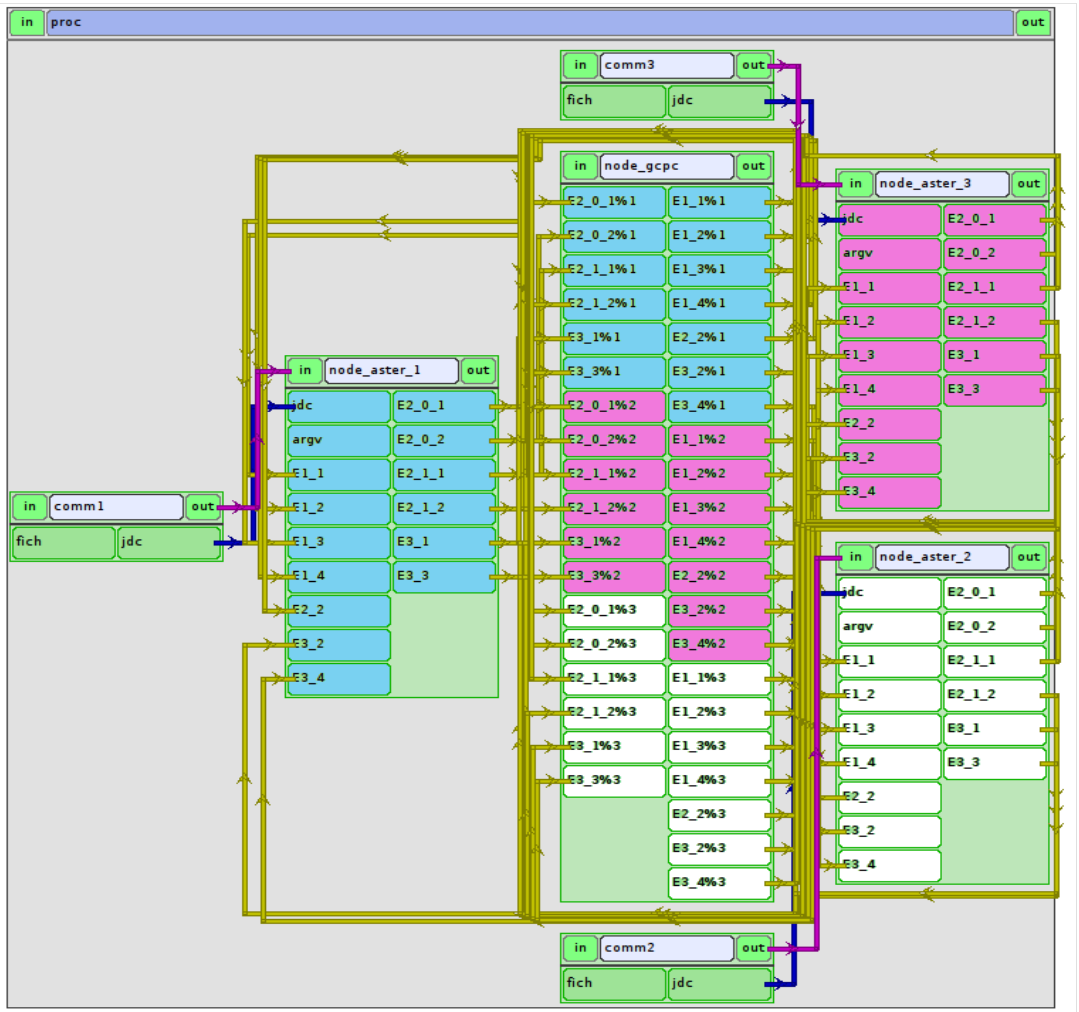
- Nb de composants ASTER
- Nb de ports

Reconstruction demandée régulièrement

Tâche fastidieuse

Scripts (38 fichiers, 2350 lignes)

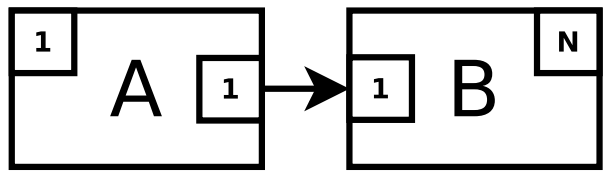
- Manque d'expressivité



Clonage de noeuds et ports

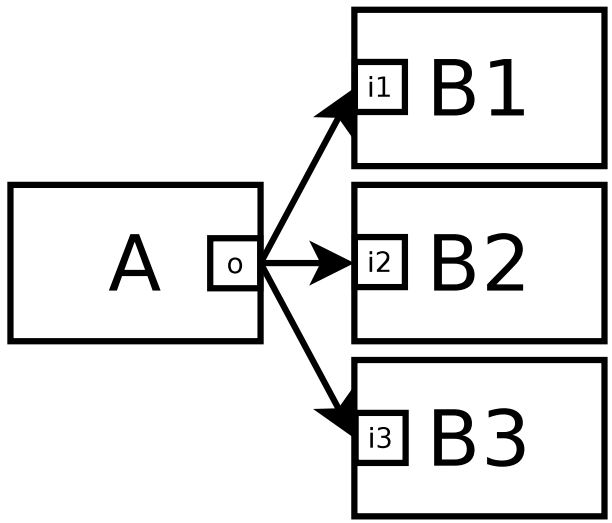
Extension du modèle de programmation

- Cardinalité attachée à chaque noeud



Clonage de noeuds

- Propriété
- Fixe le nombre de répliques



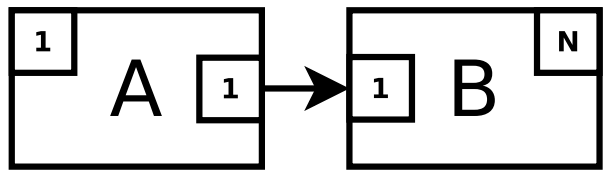
Clonage de ports

- Propriété
- Autorise le clonage

Cas possibles

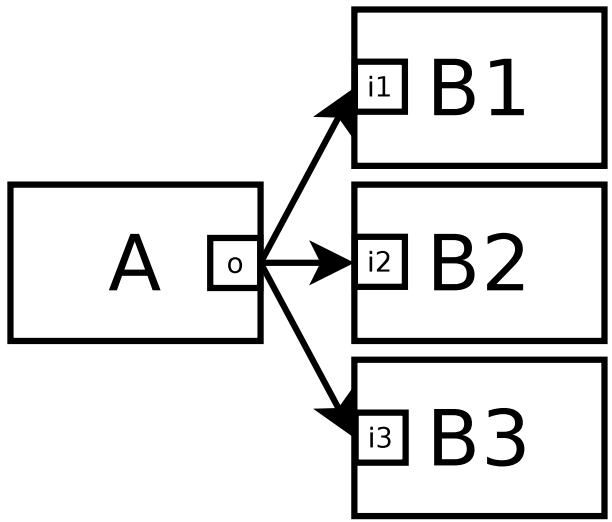
Configurations possibles

- Type de connexion (datastream/dataflow)
- Sens de la connexion
- A noeud source, B noeud de destination

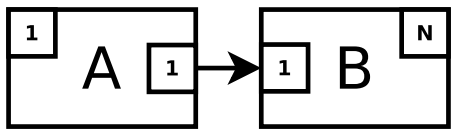


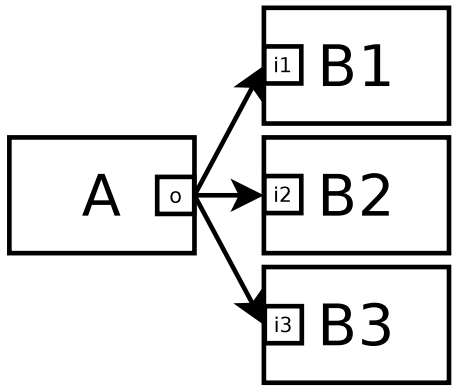
Cas

- cas NxM
- cas 1.1-N.1
- cas N.1-1.N
- cas 1.N-N.1
- cas 1.N-1.1



Cas 1.1-N.1





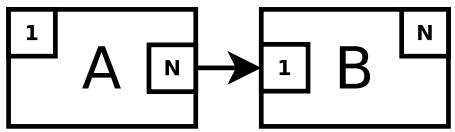
Dataflow

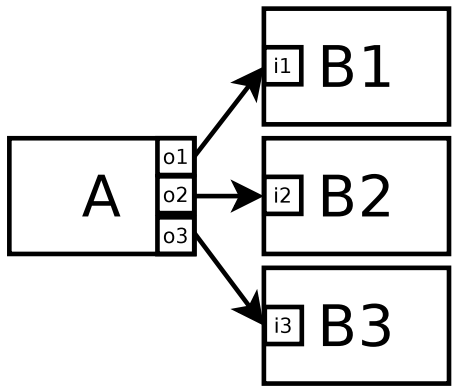
Possible (broadcast)

Datastream

Possible (uses multiple)

Cas N.1-1.N





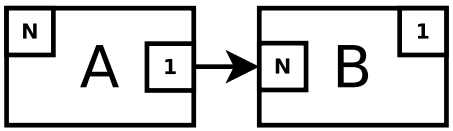
Dataflow

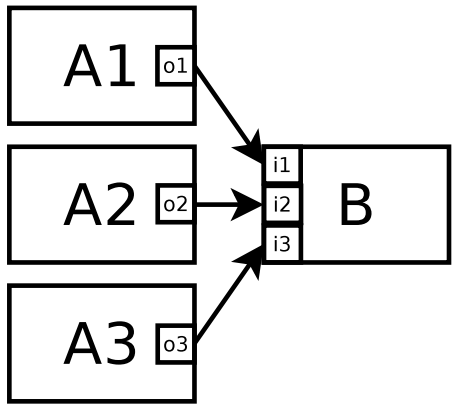
Non supporté (demande une recompilation)

Datastream

Possible (création de ports DSC)

Cas 1.N-N.1





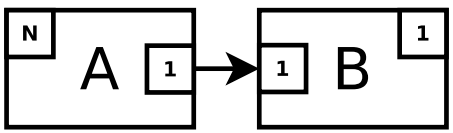
Dataflow

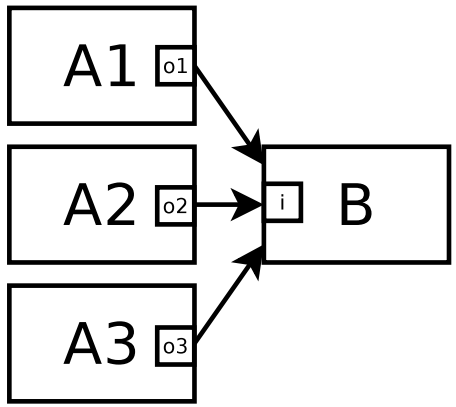
Non supporté (demande une recompilation)

Datastream

Possible (création de ports DSC)

Cas 1.N-1.1





Dataflow

Non valide (manque opérateur de réduction)

Datastream

Possible (provides multiples)

Implémentation

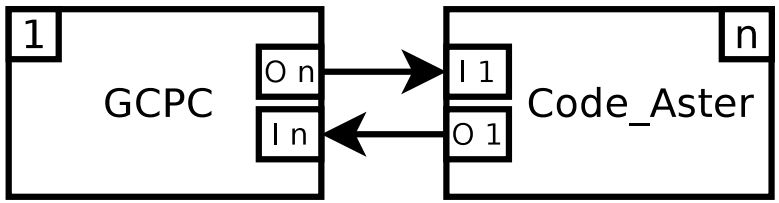
Implémentation dans la branche principale de SALOMÉ

3 classes, ∼1000 lignes de C++ ajoutées

Fonctionnement

- Interprété au chargement
- Modification intialisation
- Port propriété

Evaluation



Scripts inutiles

- Schéma de calcul générique
- Pas de recompilation

Performances égales

- Clonage avant exécution (ms)

Gain d'expressivité

Raffinement de maillage adaptatif

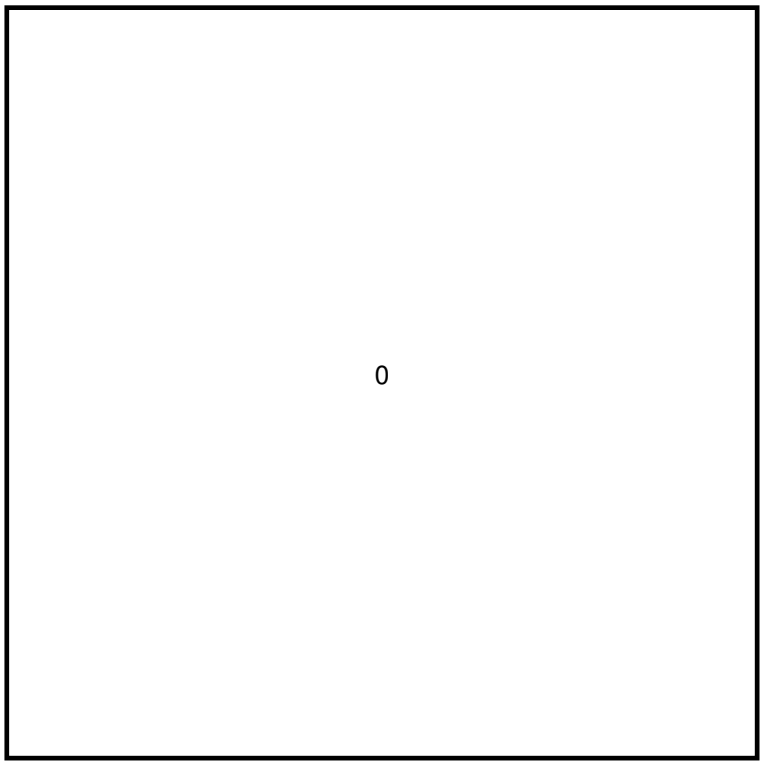
Raffinement de maillage adaptatif

- Augmenter la précision augmente le nombre de calculs
- Dans certains problèmes, l'erreur est localisée

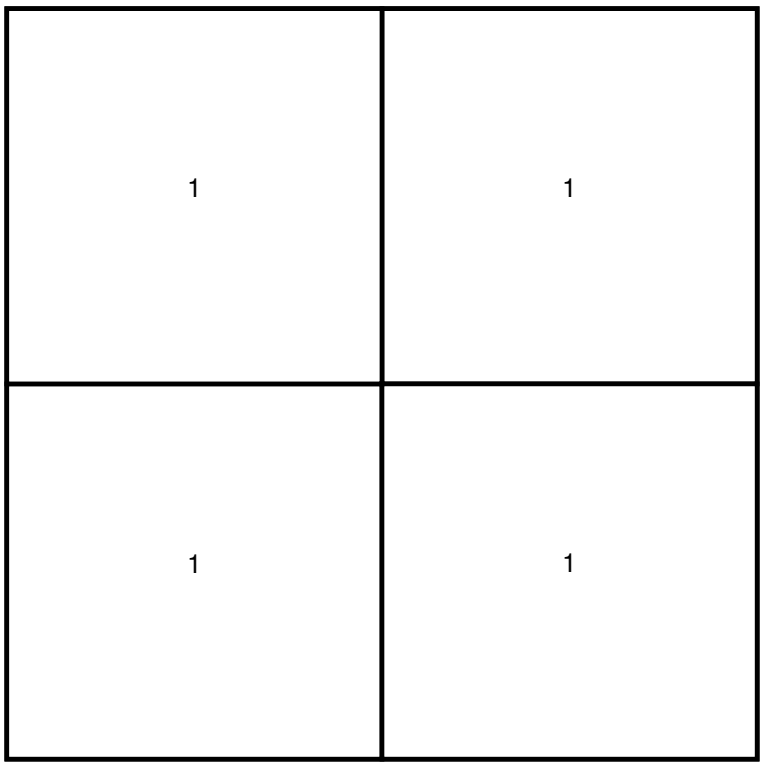
Méthode AMR

- Proposée par Berger et Oliger, 1984
- Adapter la finesse du maillage en fonction de l'erreur
- Raffinement par décomposition

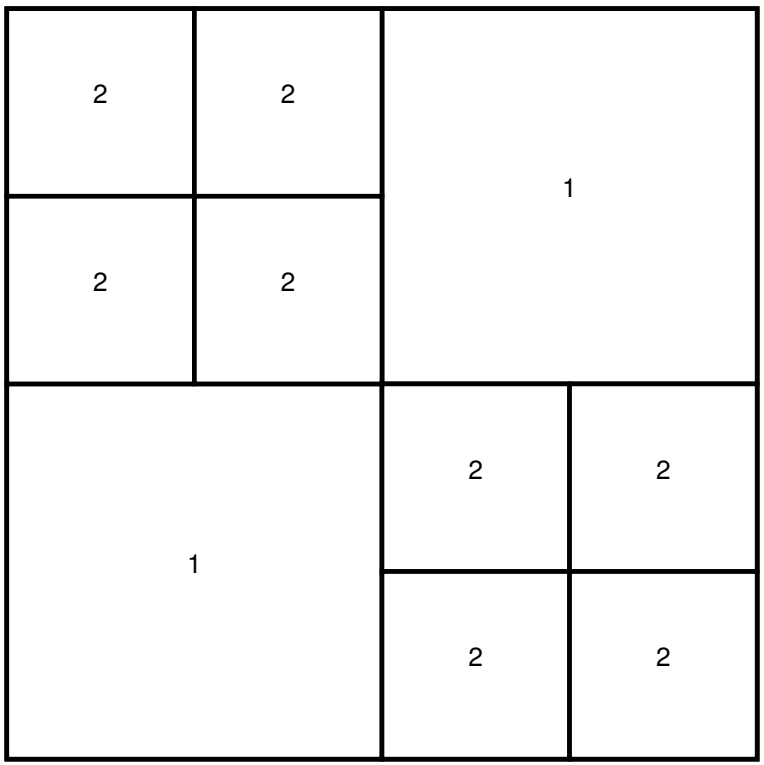
Exemple d'exécution : état initial



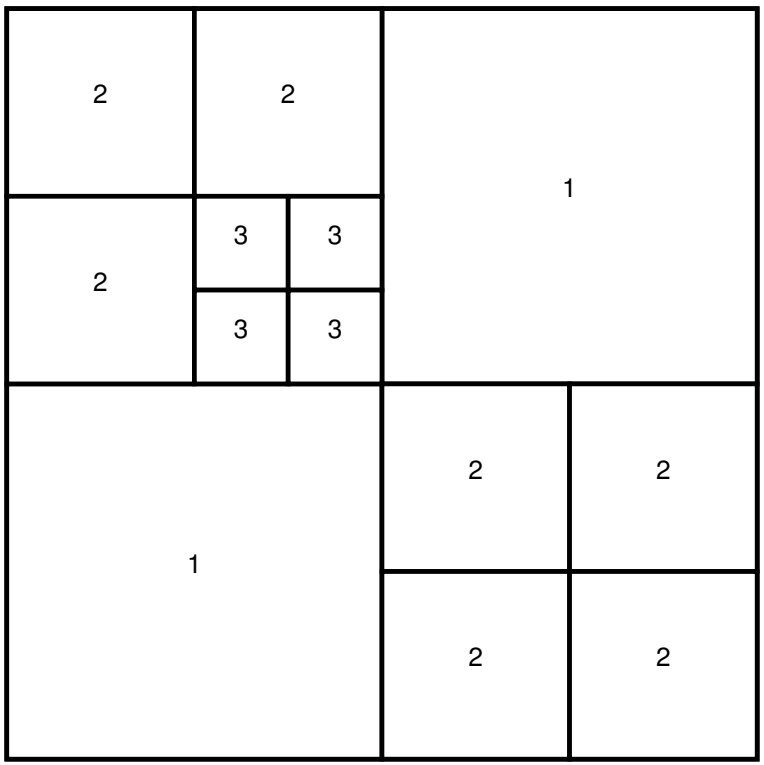
Exemple d'exécution : étape 1



Exemple d'exécution : étape 2



Exemple d'exécution : étape 3



AMR et composants

Objectif

Étudier le support de l'AMR par les modèles de composants

Domaine 2D, quad-tree

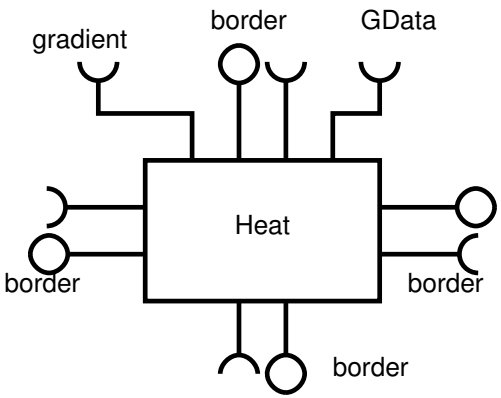
Équation de la chaleur

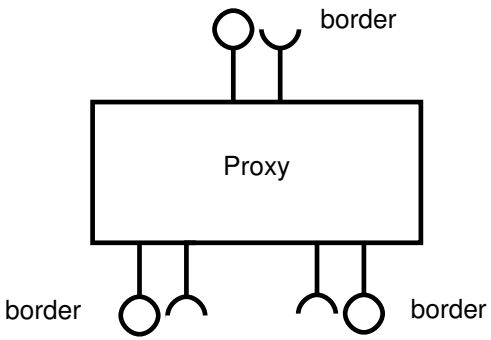
Invariant : taille des données d'un composant

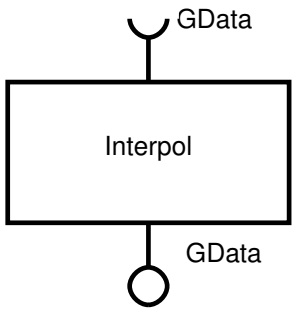
AMR et composants

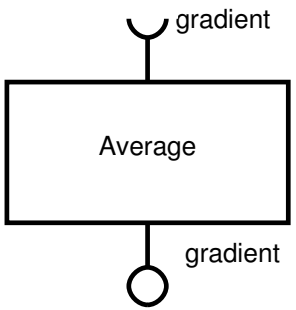
Composant primitifs

- **Heat** simule un domaine
- **Proxy** redistribue les frontières
- **Interpol** interpole les données
- **Average** rassemble l'erreur

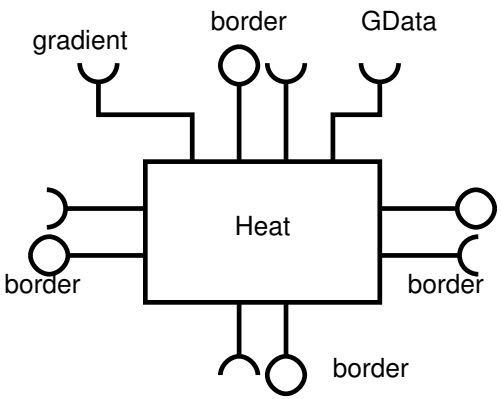


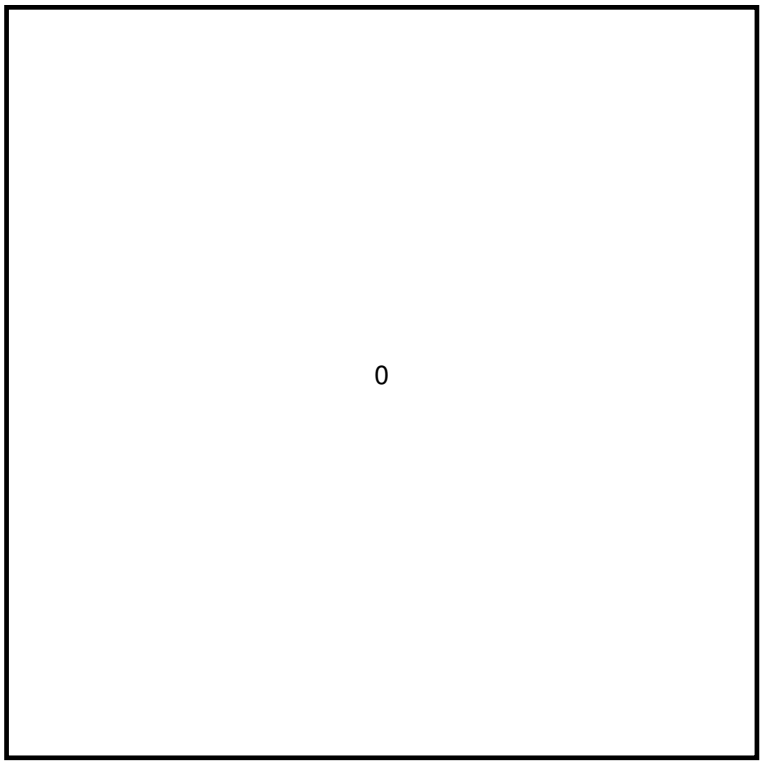




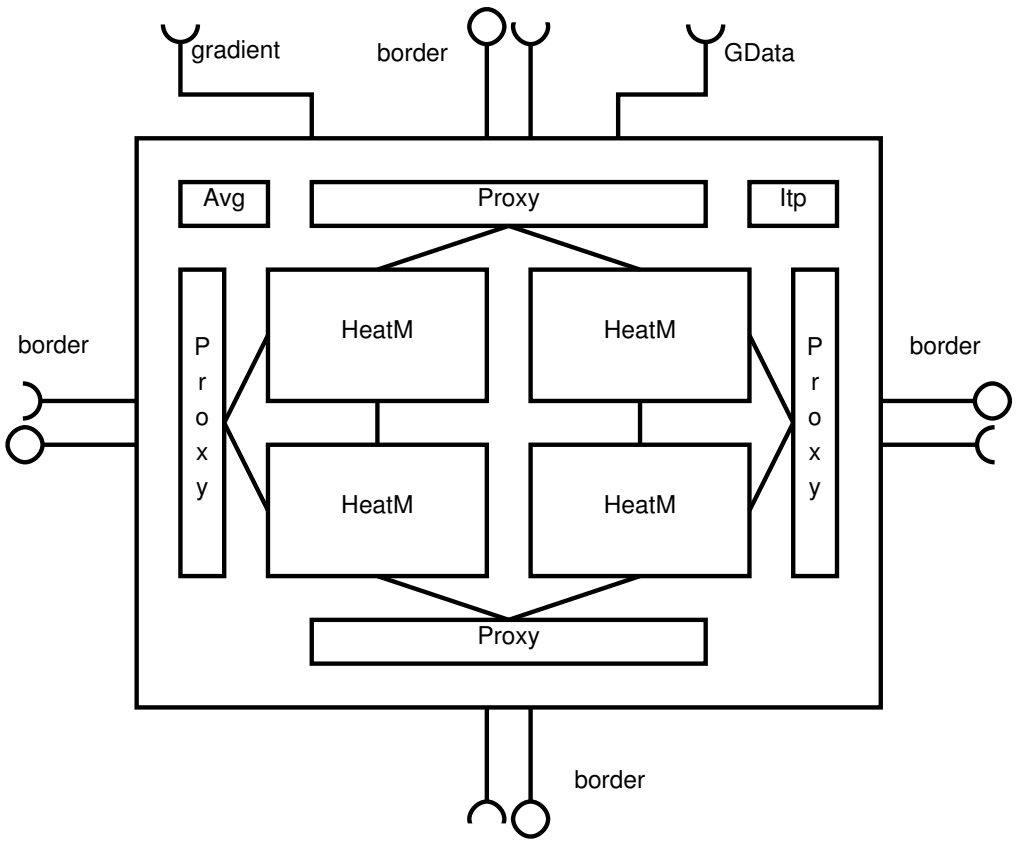


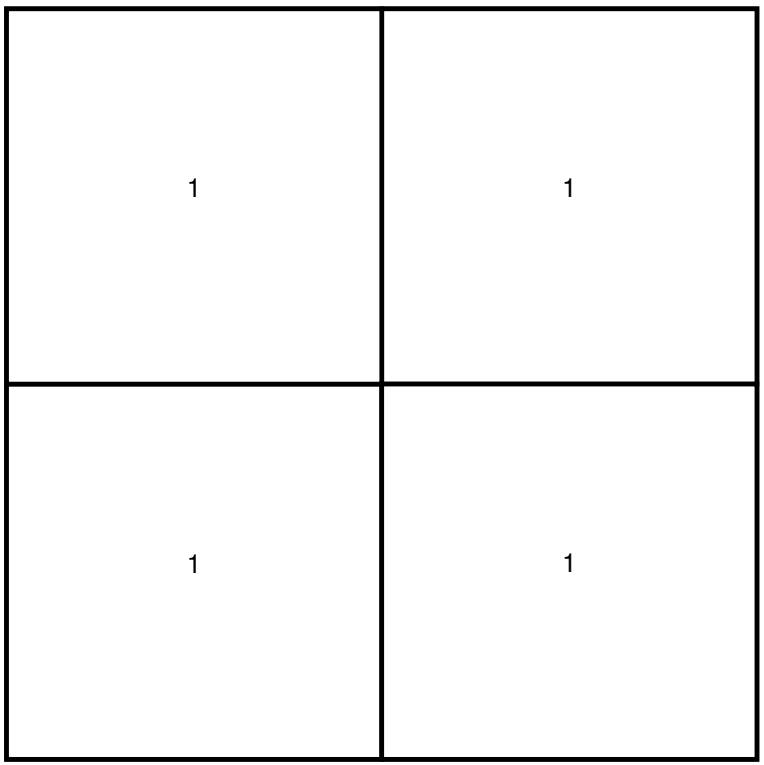
Composant HeatM





Composant HeatM





Implémentation avec SALOME

Création de composants à la demande

- API Python de YACS
- Scripts

Calcul

- Transmission de la requête de calcul aux feuilles de l'arbre

Reconfiguration

- Récupération de l'erreur
- Modification de l'architecture

Composant composite

- Gestion de l'arbre de composants

Évaluation

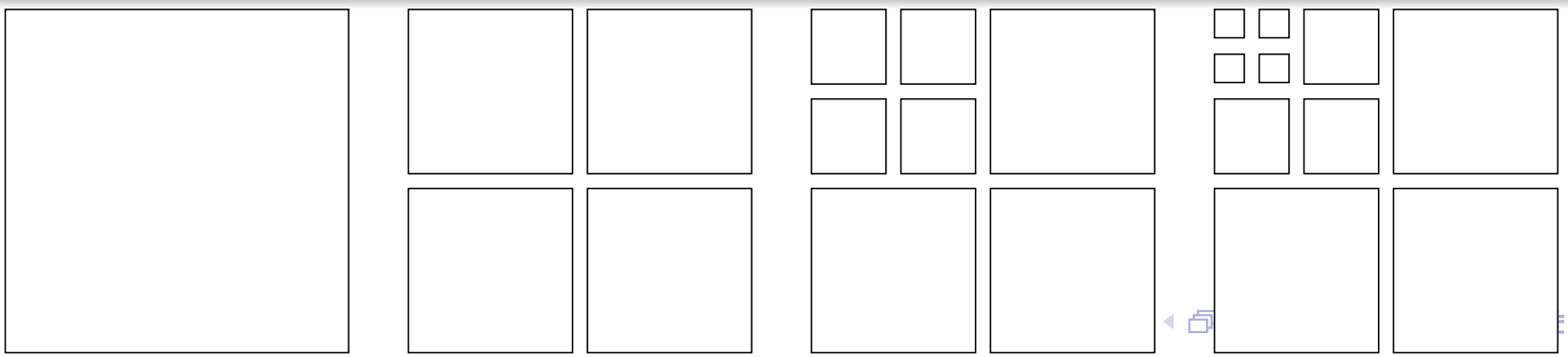
Objectifs

- Scalabilité

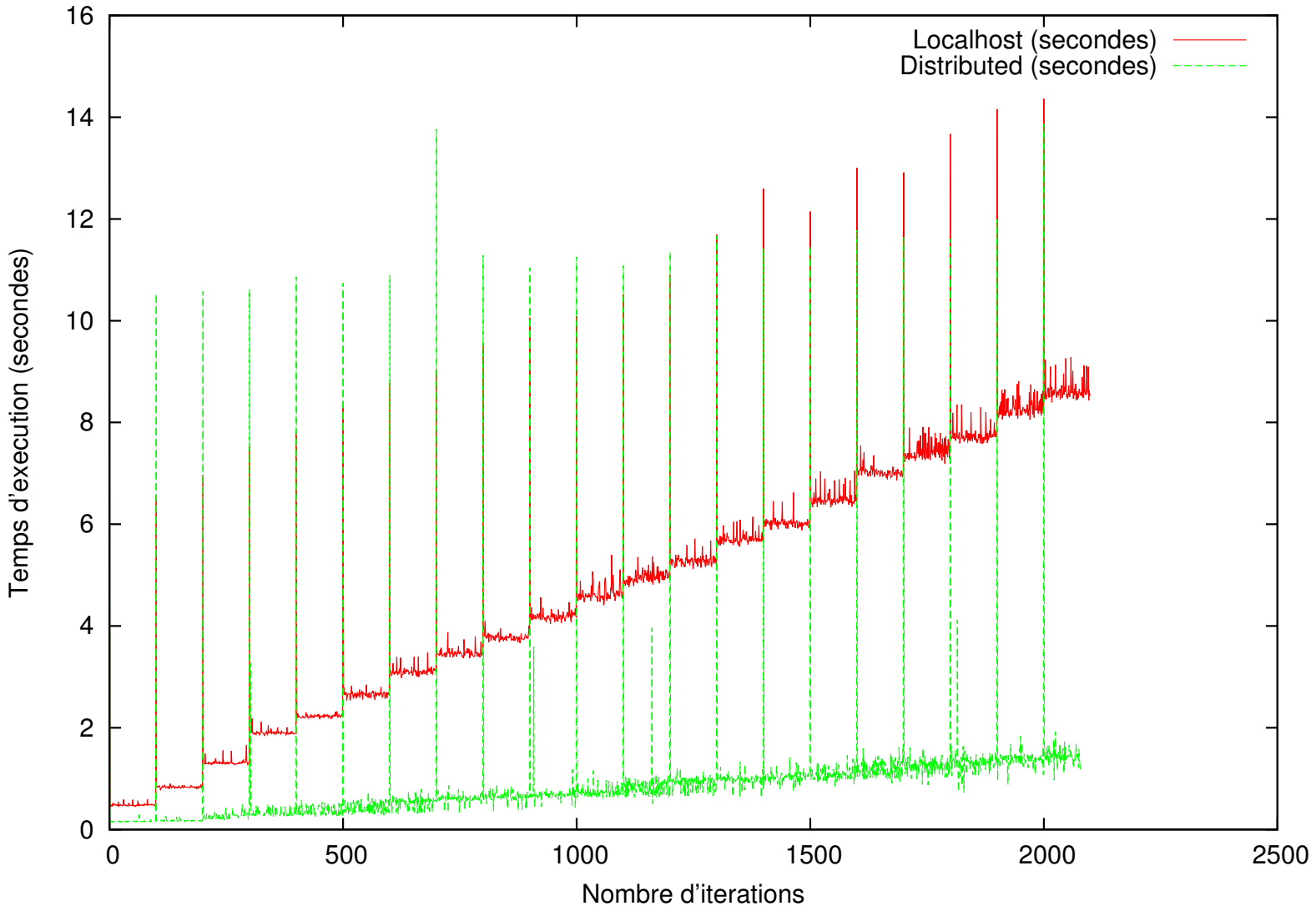
Grid'5000

Évolution d'un scénario

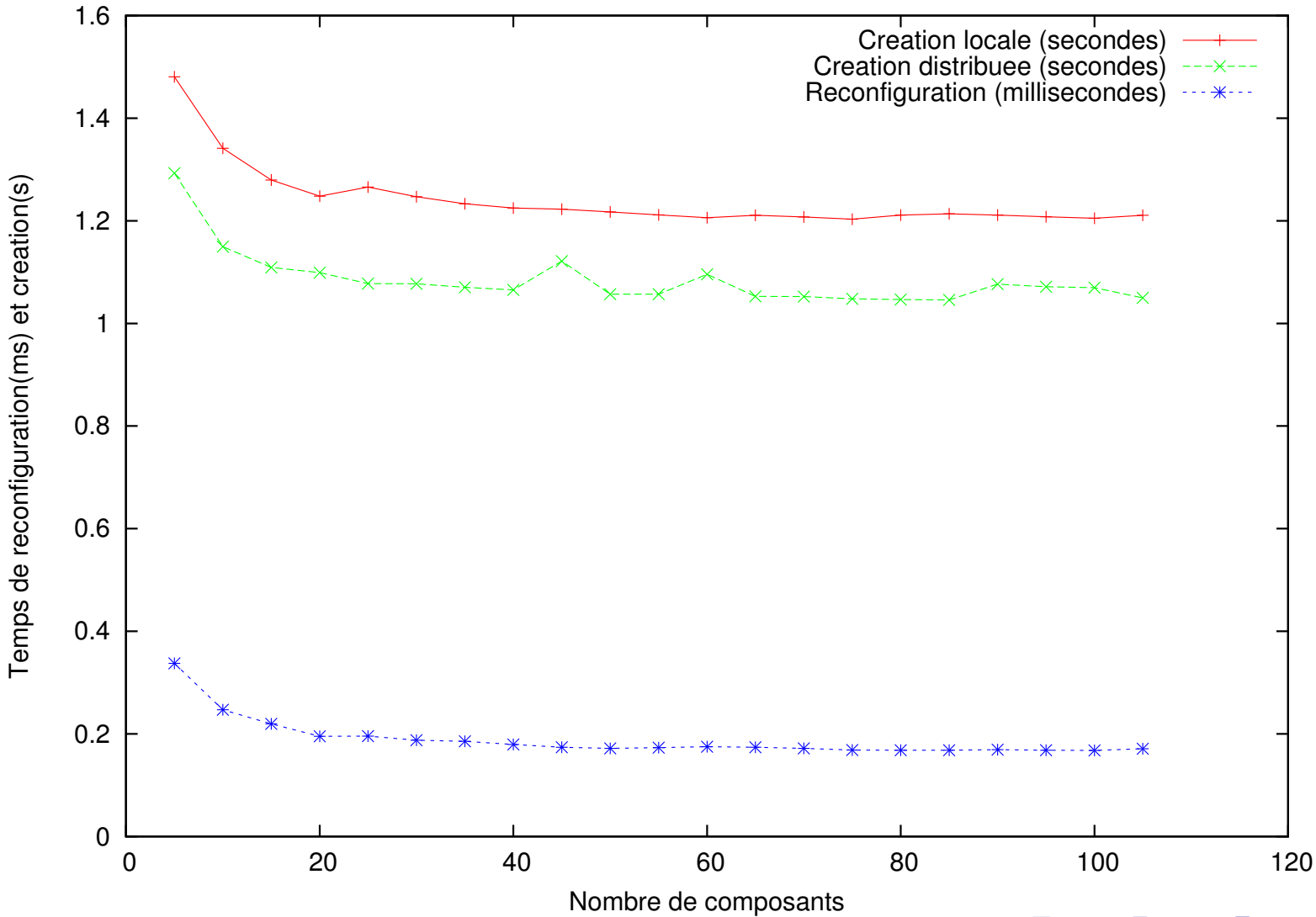
- Raffinement d'un noeud du dernier niveau
- 20 niveaux de raffinement
- 105 composants
- 100 itérations entre les étapes de raffinement



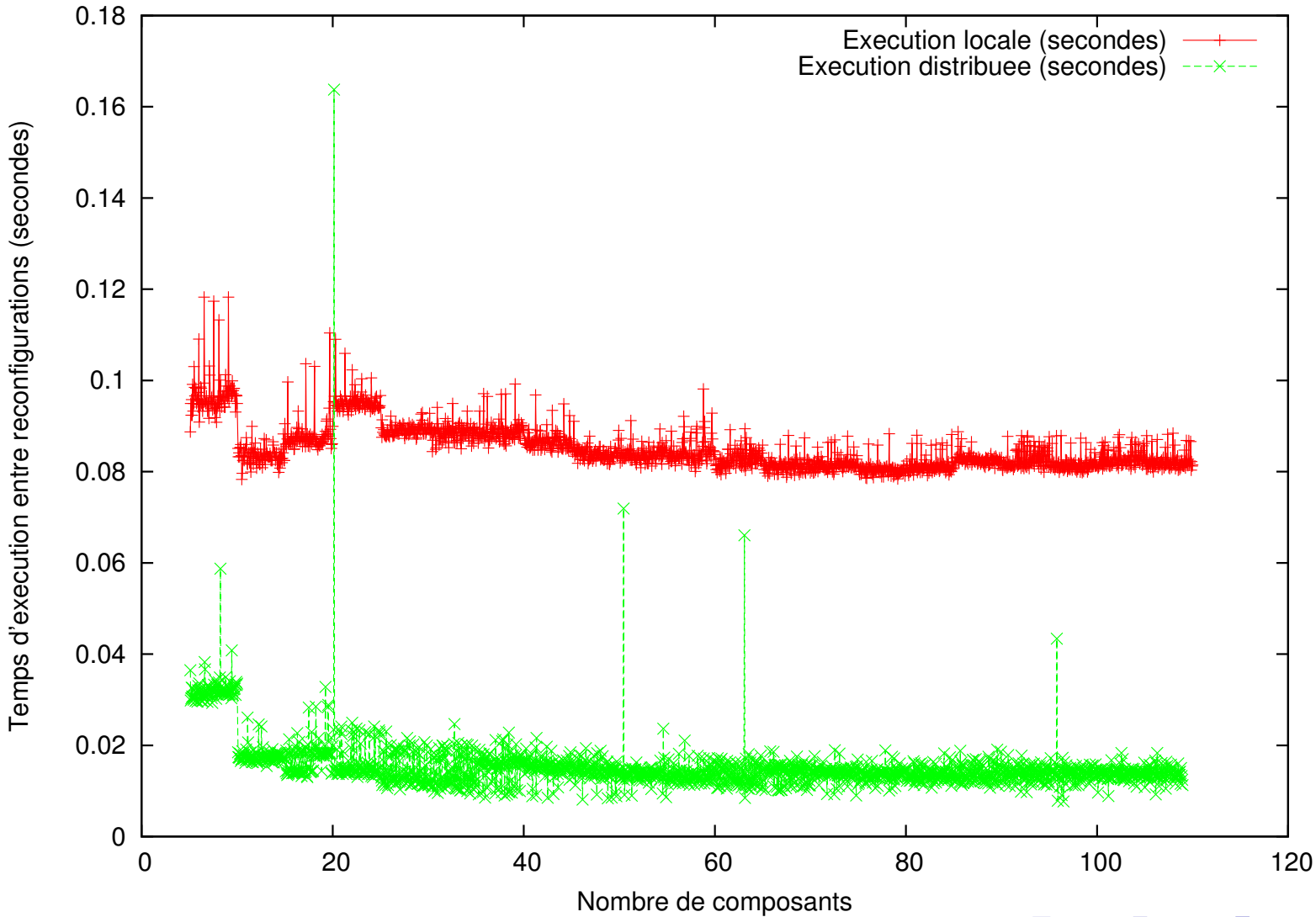
Évolution scénario



Temps de reconfiguration



Temps d'exécution



Conclusion

- Faisabilité en dehors du modèle
- Composant composite
- Dynamicité
- Déploiement séquentiel
- Moteur de workflow centralisé
- Non supperposition des phases

Décomposition de domaine et composition bas niveau

Composition bas niveau

Codes natifs

Différentes versions en fonctions des ressources visées.

- Faible réutilisation de code

Modèles de composants

- Abstraction des communications
- Pénalité de performances (CCM, CCA, ...)

Modèle de composants bas niveau

Low Level Component (L^2C)

Julien Bigot et Christian Perez

Connecteurs : Composition native (C++, MPI, CORBA)

- Création/destruction des composants
- Configuration (connexion)
- Transfert du contrôle à un composant

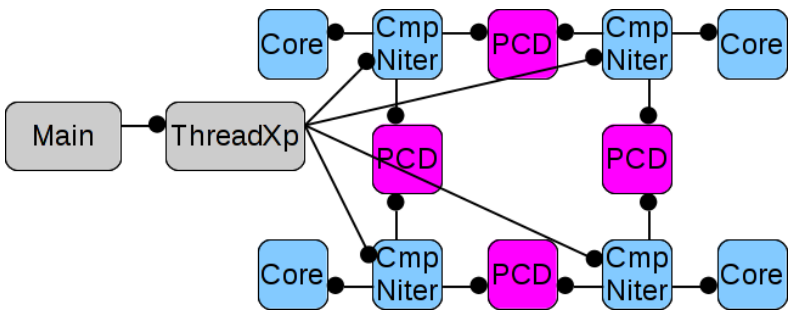
Aucun code L^2C entre les composants à l'exécution

Objectif

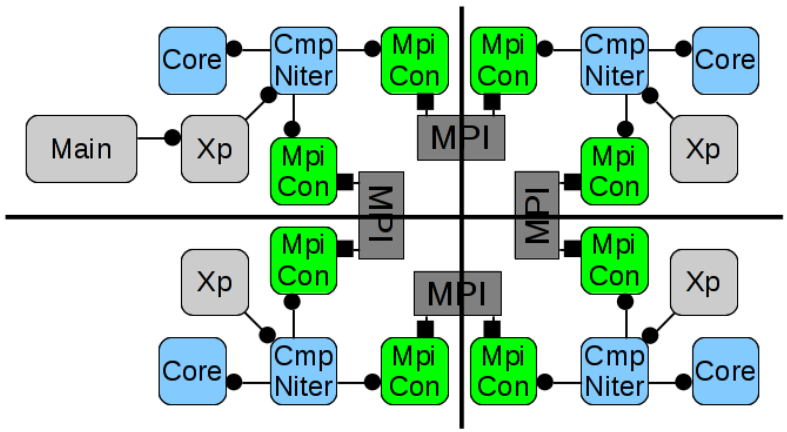
- Réutilisabilité
- Expressivité
- Performance

Thread et MPI

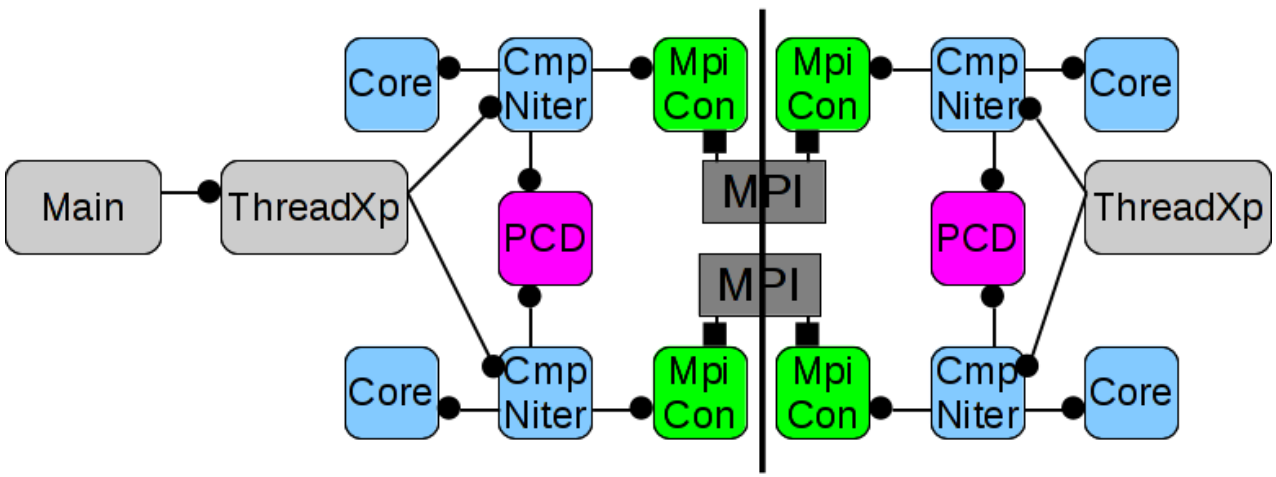
Multi-coeurs



Multi-noeuds

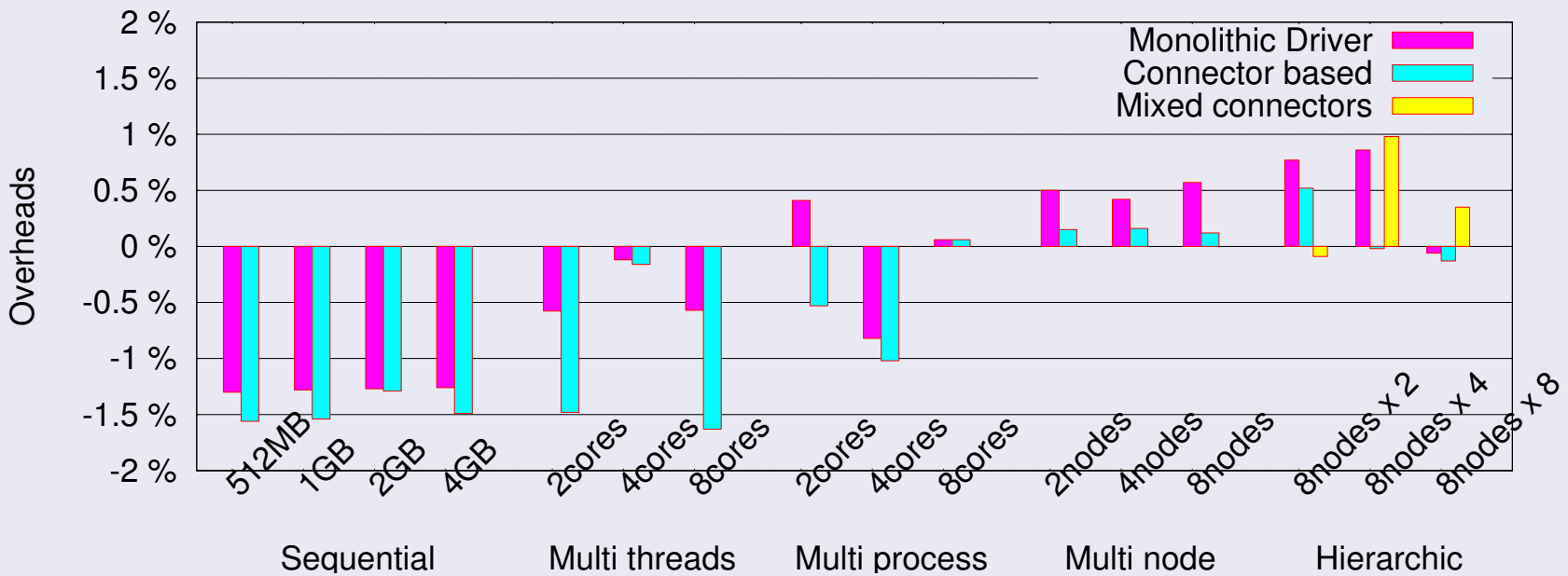


Multi-coeurs multi-noeuds (hiérarchique)



Évaluation

Performance



Complexité cyclomatique

Version	Native	Connecteur
Séquentiel	28	8
Threaded	76	26
MPI	55	13

Analyse

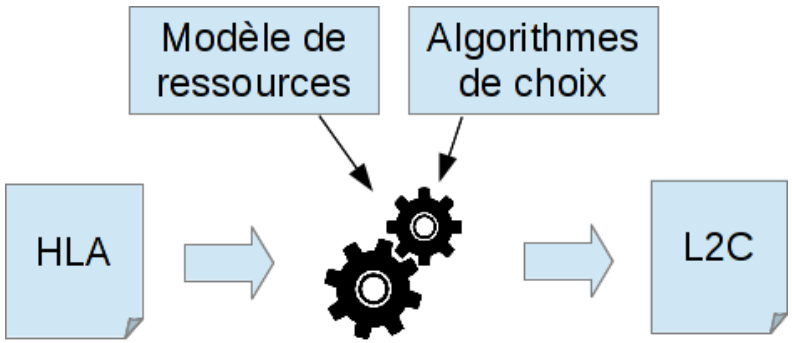
- Pas d'overhead
- Augmente l'expressivité (possibilité de combiner MPI et Thread)
- Diminue la complexité cyclomatique (séparation des rôles)

L^2C n'est pas pratique à écrire (XML)

- Assemblage produit par un modèle de plus haut niveau

HLCM

- Hiérarchique, générique avec connecteurs
- Algorithme de choix
- Modèle de ressources



Conclusion

Applications scientifiques à base de composants logiciels

Décomposition de domaine avec SALOMÉ

- Ajout du concept de clonage de noeuds/ports
- Implémentation de l'extension

AMR

- Réalisation de l'AMR avec SALOMÉ
- Identification des limites du modèle

Décomposition de domaine avec modèle bas niveau

- Étude de performance avec L^2C
- Mêmes performances que les codes natifs

Perspectives

SALOMÉ

Augmentation du niveau d'abstraction

- Composite
- Généricité

Dynamicité

- Création de composants à la demande

HLCM

Généricité

- Algorithme de choix
- Modèles de ressource

Dynamicité