

# TER

Simulation d'applications dynamiques pour plateformes de  
calculs hautes performances

Steven QUINITO MASNADA

Équipe MESCAL/LIG  
Sous la direction d'A. Legrand

Grenoble, 8 Juin 2015

- 1 Présentation générale
- 2 Analyse du problème
- 3 Méthodologie
- 4 Contribution
- 5 Conclusion

## Multicœurs



OpenMP

### API multithread :

- De plus haut niveau que PThread
- Permet d'exploiter les architectures multicœurs
- Facilite le découpage des traitements

## Hybride



### API de calculs sur GPU :

- Exécution de noyaux de calculs
- Transferts entre la RAM des GPUs et des CPUs



## Clusters



MPI

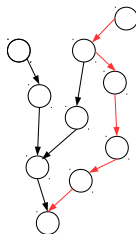
### API de communication :

- De plus haut niveau que les sockets
- Mécanismes de communication supplémentaires (exemple broadcast)

- Utiliser **plusieurs paradigmes**  $\leadsto$  **programmation complexe**
- Exemple pour exploiter efficacement un GPU sur **un seul noeud**:
  - transférer données du CPU au GPU,
  - lancer le calcul sur le GPU
  - gérer synchronisation pendant attente résultat
  - occuper CPU
  - récupérer résultat.
- Et avec **plusieurs noeuds/cores/GPUs** ?
  - **Statique**, système réglé comme une horloge  $\leadsto$  pas scalable.
  - Solution: Dynamique mais **très difficile avec APIs classiques**.

# Nouvelle approche: paradigme de tâches

- Nouvelle abstraction: les tâches
  - Plus besoin de se soucier de la ressource sur laquelle le traitement est effectué
  - Exprimer calcul en terme **graphe de tâches**  $\leadsto$  plus de **souplesse** et une meilleure **portabilité**



- Librairie StarPU:
  - Un runtime développé au LaBRI (RUNTIME/STORM).
  - Graphe de dépendances  $\leadsto$  ordonnancement **dynamique et opportuniste**
  - Première version pour système hybride et récemment StarPU MPI
- **Problématique** : Performances difficiles à évaluer car exécution du flot de contrôle non déterministe
  - Configuration **runtime**, heuristique d'ordonnancement
  - Configuration **application**, découpage en tâches

## Test sur systèmes réels

- Exécution réelle sur la plateforme cible  $\leadsto$  coûteux, difficulté d'accès
- Exécution non déterministe nécessite de réaliser beaucoup d'expériences  $\leadsto$  extrapolations difficiles

## Simulation : Généralités

- Utilisation de modèles pour prédire comportements
- Permet s'affranchir de la plateforme  $\leadsto$  peu coûteux
- Extrapolation simplifiée / simulation très rapide

# Simulation: 2 possibilités

## Simulation par rejeu de trace

Pas adapté ici car flot de contrôle non déterministe

## Hybride simulation / émulation

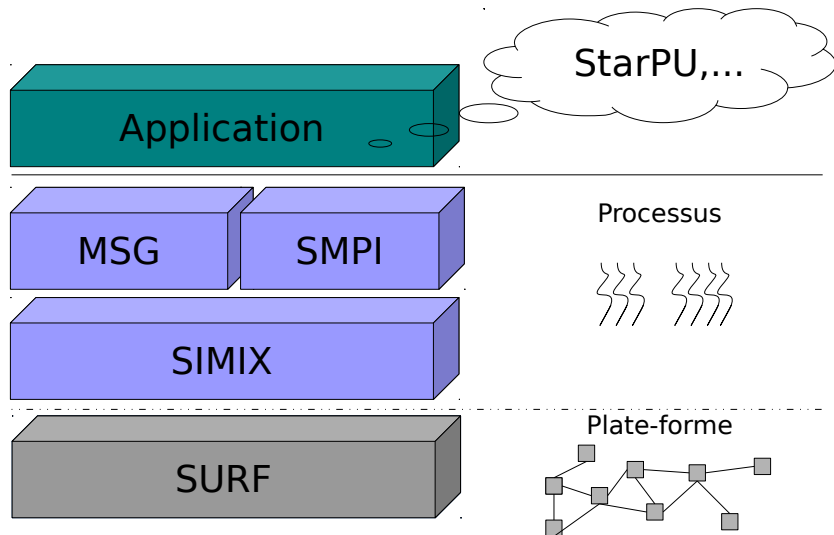
- Simulation réaliste  $\leadsto$  exécuter vrai code
- Simuler plate-forme et OS.
- Émuler de l'application.

SimGrid simulateur de systèmes distribués, grilles de calculs, systèmes peer to peer et cloud.

StarPU SimGrid, approche simulation / émulation mais avec un seul noeud.

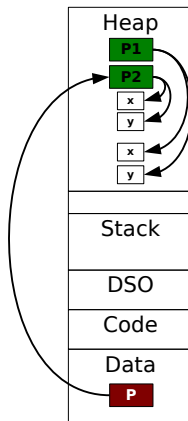
- 1 Présentation générale
- 2 Analyse du problème**
- 3 Méthodologie
- 4 Contribution
- 5 Conclusion



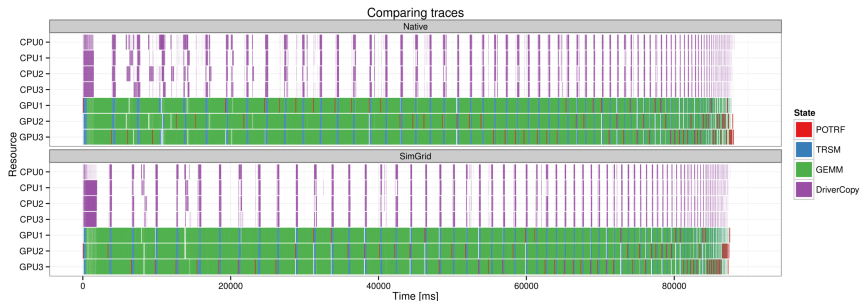


# Privatisation du segment data

- Dans SimGrid les **processus** sont **modélisés par des threads**  $\leadsto$  espace adressage partagé.
- MPI environnement à **mémoire distribuée**.
- Mécanisme de privatisation: **ségment virtuel** (mmap)



- Première version StarPU  $\leadsto$  hybride
- Basé sur MSG car **modèle de performance plus proche** (communications, environnement mémoire partagé), CPUs GPUs.
- Émulation: le vrai code de l'ordonnanceur de StarPU et de l'application sont exécutés
- Simulation: calculs, allocations mémoire des tâches, transfert CUDA.



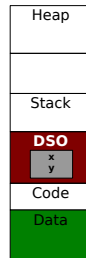
# StarPU SMPI: Difficultés de mise en oeuvre

- Besoin de 2 modèles de programmation différents à la fois:
  - StarPU intra noeuds: mémoire partagée  $\leadsto$  MSG, partage
  - StarPU inter noeuds: mémoire distribuée  $\leadsto$  SMPI, privatisation
- Besoin de 2 modèles de performance différents à la fois:
  - StarPU intra noeuds: CPU-GPU  $\leadsto$  MSG ad hoc
  - StarPU inter noeuds: réseau  $\leadsto$  SMPI

Besoin de modifications un peu complexes dans SURF  $\leadsto$  pas dans le cadre de ce stage.

- MSG et SMPI normalement pas utilisés ensemble  $\leadsto$  initialiser correctement les 2.

- Problème des bibliothèques dynamiques.



- 1 Présentation générale
- 2 Analyse du problème
- 3 Méthodologie**
- 4 Contribution
- 5 Conclusion

## Prise en main

- Dépôt git submodules:
  - StarPU SMPI:
    - SimGrid
    - StarPU
- Suivi:
  - Cahier de laboratoire org mode github.
- Compréhension:
  - Documentation.
  - SimGrid = 106 350 lignes de codes.
  - StarPU = 172 251 lignes de codes.
  - "Code mining" et vérifications: GDB, Valgrind.

## Validation

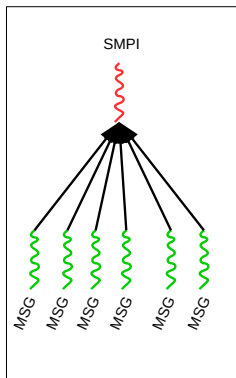
- Test simple: Modèle simplifié de StarPU MPI  $\leadsto$  isoler problèmes.
- Test StarPU: MPI, Cholesky  $\leadsto$  valider modifications

- 1 Présentation générale
- 2 Analyse du problème
- 3 Méthodologie
- 4 Contribution**
- 5 Conclusion

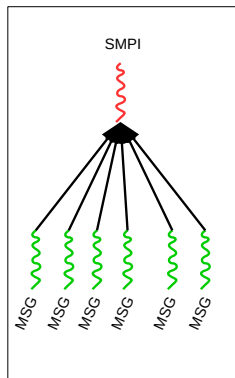
# Modification de SimGrid

- Initialisation MSG + SMPI
- Gestion du segment data: les processus MSG créés par un processus SMPI "héritent" du segment de leur père.

Noeud x



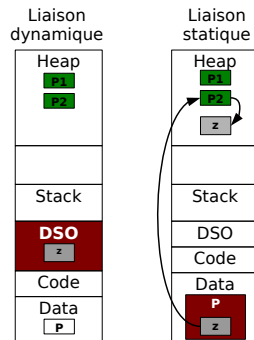
Noeud y





# Librairie dynamiques et modifications StarPU

- Bibliothèques dynamiques:
  - Utilisation bibliothèques statiques.
- Modification StarPU:
  - Initialisation, car privatisation tardive.



- 1 Présentation générale
- 2 Analyse du problème
- 3 Méthodologie
- 4 Contribution
- 5 Conclusion**

## Bilan

- StarPU + SimGrid modifié pour simuler StarPU MPI
- Difficulté: apporter modifications minimales dans un code non trivial.  
Environ 20 lignes sur un total de plus de 270 000

## Prochaine étape

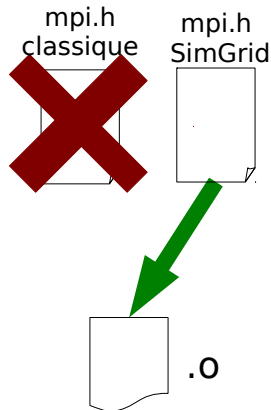
- Simulations et mesures avec solveur d'algèbre linéaire
- Vérifications système réel: Grid5000
- stabiliser le prototype (intégrer les modifications aux dépôts principaux de StarPU et de SimGrid)

Merci pour votre attention.

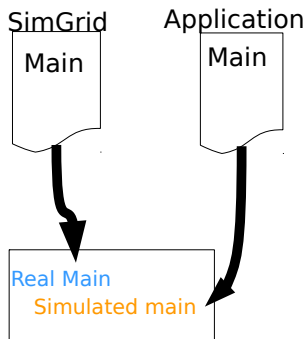
Merci à Arnaud LEGRAND pour m'avoir permis de faire un stage dans son équipe et pour avoir été autant disponible.

# Annexe: Construction de l'application MPI simulée

## Compilation



## Édition de liens



## Exécution

