Ingénieurie logicielle sur un framework de metaheuristiques

Alexandre QUEMY

15 janvier 2013

Table de matières

Inria

Inria

- Laboratoire publique dédié aux sciences du numérique.
- Création lors du Plan Calcul sous De Gaule, en 1967.

Quelques chiffres

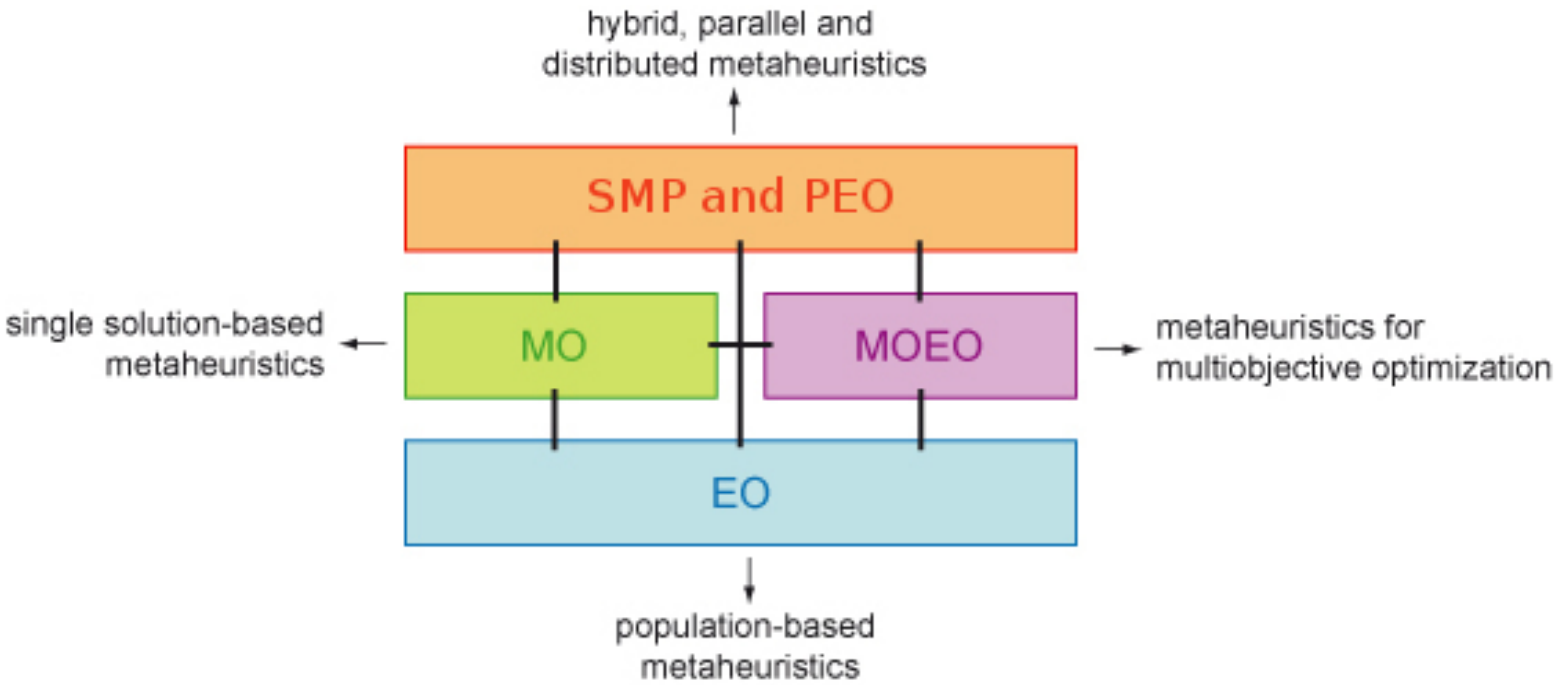
- 8 centres
- 4290 personnes, 730 stagiaires, 70 équipes
- 252 millions de budget (2010)
- 800 contrats de recherche en cours, 105 start-up créées

Présentation

ParadisEO

- Framework C++ pour le développement de méthodes d'optimisation approchées (métaheuristiques).
- Développé par l'équipe DOLPHIN à Inria Lille.
- Logiciel open-source, sous licence CeCILL.

Architecture



Missions

Stabilisation de ParadisEO

- Contexte de la fusion avec Evolving Objects.
- Pas de version stable depuis plusieurs années.
- Difficultés d'installation, de maintenabilité,...

Parallélisme à mémoire partagée

- Remplacement de la partie mémoire partagée de PEO.
- Passage à une technologie récente : C++11.

Existant & Objectifs

Existant

- Pas de version stable depuis plusieurs années.
- Support Windows délaissé.
- Installation basé sur le langage de script Bash.

Objectifs

- Sortir une version 2.0 de ParadisEO.
- Préparer la fusion avec EO.

Serveur Git

Objectifs et raisons de la migration

- Outil plus moderne que CVS ou SVN.
- Gestion de versions décentralisé.

Etapes de la migration

- Conversion automatique de l'historique SVN vers Git.
- Modification de la branche principale.
- Tag de la version 2.0.
- Conversion manuelle des branches et tags SVN vers Git.

Préstabilisation

Préambules à la stabilisation

- Correction d'erreurs syntaxiques.
- Mise à jour du code pour gcc 4.7.
- Intégration de la dernière version de EO (git).
- Découplage de OldMO et PEO.

Mise en place du système de build

Quelques besoins fonctionnels :

- Portabilité
- Build Type : Debug / Release
- Dashboard
- Packages & Installation
- Documentation
- ...

Réécriture du CMake

Méthodologie

- Réecriture incrémentale du build d'un module
- Réplication du système de build pour chaque module.
- Création du module CMake.
- Création des packages.

Réécriture du CMake

Répertoire cmake

Répertoire dédié à la construction logicielle :

- Config : cibles, flags, modes d'installation, etc.
- Macro : factorisation des comportements de construction.
- Package : construction logiciel multiplateforme.
- Target : diverses cibles (doc globale, profiling, mrproper).
- Module CMake.

Module CMake

Objectif

Trouver ParadisEO et pouvoir charger ses composants facilement.

Utilisation basique

```
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH}"
${CMAKE_SOURCE_DIR}/cmake/module")
find_package(Paradiseo COMPONENTS moeo eoutils eo)
include_directories(${PARADISEO_INCLUDE_DIR})
```

QuickStart & Documentation

Fournir un cadre sain d'utilisation du framework par :

- Un projet exemple utilisant ParadisEO, couplé à CMake.
- Un guide proposant des conseils basiques.
- Un squelette de projet.

Objectifs

- Limiter la difficulté liée à l'utilisation de ParadisEO.
- Augmenter la qualité des contributions.

Introduction au parallélisme

Mémoire partagée

- Mémoire commune : processus légers / threads.
- Synchronisation via des mutex, sémaphores.

Technologies

- pThread, std : :thread, boost : :thread.
- OpenMP

Avantages

Overhead très faible.

Introduction au parallélisme

Mémoire distribuée

- Cluster, grilles. Pas de mémoire commune.
- Envoie de messages par le réseau.

Technologies

Différentes implémentation de MPI : OpenMPI, MPICH2, etc.

Désavantages

Cohérence mémoire, tolérance à la panne.

Existant

Problèmes de PEO

- Mélange distribuée et partagée.
- Problème en mode asynchrone.
- Code peu maintenable.
- Choix de design perfectibles.

Difficultés du découplage

Technique utilisée

Retro-ingénierie avec BoUML.

Difficulté

- Couplage extrêmement fort entre MPI et pThread.

Ceci a conduit à l'abandon de cette approche.

Objectifs & Contraintes

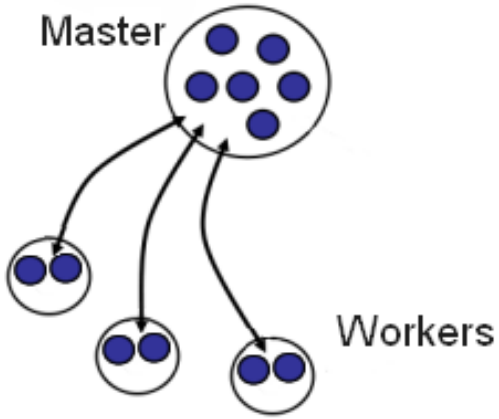
Objectifs

- Fournir une base fonctionnelle intégrable à ParadisEO 2.0.
- Mise en place d'un pattern Maître / Esclave.

Contraintes

- Proposer une API très simple par une généricité maximale.
- Indépendance vis à vis des autres modules.
- Taux de couverture acceptable ($\geq 75\%$).

Pattern Maître / Esclave



Objectif

Répartir la charge d'une opération coûteuse entre plusieurs unités, les workers, afin de diviser le temps de calcul.

Maître / Esclave pour ParadisEO

Interêt

Réduire le temps d'évaluation des populations.

Eléments de modélisation

- Le maître est un wrapper héritant de l'algorithme.
- Le maître possède un ≪ Scheduler ≫ pour répartir la charge.
- Des paquets d'individus sont envoyés sur les workers par le Scheduler.
- Les workers sont des threads *bindés* avec un foncteur d'évaluation.

Approche non intrusive

Avantages

- Tirer avantage de la métaprogrammation du C++11.
- Grande scalabilité.

Raison de la non faisabilité :

- EO est agé : présente des idomes dépassés.
- Quelques choix de design curieux.

Approche intrusive

Fonctionnement

- Un foncteur par algorithme implémenté.
- Appel du bon foncteur par tag-dispatching.

Inconvénients

- Recopie d'une partie du code.
- Limite l'abstraction et la facilité d'utilisation.
- Oblige à recompiler ParadisEO pour intégrer son algorithme.

Politiques

Deux politiques principales :

Linéaire

Découpage unique, distribution équitable des individus aux workers.

Avantage

Meilleur speed-up dans le cas moyen.

Désavantage

Temps final dépendant du worker le plus lent.

Politiques

Deux politiques principales :

Progressive (ou exponentielle)

- Planning prévisionnel des charges des workers.
- Chargement progressif et indépendant des workers.

Avantage

- Tolérance à la saturation des coeurs importante.
- Speed-up supérieur dans le meilleur des cas.

Désavantage

- Difficulté du calibrage.
- Speed-up moyen plus faible.

Tests

Méthode

- Créer un algorithme basique.
- Fixer une graine pour un résultat déterministe.
- Vérifier que la valeur sérialisée correspond à la valeur parallèle.

Remarques

- Assez peu de tests du fait de la généricité.
- Un test par type d'algorithme implémenté.

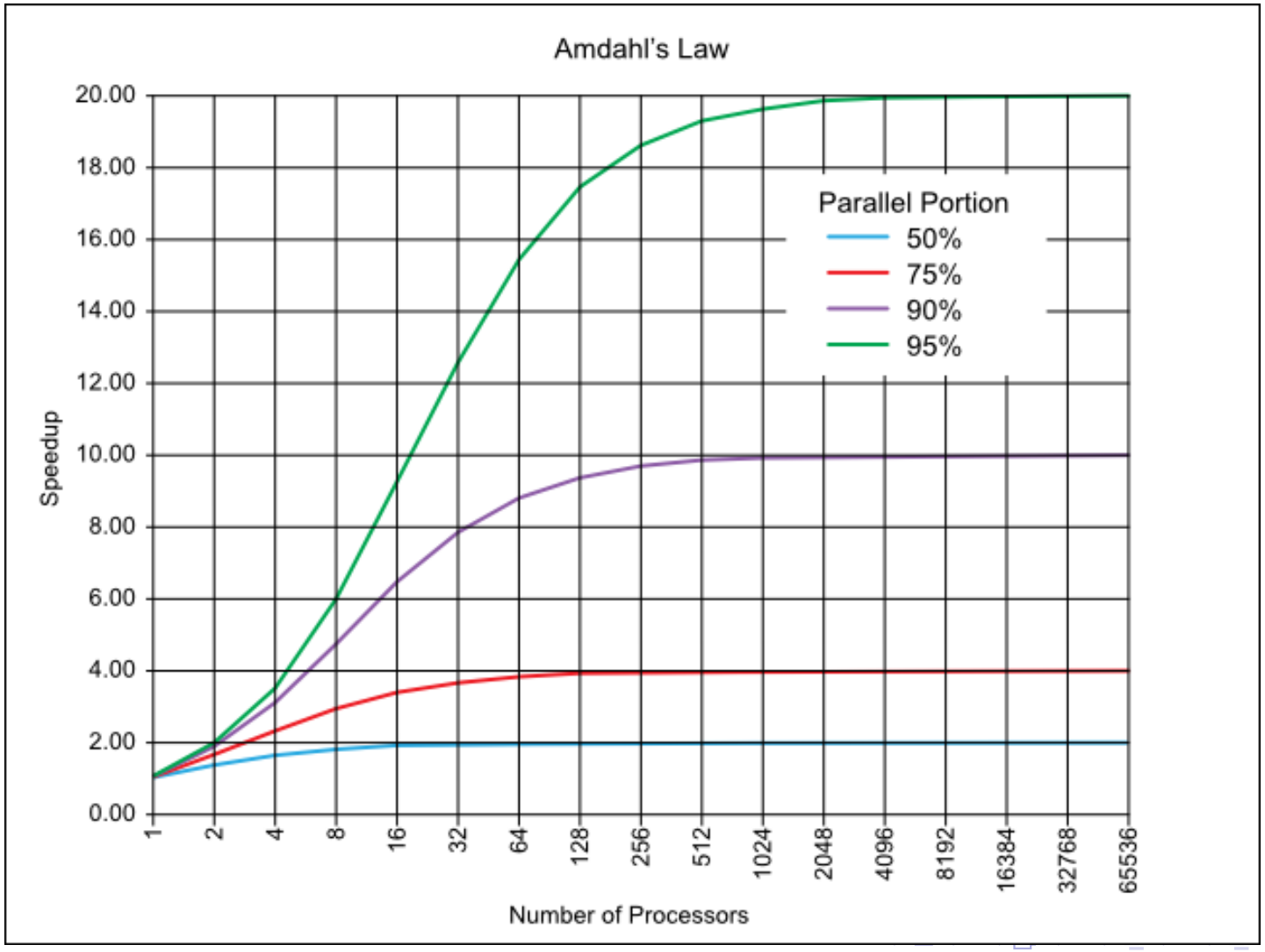
Performances

Loi d'Amdahl

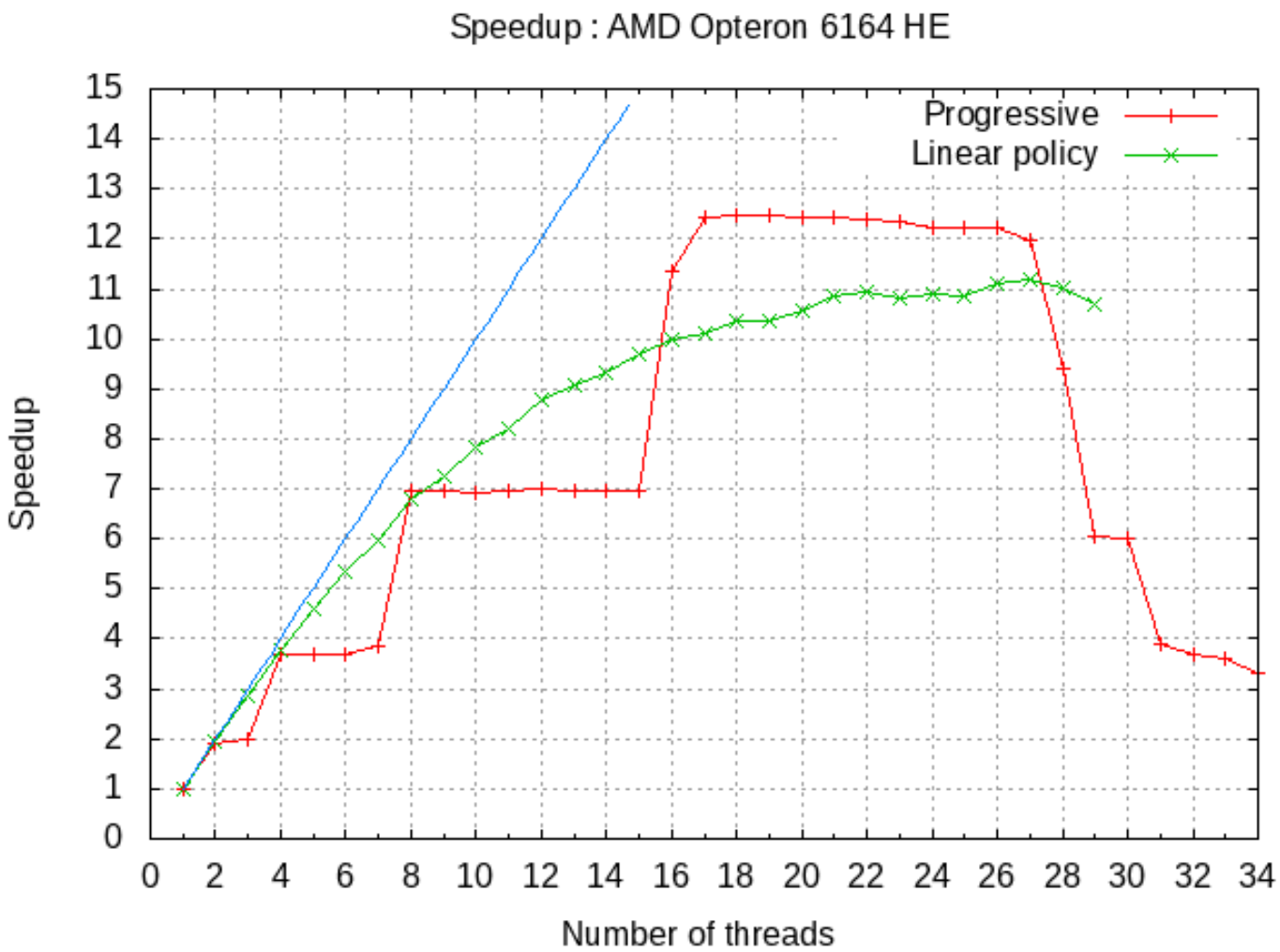
$$T_n = (1 - s)T_a + \frac{sT_a}{A} \quad (1)$$

- T_n temps sur le nouveau système.
- T_a temps sur l'ancien système.
- s fraction du code concerné par l'amélioration.
- A Accélération obtenue par l'amélioration.

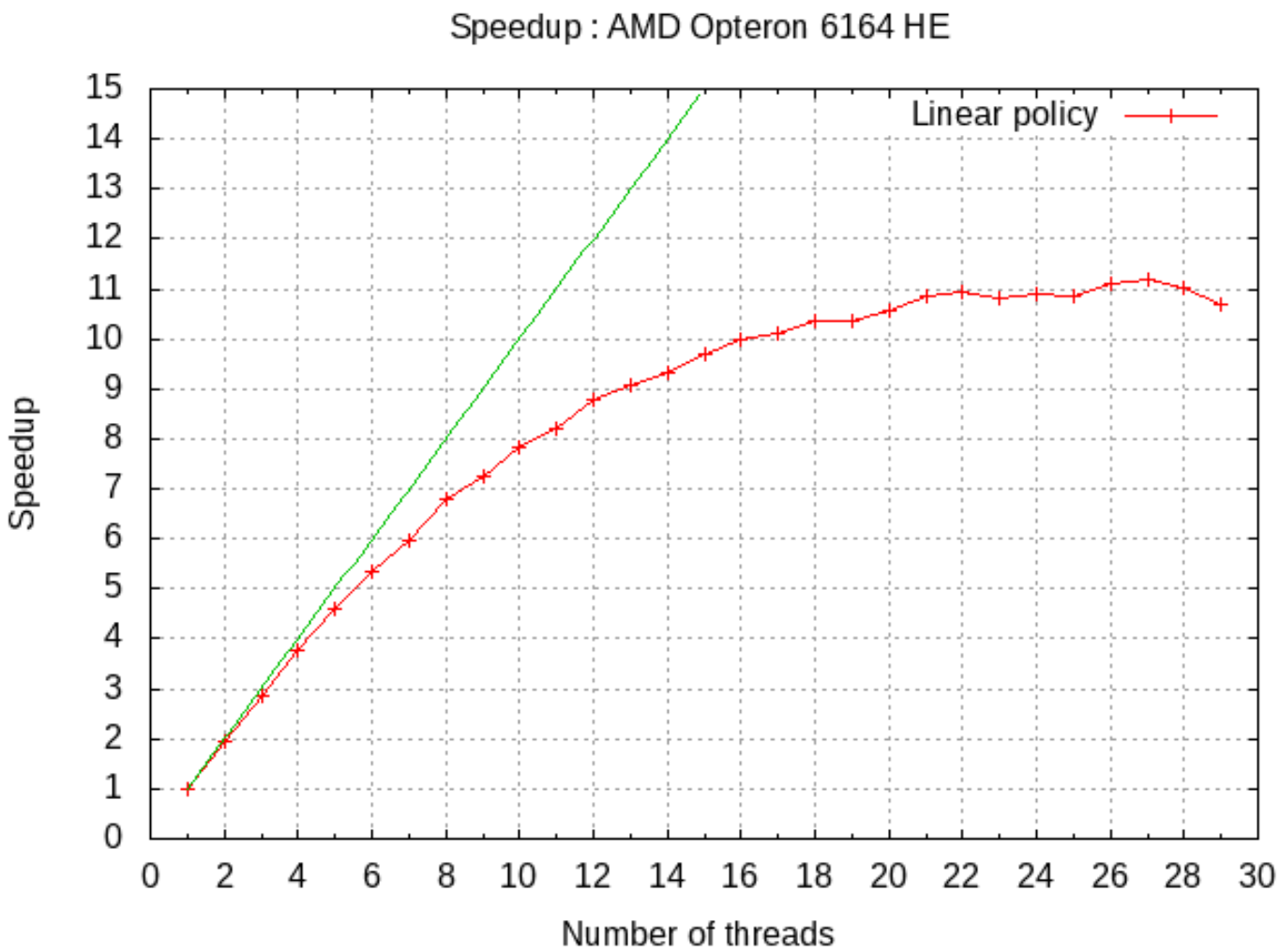
Performances



Performances



Performances



Et ensuite ?

Depuis le stage :

- Suivi de la mailing-list.
- Patch 2.1 : corrections mineures.
- Projet de modèle d'îles.

Questions

Merci pour votre attention ! :-)

Des questions ?