Analyse quantitative de programmes applicatifs à base de squelettes algorithmiques

Enhance (bourse EPSRC numéro GR/S21717/01)

Enhancing the Perf. Predictability of Grid Appli. with Patterns and Process Algebras

A. Benoit, M. Cole, S. Gilmore, J. Hillston

School of informatics http://homepages.inf.ed.ac.uk/abenoit1/

Introduction - programmes fonctionnels

- Langages de programmation applicatifs
  - beaucoup de vertus : systèmes robustes et descriptifs, facilité de structuration, sémantique rigoureuse, ...
  - tâches coûteuses $\rightarrow$ exécution parallèle
  - utilisation de programmation parallèle structurée (*squelettes algorithmiques*) pour conserver les propriétés de fiabilité

$\rightarrow$ intérêts *qualitatifs*, mais pas de garanties *quantitatives*

Introduction - Analyse quantitative

- Modélisation de l'application
  - contexte d'exécution hétérogène (grille d'ordinateurs)
  - dispo. et perf. des ressources imprévisible
  - modélisation probabiliste $\rightarrow$ processus stochastique
  - formalisme haut niveau: algèbres de processus stochastiques PEPA
- Résolution : obtention d'indices de performances
- Déboguage des modèles : utilisation d'un simulateur pas à pas en O'Caml

Plan de l'exposé

- Modélisation de squelettes algorithmiques
  - Squelettes, et la bibliothèque *eSkel*
  - Algèbre de processus pour l'évaluation des performances PEPA
  - Modèles de performance de squelettes
- Simulateur pas à pas de modèles
  - Description du simulateur
  - Exemple d'application
- Conclusions et perspectives

Squelettes algorithmiques

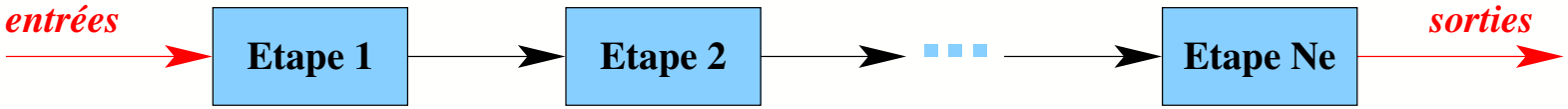
- Abstraction des schémas classiques de calcul parallèle et d'interaction
- Bibliothèque de fonctions paramétrées
- Squelettes algorithmiques
  - abstraction d'un schéma : *pipeline, donne, ...*
  - le programmeur invoque un ou plusieurs squelettes pour décrire la structure du programme
  - squelettes personnalisés pour l'application
  - bibliothèque *eSkel*

eSkel

- Murray Cole et Anne Benoit, 2002-2005
- Bibliothèque de fonctions C, basée sur MPI
- Adresse les problèmes soulevés par la programmation à base de squelettes

http://homepages.inf.ed.ac.uk/abenoit1/eSkel

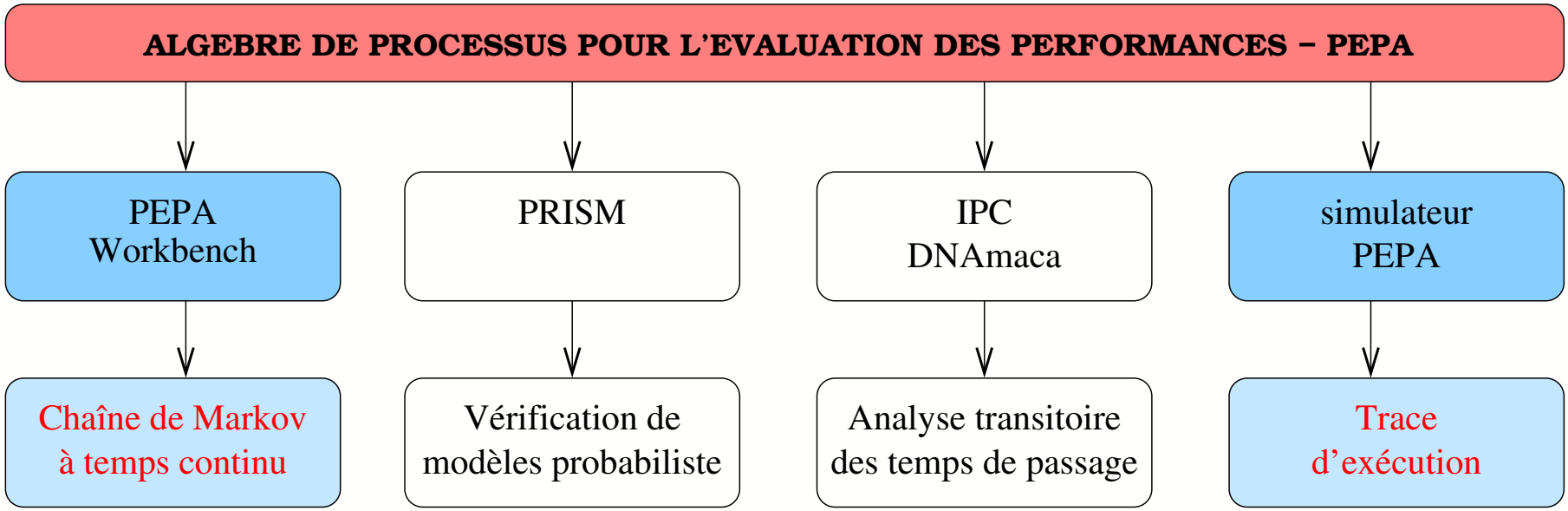
Pipeline - Principe du squelette



- N_e *étapes* travaillent sur une séquence d'entrées (*inputs*) pour créer une séquence de sorties (*outputs*)
- Toutes les données passent à travers chaque étape dans le même ordre
- L'activité interne d'une étape peut être parallèle, mais ceci est transparent à notre modèle

Évaluation des performances

- Algèbres de processus pour l'évaluation des performances PEPA [Hillston96]
- Outils pour PEPA



Pipeline - Modèle

- Mapping de l'application sur les ressources informatiques: le réseau et les processeurs
- Modèle de l'application: indépendant des ressources
- Modèle du réseau: information sur l'efficacité des connections réseau entre couples de processeurs
- Modèle des processeurs: information sur la puissance des processeurs

Pipeline - Modèle de l'application

- 1 composant PEPA par étape du Pipeline $(i = 1..N_e)$

  $$Etape_i \overset{def}{=} (tr_i, \top).(traite_i, \top).(tr_{i+1}, \top).Etape_i$$

- Séquentiel: obtient une donnée (transfert) (tr_i), la traite $(traite_i)$, transmet le résultat à l'étape suivante (tr_{i+1})
- Taux non spécifiés $(\top)$: déterminés par les ressources
- Pipeline = coopération des différentes étapes

  $$Pipeline \overset{def}{=} Etape_1 \underset{\{tr_2\}}{\bowtie} Etape_2 \underset{\{tr_3\}}{\bowtie} \ldots \underset{\{tr_{N_e}\}}{\bowtie} Etape_{N_e}$$

- Cas limites:
  - tr_1: arrivée d'une donnée dans l'application
  - tr_{N_e+1}: transfert d'un résultat hors du Pipeline

Pipeline - Modèle du réseau

- Affecte les taux λ_i aux activités de transfert de données entre étapes tr_i $(i = 1..N_e + 1)$

$$Reseau \stackrel{def}{=} (tr_1, \lambda_1).Reseau + \ldots + (tr_{N_e+1}, \lambda_{N_e+1}).Reseau$$

- λ_i représente la connection entre le proc. j_{i-1} hébergeant l'étape $i-1$ et le proc. j_i hébergeant l'étape i
- Cas limites:
  - j_0 est le processeur fournissant les données au Pipeline
  - j_{N_e+1} est là où l'on désire transmettre les résultats

Pipeline - Modèle des processeurs

- Modèle des processeurs: L'application tourne sur un ensemble de N_p processeurs
- Taux μ_i de l'activité $traite_i$ $(i = 1..N_s)$: charge du processeur, et autres indices de performance
- Une étape par processeur $(N_p = N_e \ ; \ i = 1..N_p)$:

  $$Proc_i \stackrel{def}{=} (traite_i, \mu_i).Proc_i$$

- Plusieurs étapes par processeur:

  $$Proc_1 \stackrel{def}{=} (traite_1, \mu_1).Proc_1 + (traite_2, \mu_2).Proc_1$$

- Ensemble de processeurs: composition parallèle

  $$Processeurs \stackrel{def}{=} Proc_1 || Proc_2 || ... || Proc_{N_p}$$

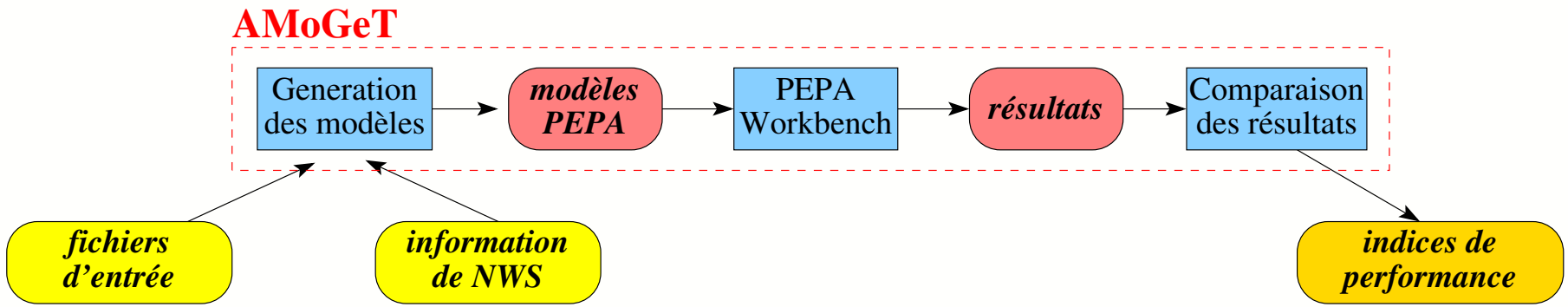
Pipeline - Modèle final

- Le modèle final est le mapping des étapes sur les processeurs et le réseau, en utilisant l'opérateur de coopération
- $L_p = \{traite_1, ..., traite_{N_e}\}$ synchronise $Pipeline$ et $Processeurs$
- $L_r = \{tr_1, ..., tr_{N_e+1}\}$ synchronise $Pipeline$ et $Reseau$

$$Mapping \stackrel{def}{=} Reseau \underset{L_r}{\bowtie} Pipeline \underset{L_p}{\bowtie} Processeurs$$

AMoGeT

- AMoGeT: Automatic Model Generation Tool
- Composant d'analyse générique



Plan de l'exposé

- Modélisation de squelettes algorithmiques
  - Squelettes, et la bibliothèque *eSkel*
  - Algèbre de processus pour l'évaluation des performances PEPA
  - Modèles de performance de squelettes
- Simulateur pas à pas de modèles
  - Description du simulateur
  - Exemple d'application
- Conclusions et perspectives

Simulateur - motivation

- Modèles de haut niveau
  - pas forcément correct, comme un programme
  - erreurs triviales, identifiées lors de la génération de l'espace d'états du modèle
  - erreurs difficiles à détecter

$\rightarrow$ *simulateur de modèles PEPA*

- évoluer pas à pas dans l'espace d'états
- permet de rechercher des erreurs de modélisation

Simulateur - description

- Implanté en O'Caml
- Utilise la boucle de haut niveau de O'Caml pour passer des commandes
  - évoluer pas à pas
  - choisir des activités à suivre
  - mettre des points d'arrêts sur des composants ou des activités
- Code lié au modèle que l'on étudie, permettant une vérification rigoureuse des types

Simulateur - exemple

- Modèle considéré : pipeline à 2 étapes
- Définition des identificateurs, activités et taux (dépendant du modèle)

```
type identificateur = Etape1 | Etape2
  | Reseau | Proc1 | Proc2;;

type activite = Tau of activite
  | Transfert1 | Transfert2 | Transfert3
  | Traite1 | Traite2;;

type taux = Top | Const of float | Lambda | Mu1 | Mu2;;
```

Simulateur - exemple

- Type composant, fidèle à la sémantique de PEPA (indépendant du modèle)

```
type composant  =
      Prefixe of ((activite * taux) * composant)
    | Choix   of composant * composant
    | Coop    of composant * activite list * composant
    | Hiding  of composant * activite list
    | Var     of identificateur;;
```

Simulateur - exemple

- Définition du modèle: fonction définissant chaque identificateur

```
let definition = function
| Etape1 -> Prefixe((Transfert1, Top),
  Prefixe((Traite1,Top), Prefixe((Tr2,Top),Var Etape1)))
| Etape2 -> Prefixe((Transfert2, Top),
  Prefixe((Traite2,Top), Prefixe((Tr3,Top),Var Etape2)))
| Reseau -> Choix(Choix(Prefixe((Tr1,Lambda),Var Reseau),
  Prefixe ((Tr2, Lambda),  Var Reseau)),
  Prefixe ((Tr3, Lambda),  Var Reseau))
| Proc1  -> Prefixe ((Traite1,Mu1), Var Proc1)
| Proc2  -> Prefixe ((Traite2,Mu2), Var Proc2);;
```

Simulateur - exemple

- définitions et fonctions permettant l'affichage des composants
- fonction qui retourne la liste des dérivatives d'un composant

```
val derive_un_pas:
 composant -> ((activite * taux) * composant) list = <fun>
```

Simulateur - exemple

- définition de l'application et utilisation du simulateur

```
let pipeline =
  Coop(Coop(Var Proc1,[],Var Proc2), [Traite1;Traite2],
  Coop(Coop(Var Etape1,[Tr2],Var Etape2),[Tr1;Tr2;Tr3],
       Var Reseau));;

# affiche_derivatives (derive_un_pas pipeline);;
-(transfert1, lambda)->((Proc1<>Proc2) <traite1,traite2>
(((traite1,top).(tr2,top).Etape1 <transfert2> Etape2)
<transfert1, transfert2, transfert3> Reseau));
- : unit = ()
```

Plan de l'exposé

- Modélisation de squelettes algorithmiques
  - Squelettes, et la bibliothèque *eSkel*
  - Algèbre de processus pour l'évaluation des performances PEPA
  - Modèles de performance de squelettes
- Simulateur pas à pas de modèles
  - Description du simulateur
  - Exemple d'application
- Conclusions et perspectives

Conclusions

- Intérêt de la programmation parallèle structurée
  - Notion de squelette algorithmique
  - Présentation de la bibliothèque *eSkel*
- Évaluation des performances
  - Algèbres de processus PEPA
  - Outils de résolution
- Modèles PEPA de squelettes algorithmiques
- Simulateur pas à pas pour ces modèles

Perspectives

- *eSkel* : développement en cours
  - promouvoir le concept de squelettes
  - développement d'applications avec *eSkel*
- Projet Enhance : modèles de squelettes
  - intégration des modèles dans *eSkel*
  - comparaison prédiction/mesures de performance
- Simulateur pas à pas
  - ajouter des nouvelles fonctionnalités
  - génération automatique en fonction du modèle

Merci pour votre attention!

Des questions?