

# Evaluation des performances de programmes parallèles haut niveau 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



<http://graal.ens-lyon.fr/~abenoit>

# Introduction - Contexte

- Programmes parallèles dans un contexte hétérogène
  - tournent sur un ensemble distribués d'ordinateurs
  - dispo. et perf. des ressources imprévisible
  - problèmes de scheduling/rescheduling
- Programmation parallèle haut-niveau
  - bibliothèque de squelettes (schémas parallèles)
  - de nombreuses appli. utilisent ces squelettes
  - modularité → simplicité d'utilisation
  - Edinburgh Skeleton Library eSkel (MPI) [Cole02]

# Introduction - Evaluation des performances

- Utilisation d'un squelette donné:  
information sur les dépendances impliquées
  - Modélisation à l'aide d'**algèbres de processus stochastiques**
    - inclut les aspects d'incertitude propre aux grilles
    - prise en compte dynamique des performances des ressources, processus de modélisation auto.
- permet de meilleures décisions de scheduling, et un éventuel **rescheduling** des applications
- *Amélioration des performances des programmes parallèles*

# Plan de l'exposé

- La bibliothèque de squelettes *eSkel*

*et comparaison avec d'autres approches*

- Motivation et concepts généraux

- Les squelettes de *eSkel*

- Utilisation de *eSkel*

- Modèles de performance de squelettes

- Le modèle du Pipeline

- AMoGeT (Automatic Model Generation Tool)

- Résultats numériques

- Conclusions et perspectives

- Concept de **squelettes algorithmiques** largement motivé dans la littérature
- *eSkel*
  - **Murray Cole**, 2002
  - Bibliothèque de fonctions C, basée sur MPI
  - Adresse les problèmes soulevés par la programmation à base de squelettes



## eSkel 2

- Murray Cole et Anne Benoit, 2004-2005
- Nouvelle interface et implémentation
- Plus de concepts adressés pour plus de flexibilité

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

- Mode de **composition** (*nesting*)
    - définit comment composer plusieurs squelettes
  - Mode d'**interaction**
    - définit les interactions entre les différentes parties d'un même squelette, et entre différents squelettes
  - Mode de **données**
    - lié aux autres concepts, définit comment les données sont prises en charge
- *Comment nous adressons ces concepts (eSkel)?*  
*Et comparaison avec d'autres approches*

- Peut être **transitoire** or **persistant**
- Composition transitoire (*transient nesting*)
  - une activité invoque un squelette
  - ce squelette contient ou crée ses propres données
- Composition persistante (*persistent nesting*)
  - le squelette est invoqué une fois
  - il récupère les données directement depuis le squelette “père” (qui l’a créé)
- Lié au mode de données (détaillé plus tard)

- Un **arbre des appels** est construit lors de la première interaction
- Représente la structure des squelettes composés de façon **persistante**
  - recherche dans l'arbre pour trouver les partenaires d'interaction
- Squelettes composés **transitoirement**
  - n'apparaissent pas dans l'arbre principal
  - créés dynamiquement, durée de vie limitée
  - sous-arbre créé dynamiquement à l'invocation

- Langage de Programmation Parallèle de Pise **P3L**
  - toutes les compositions sont **persistantes**
  - définies dans la couche P3L
  - clairement séparées du code séquentiel définissant les activités
- Bibliothèques P3L (**Lithium, SKELib**)
  - concept de composition transitoire : pas adressé explicitement
  - pas interdit mais pas supporté
- **ASSIST**: pas de composition dans ce sens

- **Implicite**

- une activité n'a pas de contrôle sur ses interactions
- fonction prenant une donnée en entrée et retournant une donnée

- **Explicite**

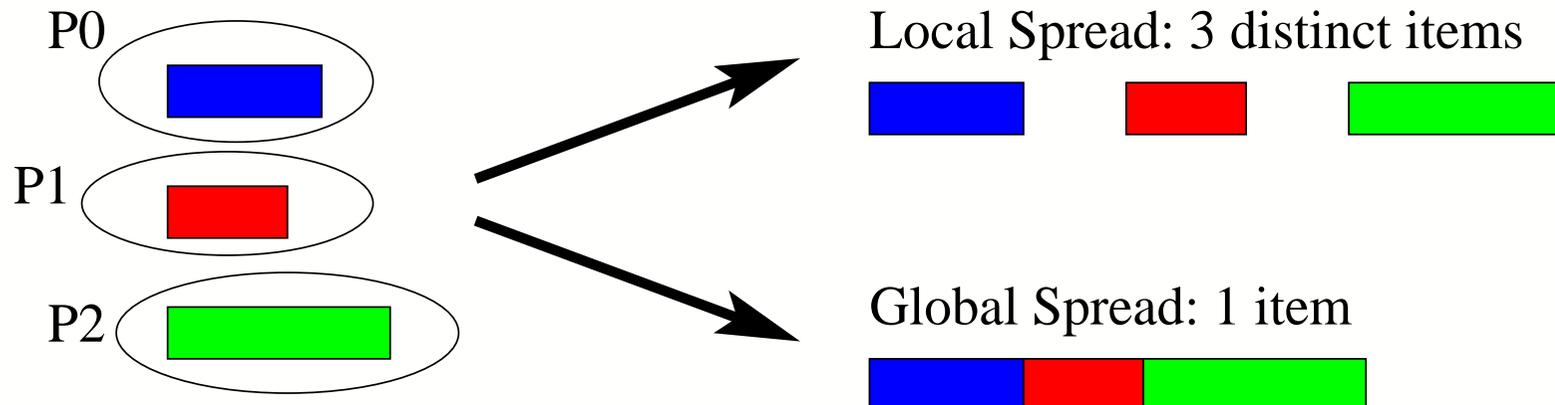
- interactions déclenchées dans le code de l'activité
- appels aux fonctions génériques `Take` et `Give`

- Mode **dévoué** (*devolved*) pour les squelettes composés: un squelette père peut utiliser le mode d'interaction du squelette fils composé

- **P3L** et les bibliothèques liées
  - interactions à travers un flot de données
  - défini implicitement par le squelette
- **ASSIST**
  - plus de flexibilité
  - interactions implicites ou explicites possibles

- Lié aux concepts précédents
- Modes Buffer / Stream
  - BUF: données dans un **buffer** (*composition transitoire*)
  - STRM: les données arrivent dans le squelette directement depuis les activités d'un squelette "père" (*composition persistante*)
- *eSkel Data Model* **eDM**
  - sémantique des interactions
  - unité de transfert : *molecule eDM*

- *molecule* eDM: collection d'*atomes* eDM
- **Type**: defini à l'aide des types MPI standards
- atome eDM: diffusion (*spread*) **locale** ou **globale**



# eSkel - Squelettes: brève description

- **Pipeline & Farm**: squelettes classiques, définis de façon générique
- **Deal**: similaire à Farm, sauf que les tâches sont distribuées de façon cyclique aux travailleurs
- **HaloSwap**: tableau 1-D d'activités qui, successivement, (1) échangent des données avec le voisin immédiat, (2) traitent les données localement, (3) décident collectivement si l'on doit procéder à une nouvelle itération
- **Butterfly**: classe d'algorithmes *diviser pour régner*

- Les squelettes sont classifiés en :
  - **parallélisme de tâche**: comm. dynamique entre processus pour distribuer le travail – *pipeline, farm*
  - **parallélisme de données**: travail sur une structure de données distribuée – *map, fold*
  - **squelettes de contrôle**: modules séquentiels et itération de squelettes – *seq, loop*
- eSkel: seulement squelettes pour le **paral. de tâches**
  - parallélisme de données: utilisation de *eDM*
  - contrôle exprimé directement dans le code C/MPI

## ● eSkel:

- interface pas évidente
- basé sur MPI, l'utilisateur doit connaître MPI
- structuration de **code MPI parallèle**

## ● P3L:

- plus facile à utiliser, structure simple
- moins de flexibilité, structuration de **code séquentiel**
- paral. de tâches/données et squelettes de contrôle
- pipeline 3-étapes: (créé données, traite, collecte résultats)

# eSkel - Interface: Pipeline

```
void Pipeline (int ns, Imode_t imode[], eSkel_molecule_t *
(*stages[])(eSkel_molecule_t *), int col, Dmode_t dmode, spread_t
spr[], MPI_Datatype ty[], void *in, int inlen, int inmul, void
*out, int outlen, int *outmul, int outbuffsz, MPI_Comm comm);
```

- information générale sur le pipeline (ns, ...)
- précise les modes: mode d'interaction (**imode**); mode de données (**dmode**), spread (**spr**) et type (**ty**)
- information relative au **buffer d'entrées**
- information relative au **buffer de sorties**

# eSkel - Interface: Deal

```
void Deal (int nw, Imode_t imode, eSkel_molecule_t *worker
(eSkel_molecule_t *), int col, Dmode_t dmode, void *in, int inlen,
int inmul, spread_t inspr, MPI_Datatype inty, void *out, int
outlen, int *outmul, spread_t outspr, MPI_Datatype outty, int
outbuffsz, MPI_Comm comm);
```

- information générale sur le deal (nw, ...)
- précise les modes: mode d'interaction (**imode**) et mode de données (**dmode**)
- information relative au **buffer d'entrées**
- information relative au **buffer de sorties**

- Programme C/MPI appelant les **fonctions squelettes**
- Besoin de faire attention aux **paramètres**
- Définition des squelettes composés, des travailleurs, ... à l'aide de **fonctions C/MPI standards**
- Seuls **Pipeline** et **Deal** sont implémentés pour l'instant dans eSkel2-0.1

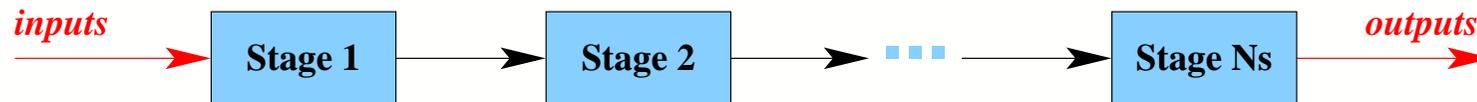
- Installation d'une **implémentation MPI thread-safe**
  - Los Alamos MPI <http://public.lanl.gov/lampi>
  - Installation des exécutables et bibliothèques
- Installation et test de **eSkel**
  - <http://homepages.inf.ed.ac.uk/abenoit1/eSkel>
  - récupération de la dernière version
  - lancement à l'aide d'une commande du style

```
mpirun -ssh -threads -H host1,host2,host3 -np 3,5,2 ./exemple
```

# Plan de l'exposé

- La bibliothèque de squelettes *eSkel*
  - Motivation et concepts généraux
  - Les squelettes de *eSkel*
  - Utilisation de *eSkel*
- Modèles de performance de squelettes
  - Le modèle du Pipeline
  - AMoGeT (Automatic Model Generation Tool)
  - Résultats numériques
- Conclusions et perspectives

# 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

# Pipeline - Modèle

- Le modèle est exprimé à l'aide d'algèbres de processus pour l'évaluation des performances  
(*Performance Evaluation Process Algebra*)  
**PEPA** [Hillston96]
- Mapping de l'**application** sur les ressources informatiques: le **réseau** et les **processeurs**

# Pipeline - Modèle de l'application

- **Modèle de l'application:** indépendant des ressources

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

$$Etape_i \stackrel{def}{=} (move_i, \top).(traite_i, \top).(move_{i+1}, \top).Etape_i$$

- Séquentiel: obtient une donnée ( $move_i$ ), la traite ( $traite_i$ ), transmet le résultat à l'étape suivante ( $move_{i+1}$ )

- Taux non spécifiés ( $\top$ ): déterminés par les ressources

- Pipeline = coopération des différentes étapes

$$Pipeline \stackrel{def}{=} Etape_1 \underset{\{move_2\}}{\bowtie} Etape_2 \underset{\{move_3\}}{\bowtie} \dots \underset{\{move_{N_e}\}}{\bowtie} Etape_{N_e}$$

- $move_1$ : arrivée d'une donnée dans l'application

$move_{N_e+1}$ : transfert d'un résultat hors du Pipeline

# Pipeline - Modèle du réseau

- **Modèle du réseau**: information sur l'efficacité des connections réseau entre couples de processeurs
- Affecte les taux  $\lambda_i$  aux activités  $move_i$  ( $i = 1..N_e + 1$ )  
$$Reseau \stackrel{def}{=} (move_1, \lambda_1).Reseau + \dots$$
$$+ (move_{N_e+1}, \lambda_{N_e+1}).Reseau$$
- $\lambda_i$  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  $\mu_i$  de l'activité  $traite_i$  ( $i = 1..N_e$ ): 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  
$$Processeurs \stackrel{def}{=} Proc_1 || Proc_2 || \dots || Proc_{N_p}$$

# Pipeline - Modèle final

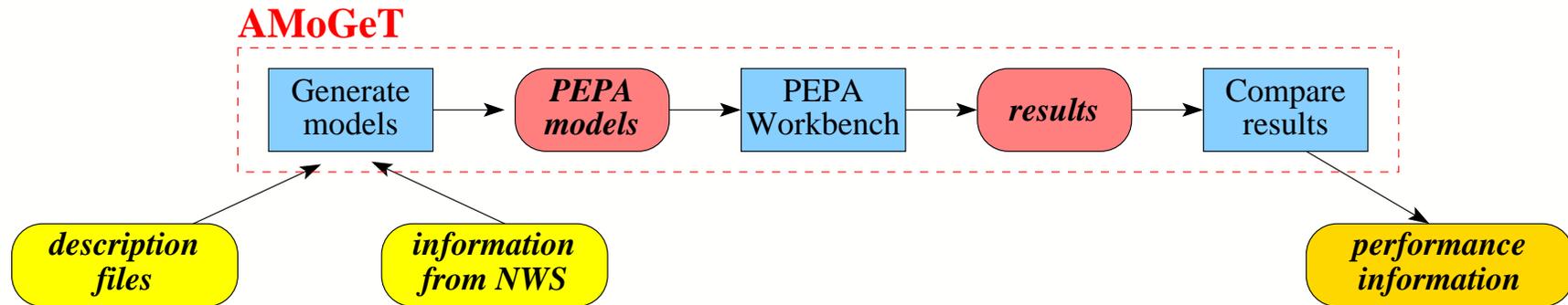
- Le modèle final est le mapping des **étapes** sur les **processeurs** et le **réseau**, en utilisant l'opérateur de coopération

- $L_t = \{traite_1, \dots, traite_{N_e}\}$  synchronise  
*Pipeline et Processeurs*

- $L_m = \{move_1, \dots, move_{N_e+1}\}$  synchronise  
*Pipeline et Réseau*

*Mapping*  $\stackrel{\text{def}}{=} \text{Réseau} \underset{L_m}{\bowtie} \text{Pipeline} \underset{L_t}{\bowtie} \text{Processeurs}$

# AMoGeT - Vue générale



- AMoGeT: **A**utomatic **M**odel **G**eneration **T**ool
- Composant d'analyse générique
- Rôle ultime: composant intégré à un scheduler et rescheduleur temps réel

# AMoGeT - Fichiers d'entrée (1)

- Préciser les noms des processeurs
  - fichier `hosts.txt`: liste des adresses IP
  - rang  $i$  dans la liste  $\rightarrow$  processeur  $i$
  - le processeur 1 est le *processeur de référence*

wellogy.inf.ed.ac.uk

bw240n01.inf.ed.ac.uk

bw240n02.inf.ed.ac.uk

bicephale.univ-paris12.fr

- Décrit l'application *exemple*
  - fichier `exemple.des`
  - étapes du Pipeline: nombre d'étapes  $N_e$  et temps  $tr_i$  (sec) requis pour produire une sortie pour chaque étape  $i = 1..N_e$  sur le processeur de référence  
`nbetape= $N_e$ ; tr1=10; tr2=2; ...`
  - mappings des étapes sur les processeurs: **location des entrées**, **le processeur hébergeant chaque étape**, et **la location des sorties**.  
`mappings=[1, (1,2,3), 1], [1, (1,1,1), 1];`

- The Network Weather Service (NWS) [Wolski99]
  - Prédiction dynamique des performances du réseau et des ressources
  - Quelques scripts à lancer sur les noeuds étudiés
  - Information que l'on utilise:
    - $av_i$  - **fraction de CPU disponible** pour une tâche qui commence sur le processeur  $i$
    - $la_{i,j}$  - **latence** (en ms) d'une communication du processeur  $i$  vers le processeur  $j$
- $cpu_i$  - **fréquence** du processeur  $i$  en MHz (/proc/cpuinfo)

- Un modèle de Pipeline par mapping
- Problème: calcul des taux
  - Etape  $i$  ( $i = 1..N_e$ ) sur le processeur  $j$  (et un total de  $nb_j$  étapes sur ce processeur):

$$\mu_i = \frac{av_j}{nb_j} \times \frac{cpu_j}{cpu_1} \times \frac{1}{tr_i}$$

- Taux  $\lambda_i$  ( $i = 1..N_e + 1$ ): lien réseau entre le processeur  $j_{i-1}$  hébergeant l'étape  $i - 1$  et le processeur  $j_i$  (étape  $i$ ):  $\lambda_s = 10^3 / la_{j_{i-1}, j_i}$   
(limites: étape 0 = entrée et étape  $N_e + 1$  = sortie)

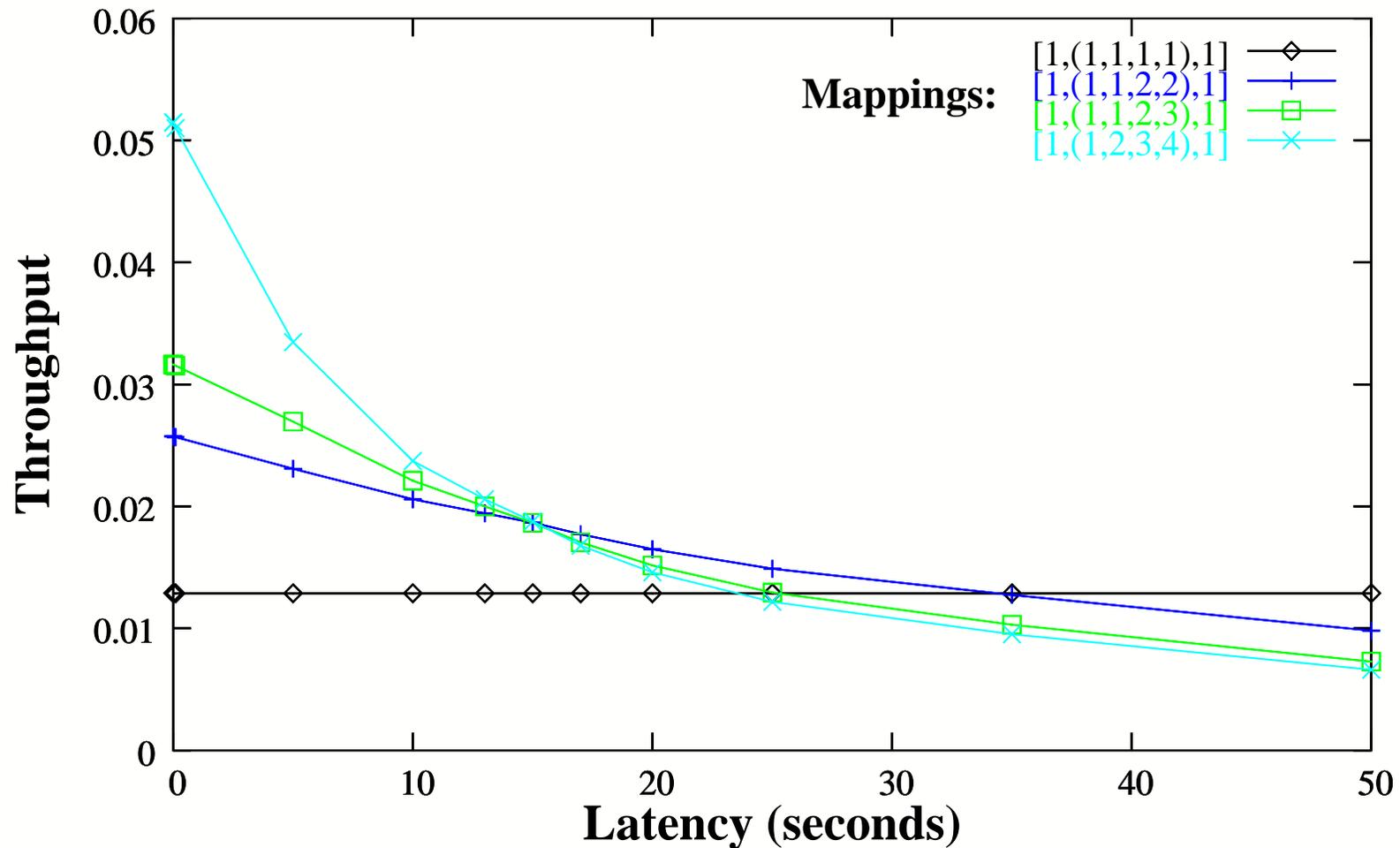
- Résultats numériques obtenus à l'aide du PEPA Workbench [Gilmore94]
- Résultat de performance: **débit** des activités  $move_i =$  débit de l'application
- Résultat obtenu par une simple ligne de commande, tous les résultats sont sauvés dans un même fichier
- Quel mapping produit le meilleur débit?
- Utiliser ce mapping pour faire tourner l'application

# Résultats numériques

- **Exemple 1**: Pipeline 3-étapes, jusqu'à 3 processeurs
- 27 états, 51 transitions → moins d'une sec. pour résoudre
- latence des com 0.001 sec; tous processeurs/étapes sont identiques; temps requis pour compléter une étape:  $t$
- $\mu_i = 1/(t \times nb_j)$  ( $nb_j$ : nb d'étapes sur le proc  $j$ )
- Mappings comparés: tous les mappings avec l'étape 1 sur le proc 1 (mappings  $[1, (1, *, *), *]$ )
  - $t = 0.1$ : mappings optimaux (1,2,3) et (1,3,2) avec un débit de 5.64
  - $t = 0.2$ : mêmes mappings optimaux (1 étape sur chaque proc), mais débit divisé par 2 (2.82)

# Résultats numériques

- **Exemple 2:** Pipeline 4-étapes, jusqu'à 4 proc,  $t = 10$  sec.



# Plan de l'exposé

- La bibliothèque de squelettes *eSkel*
  - Motivation et concepts généraux
  - Les squelettes de *eSkel*
  - Utilisation de *eSkel*
- Modèles de performance de squelettes
  - Le modèle du Pipeline
  - AMoGeT (Automatic Model Generation Tool)
  - Résultats numériques
- Conclusions et perspectives

# Conclusions - Partie 1

- Pourquoi utiliser un modèle de **programmation parallèle structurée**?  
(*exposé de Murray Cole à EuroPar 2004, ...*)
- Présentation de la **bibliothèque eSkel**
  - Concepts à la base de la bibliothèque
  - Comment nous adressons ces concepts
- Comparaison avec le langage **P3L** et ses concepts

# Perspectives - Partie 1

- **eSkel**: développement en cours
  - Encore plusieurs squelettes à implémenter
  - L'interface peut être simplifiée et plus facile à utiliser, mode débogage
- **Validation** de ces concepts
  - Développer une vraie application avec *eSkel*
  - Promouvoir le concept de squelettes
- **Comparaison** avec d'autres approches
  - P3L, ASSIST, ... / Kuchen / Eden

# Conclusions - Partie 2

- Utilisation de **squelettes** et de **modèles de performance** pour améliorer les performances de programmes parallèles haut-niveau
  - squelettes **Pipeline** et **Deal**
  - **Outil AMoGeT**: permet d'automatiser le processus pour obtenir facilement des résultats
  - **Modèles**: permettent de choisir le mapping qui produit le meilleur débit pour l'application
  - Utilisation du **Network Weather Service** pour obtenir des modèles réalistes

# Perspectives - Partie 2

- Fournir des détails sur le **temps** requis par AMoGeT pour conforter son utilité
- Etendre le travail à d'**autres squelettes**
- Expériences sur une **application réelle** sur une vraie **grille de calcul hétérogène**
- Intégrer le tout dans une **interface graphique** pour aider la conception d'applications avec *eSkel*

*Premier cas d'étude → on a le potentiel pour améliorer les performances de programmes parallèles haut-niveau en utilisant des squelettes et des algèbres de processus*



Des questions?

*... et des infos sur les travaux avec Pise? ...*

# Automatic mapping of ASSIST applications using process algebra



A. Benoit and M. Aldinucci



ISTITUTO DI SCIENZA E TECNOLOGIE  
DELL'INFORMAZIONE "A. FAEDO"

Enhance project (funded by the EPSRC, grant number GR/S21717/01)

Italian MIUR FIRB Grid.it project (RBNE01KNFP)

CoreGRID - Network of Excellence

# Introduction - Context of the work

- **Grid computing**
  - widely distributed collection of computers
  - resource availability and performance unpredictable
  - mapping, scheduling, dynamic rescheduling issues
- **Low-level** approach: unacceptable for the programmer
- **High-level, layered** programming model
  - ASSIST, eSkel, GrADS, ProActive, Ibis, ...
  - grid specific efforts done by the environment
  - programmers: only application specific code

# Introduction - Motivations

- **ASSIST** environment (A Software System based on Integrated Skeleton Technology)
  - **autonomic** (self-optimisation) behaviour
  - relies on **run-time** application monitoring
  - not effective for application deployment
- The problem of **application mapping**
  - performance models with **stochastic process algebra**
  - **automated** modelling process
  - static analysis of the application

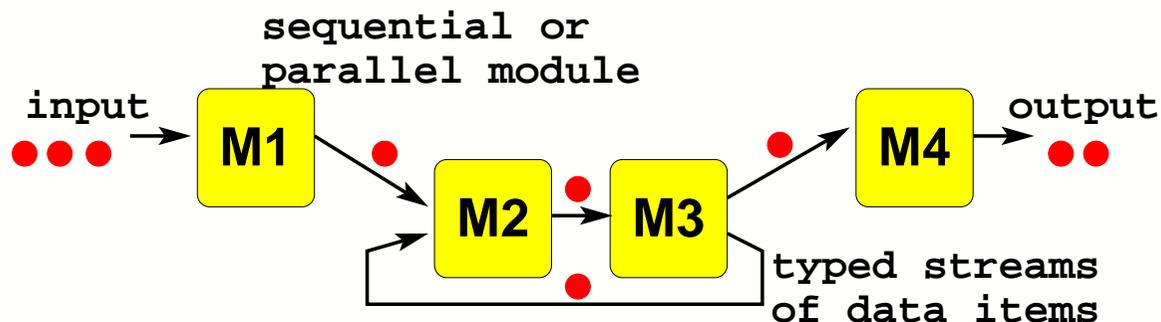
# Structure of the talk

---

- Introduction and motivation
- The ASSIST environment
- Performance models of ASSIST application
  - The C4.5 algorithm
  - PEPA models and automatic model generation
  - Performance results
- Conclusions and Perspectives

# The ASSIST environment

- A Software System based on Integrated Skeleton Technology
- Coordination language
  - express arbitrary graphs of modules
  - interconnection via typed streams of data
  - modules = sequential or parallel



# The ASSIST environment – Example

ASSIST code for the graph given as an example:

```
generic main() {
    stream task_t s1; stream task_t s2;
    stream task_t s3; stream task_t s4;
    M1 ( output_stream s1 );
    M2 ( input_stream s1,s2 output_stream s3 );
    M3 ( input_stream s3 output_stream s2,s4 );
    M4 ( input_stream s4 ); }
parmod M2 (input_stream task_t s1, task_t s2
           output_stream task_t s3 ) {
    topology none vp; ... }
```

# ASSIST – Run-time and autonomic features

- Graph of modules translated into a network of processes
- Processes devoted to application QoS control
  - module adaptation manager and appli. manager
  - dynamic monitoring of the application
- Middleware
  - services for co-allocation, staging and execution
  - able to deploy and launch applications
- Our concern: mapping of the application

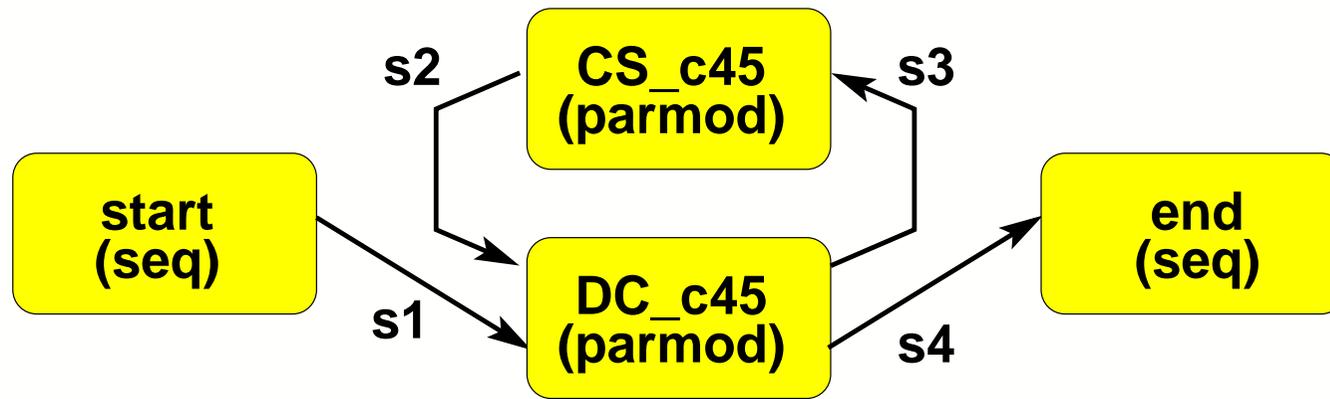
# Structure of the talk

---

- Introduction and motivation
- The ASSIST environment
- Performance models of ASSIST application
  - The C4.5 algorithm
  - PEPA models and automatic model generation
  - Performance results
- Conclusions and Perspectives

# The C4.5 application

- Data Mining C4.5 **classification** algorithm
  - **extraction** of information from data
  - non-trivial, implicit, previously unknown
  - decision trees
  - **Divide&Conquer** paradigm



# PEPA models – Introduction

- **Mapping** of the application
- Performance Evaluation Process Algebra  
**PEPA** [Hillston96]
  - aspect of uncertainty relative to the grid
  - automatic modelling process
  - standard Markov Chains techniques and solvers
- Our goal: decide which mapping of the application is the best
- Illustration on the **C4.5 application**

# PEPA models – The model

- ASSIST module == PEPA component
- ASSIST stream == PEPA synchronisation

- Example: DC module

$$DC1 \stackrel{def}{=} (s1, \top).DC2 + (s2, \top).DC2$$

$$DC2 \stackrel{def}{=} (pDC, \mu_{DC}).DC3$$

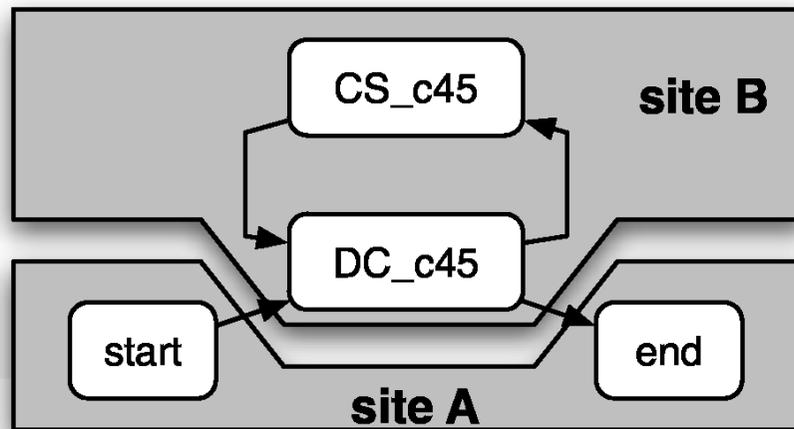
$$DC3 \stackrel{def}{=} (s3, \lambda_3).DC1 + (s4, \lambda_4).DC1$$

inputs from streams  $s1$  and  $s2$ ,  
outputs to streams  $s3$  and  $s4$

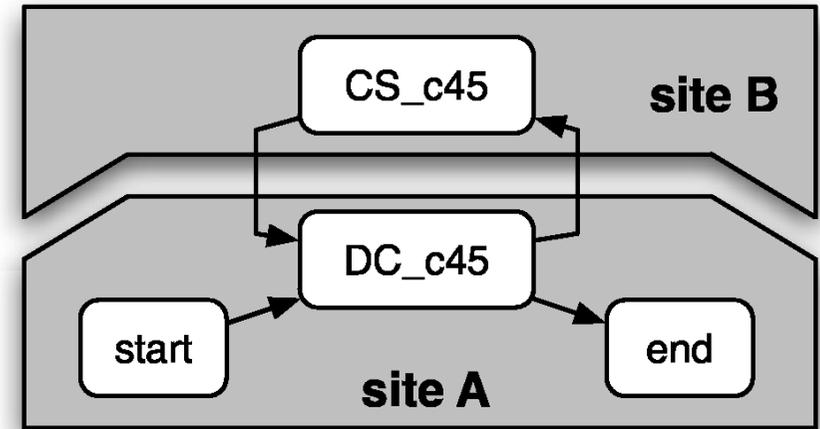
- Overall model:  $S1 \underset{s1}{\bowtie} DC1 \underset{s2, s3}{\bowtie} CS1 \underset{s4}{\bowtie} E1$

# PEPA models – Automatic model generation

- Information provided by the user in the ASSIST source code to allow an **automatic model generation**
- Two multi-site deployments



Case 1: Loop on the same cluster



Case 2: Loop on 2 clusters

# PEPA models – Computing the rates (1)

- Define the **rates** of the activities in the PEPA model
- Rates defined **arbitrarily**
- One **set of rates** per deployment schema
  - same **computation rates** in both schema
  - most of the computation done in **module CS**
  - rates = inverse of time (exponential distribution)

rate(**start**)=(100,100);

rate(**end**)=(100,100);

rate(**DC\_c45**)=(100,100);

rate(**CS\_c45**)=(1,1);

# PEPA models – Computing the rates (2)

- **Communication rates** differ, depending on the deployment schema
  - **outside the loop**, fast communications in case 1, slow communications in case 2:  
 $\text{rate}(s1)=(1,1000)$ ;  $\text{rate}(s4)=(1,1000)$ ;
  - **inside the loop**, fast communications in case 2, slow communications in case 1:  
 $\text{rate}(s2)=(1000,1)$ ;  $\text{rate}(s3)=(1000,1)$ ;
- **One model generated per deployment schema**
- Generation during a **precompilation** of the application

# Computing the performance results

- Models solved with the **PEPA workbench** [Gilmore94]
- **Steady-state probabilities** of the underlying Markov Chain
- **Simple models** (in the example, 36 states & 80 transitions)
- Resolution straightforward

# Performance results

Performance result	Percentage – case1	Percentage – case2
<i>moduleS</i>	0.49	52.14
<i>moduleDC</i>	1.23	52.24
<i>moduleCS</i>	74.10	10.60
<i>moduleE</i>	0.49	52.14
<i>s1</i>	49.37	5.21
<i>s2</i>	0.07	10.61
<i>s3</i>	0.07	10.61
<i>s4</i>	49.37	5.21

# Performance results analysis

- **First case:** loop grouped on the same site
  - time spent computing CS\_c45 module
  - communication time: sending data into the loop
- **Second case:** loop divided between two sites
  - lot of time spent in the non-computationally intensive modules
  - module CS\_c45 continuously waiting for data
- **Comparison of alternative mappings**

# Structure of the talk

---

- Introduction and motivation
- The ASSIST environment
- Performance models of ASSIST application
  - The C4.5 algorithm
  - PEPA models and automatic model generation
  - Performance results
- **Conclusions and Perspectives**

# Conclusions

- Automatic mapping of ASSIST applications
- Use of process algebra PEPA
- Case study: the C4.5 classification algorithm
- Addresses an important problem in grid application optimisation
- Complementary with the dynamic management of the ASSIST environment

# Future work

- New **on-going** work
  - **validate** the approach on more **complex applications**
  - obtain less trivial **results**
- **Modules considered as blocks**
  - model the **internal behaviour** of each module
  - integrate these **sub-models** in the global PEPA model
- **Promising results**

# Encore des travaux en cours...

- Discussion sur des techniques de scheduling pour ASSIST, inspirés de vos papiers sur la complexité des algorithmes de scheduling (papier “*scheduling divisible loads with return message*”)
- Techniques de scheduling off-line pour des “*farm*” dynamiques de composants (plateformes maître/esclave)

*Merci pour votre attention!*



**Des questions?**

*... et cette fois c'est vraiment fini!*