MOAIS

Kaapi dans Safescale

Vincent Danjean, Thierry Gautier, Samir Jafar, Jean-Louis Roch
Projet MOAIS [INRIA / CNRS / INPG / UJF]
moais.imag.fr **kaapi.gforge.inria.fr**

1. Une application Safescale sur Grid'5000
2. Kaapi : graphe dataflow, vol de travail, et adaptation
3. Tolérance aux pannes franches

Une application SafeScale sur Grid'5000 [V Danjean]

- Construction de boîtes cryptographiques [R Gillard]
  - Espace de recherche : Perm(F256) => 256!
  - Code "**paloDF.c**" : recherche statistique (calcul de minima)
- Parallélisation sur Athapascan/Kaapi et portage Grid'5000
  - **paloDF.C**: fichier principal en C++ : **370 lignes à l'origine, 680 maintenant**
    - Enrobage tâches de calcul dans des tâches "Athapascan/Kaapi"
    - Découpe récursive directe de l'espace exploré (parallélisme dynamique/vol de travail)
    - Flag de compilation pour compiler avec ou sans les modifications Kaapi
  - **mt19937ar.[hc]**: tirages aléatoires **280 lignes à l'origine, 300 maintenant**
    - Transformation thread-safe (suppression variables globales)
  - Re-vérification des boîtes intéressantes trouvées
    - Comment vérifier les autres ? (OK si attaque massive mais sinon …)
- Expériences menées sur Grid'5000:
  - Tests de fonctionnement sur les 3 clusters de Rennes en simultané
  - Cluster 100 bi-pro Nice : en 2h30, 686 boites intéressantes trrouvées
- Expériences prévues : mesures sur plus de procs, hétérogènes :
  - Utilisation tolérance aux pannes de Kaapi [Ingénieur SafeScale oct. 2006]
  - Récupération résultats partiels locaux : schéma récursif adaptatif

Modèle de programmation & vol de travail

Programmation parallèle haut niveau

- Description implicite du parallélisme
  - tâches et dépendances de données
  - graphe de flot de données
- Avantages :
  - Applications portables
  - Facilité la programmation parallèle
- Ex. : Cilk, Charm++, Menta, Satin, ATHAPASCAN/KAAPI

Ordonnancement

- Vol de travail
- Prouvé efficace en théorie avec des ressources hétérogènes [Bender'02,...]

Ordonnancement par vol de travail

Principe

- Chaque processeur gère localement la liste des tâches que lui-même a créées ordonnée selon l'ordre séquentiel
- Lorsqu'un processeur termine une tâche :
  - Soit sa liste contient des tâches prêtes ⇒ exécute la plus prioritaire suivant l'ordre séquentiel
  - Soit sa liste est vide ou ne contient pas de tâches prêtes ⇒ devient voleur et cherche à récupérer du travail sur les autres processeurs selon un ordre parallèle
- Vérifie la propriété glouton :
  - A tout instant où il existe une tâche prête mais non encore ordonnancée, tous les processeurs sont actifs

Algorithme distribué

- Processeur victime choisi au hasard

Modèle de coût : avec une grande probabilité, sur p proc. Identiques

- Temps d'exécution = $T_p \leq T_1/p + c_\infty T_\infty$

- nombre de requètes de vols = $N_{theft} \leq O(pT_\infty)$

Modèle de coût associé au vol de travail

Notations (cas processeurs identiques)

- T_s : la durée d'exécution de l'algorithme séquentiel
- T_1 : la durée d'exécution de l'algorithme parallèle sur 1 processeur
- T_∞ : la durée d'exécution du chemin critique
- T_p : la durée d'exécution de l'algorithme parallèle sur p processeurs

Théorème [BL97, GRCD98]

- le temps d'exécution T_p d'un programme avec l'algorithme de vol de travail sur p processeurs est majoré par

$$T_p \leq T_1/p + c_\infty T_\infty$$

- le nombre de vols réussis N_{theft} est majoré avec une grande probabilité par

$$N_{theft} \leq O(pT_\infty)$$

Implantation du vol de travail

Nécessite le calcul des tâches prêtes

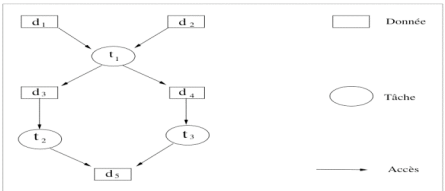
- Représentation abstraite des tâches et de leurs dépendances

$\Rightarrow$ Graphe de flot de données

Le graphe de flot de données associé à l'exécution d'une application : c'est le graphe $G = (V, E)$ tel que $V = V_t \cup V_d$

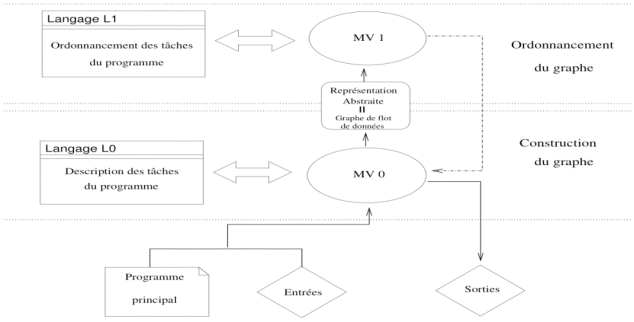
- V_t : tâches du programme, V_d : données partagées
- E : accès des tâches aux données partagées

$*$ Le graphe est dynamique (implicitement distribué)



Modèle d'exécution par graphe de flot de données

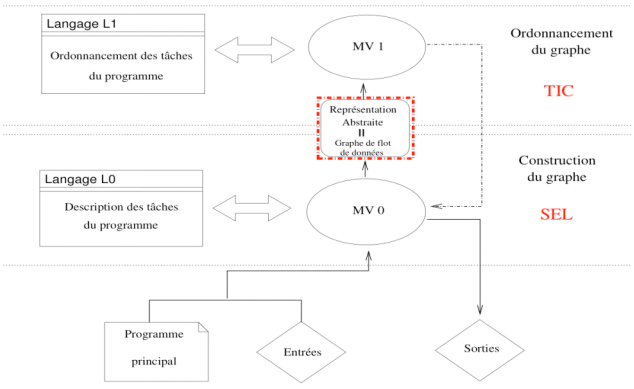
- Graphe dynamique et distribué
- Ordonnancement distribué par vol de travail



Exemple : l'intergiciel KAAPI

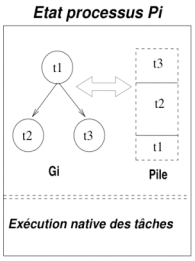
Modèle d'exécution & Sauvegarde/Reprise par graphe de flot de données

Proposition : deux protocoles pour capturer le graphe



Protocole TIC

Définition d'un point de reprise



- Un point de reprise concerne un processus
- Le point de reprise d'un processus p_i est sa partie G_i du graphe de flot de données G de l'application
  - Ses tâches et leurs paramètres

Différence importante

- Sauvegarde uniquement des tâches et de leurs paramètres
  - $\Rightarrow$ indépendant des plates-formes
- Ne sauvegarde pas l'état d'exécution des tâches
  - $\Rightarrow$ le contexte d'un processus dépend de la plate-forme

$\Rightarrow$ Sauvegarde avant ou après l'exécution d'une tâche

Analyse de complexité pour TIC

- Classe des programmes considérés : $T_1 \gg T_\infty$
- t_s : accès élémentaire aux support de stockage
- N_∞ : # maximum de tâches sur un plus long chemin du graphe ($N_\infty = O(T_\infty)$)

TIC dépend du nombre de vols réussis et période

$$T_P^{TIC} \leq T_p + [T_P^{TIC}/\tau + N_{theft}]\ f_{overhead}^{TIC}(N_\infty, t_s)$$

Coût reprise en pire cas

$T_{reprise} \leq O(N_\infty) + \tau +$ durée maximum d'exécution 1 tâche

Calcul perdu après une panne (pire cas)

Période de sauvegarde + le temps d'exécution d'une tâche

SEL : protocole par journalisation du graphe de flot de données

Principe de journalisation

Journalisation sur support stable de tous les événements de construction du graphe de flot de donneés :

- Créations / suppressions des tâches
- Créations / modifications / suppressions des données partagées

$\Rightarrow$ Nécessite l'hypothèse PWD

Propriété

Si les tâches d'une application parallèle vérifient :

H_1 Une tâche s'exécute jusqu'à la fin de son exécution sans synchronisation

H_2 L'exécution des tâches est déterministe

$\Rightarrow$ les processus exécutant cette application respectent l'hypothèse PWD

SEL : Reprise

Principe de reprise

Interprétation des événements du journal pour reconstruire le graphe de flot de données de toutes les tâches qui ne sont pas terminées

Problème

- L'exécution d'une tâche peut créer d'autres tâches

$\Rightarrow$ suite à une reprise une tâche peut être créée plusieurs fois

Solution

- Identification unique et reproductible de tous les nœuds du graphe
- Avant la création d'un nœud, on vérifie s'il existe déjà dans le journal

Analyse de complexité pour SEL

- Classe des programmes considérés : $T_1 \gg T_\infty$
- t_s accès élémentaire aux support de stockage

Coût de la journalisation dépend de la taille du graphe G

$$T_P^{SEL} \leq \frac{T_1^{SEL}}{p} + c_\infty T_\infty$$

$$T_1^{SEL} = T_1 + f_{overhead}^{SEL}(|G|, t_s)$$

Coût reprise en pire cas

$T_{reprise} = O(|G_i|)$ + durée maximum d'exécution 1 tâche
où G_i le graphe du processus défaillant

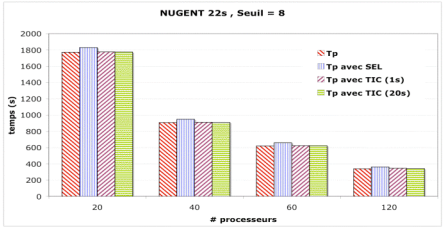
Calcul perdu après une panne (pire cas)

Au plus le calcul d'une tâche

Influence du nombre de processeurs

Coût de sauvegarde avec SEL et TIC

supports de stockage = # processeurs

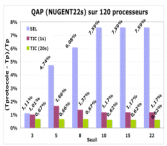


⇒ Surcoût faible
⇒ Passage à l'échelle : support de stockage distribué

Influence du grain de l'application

Surcoût de sauvegarde (SEL et TIC)
QAP (NUG22S) sur 120 processeur

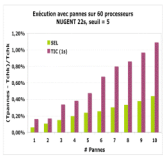
A grain fin le surcoût de SEL augmente



A gros grain le surcoût de TIC augmente

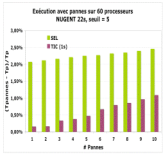
Influence du nombre de pannes

Surcoût reprise



Reprise : $SEL < TIC$

Surcoût total

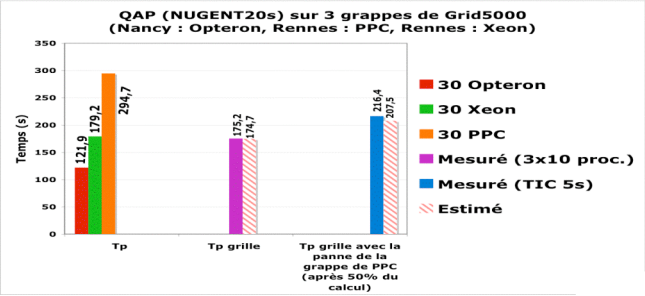


Surcoût total $< 2.5\%$

Dynamicité & hétérogénéité

Expérience sur 3 grappes hétérogènes de Grid5000

- Lancement de l'exécution sur 3 grappes : Grillon (Opteron), Paraci (Xeon) et Tartopom (PPC)
- Support de stockage : la grappe Idpot (Xeon, Grenoble), 3 supports
- Retrait de la grappe Tartopom après 50% de début de l'exécution



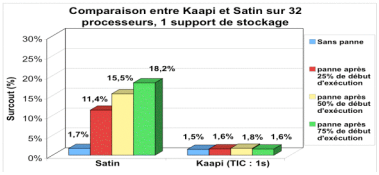
Conclusion tolérance défaillances

- **Kaapi tolère ajout et défaillance de machines**
  - Protocole TIC :
    - période à ajuster
  - Détecteur défaillances
    - signal erreurs +heartbeat
    - améliorable

Comparaison KAAPI/Satin

Surcoût relatif à l'exécution parallèle

- Application : UB_{Tree}
- Plate-forme (gdx) : 216 Opterons, Bi-processeurs, 2 GHz, mémoire 2 GB, Gigabit ethernet



Critères de comparaison	CoCheck	MPICH-V2	MPICH-CL	Satin	Athapascan / Kaapi
information sauvegardée	Image mémoire	Image mémoire	Image mémoire	Tâches orphelines	Dataflow graph
Multithreading	non	non	non	oui	Pile [TIC] / FullDag[SEL]
Hétérogénéité	non	non	non	oui	oui / oui
Remplacement de ressource défaillante	nouvelle ressource	nouvelle ressource	nouvelle ressource	pas besoin	Pas besoin
Reprise	globale	locale globle	globale	locale pas globale	Locale ou globale

Questions ?

- http://kaapi.gforge.inria.fr