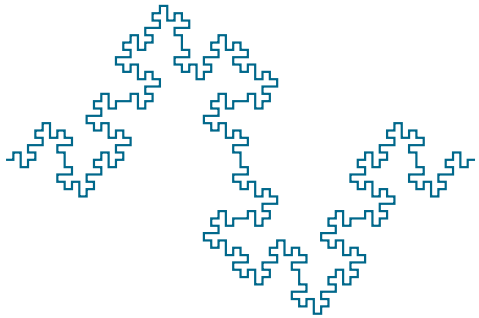
HLIN403 – Programmation Applicative

Programmation par flux de données (dataflow)

Christophe Dony – Annie Chateau
Université Montpellier – Faculté des Sciences



Introduction

Programmation par flux de données (dataflow)

Dataflow

Donnée : entité manipulée dans un programme, définie par un type.

Aujourd'hui, on parle de To de données comme on parlait de Mo il y a quelques années.

Stockages de plusieurs Po accessibles

Besoin de gestion efficace et rapide (traitements, données biologiques, physiques, ou encore données internet pour analyses commerciales et autres).

⇒ Besoin d'un paradigme qui combine l'utilisation des processeurs multicœurs et les clusters de nombreux nœuds avec le traitement et l'analyse des données.

Dataflow

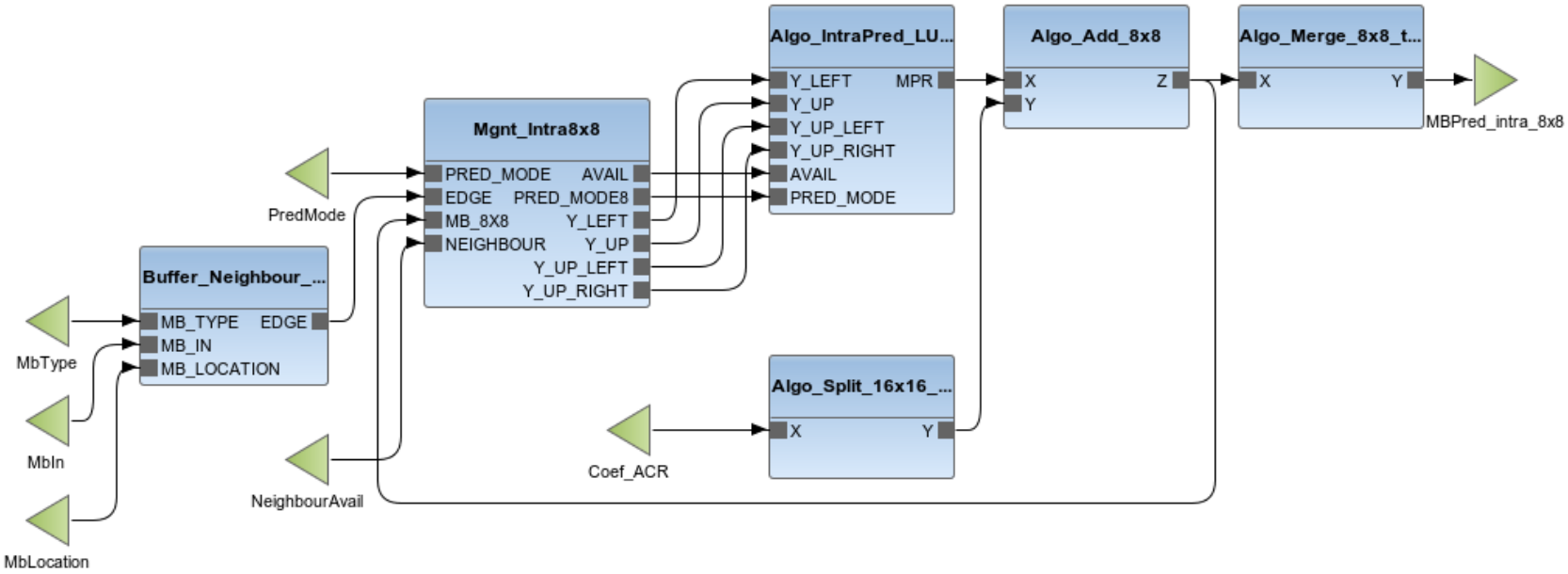
Le paradigme « flux de données » est constitué de noeuds dans un graphe orienté, reliés par des files d'attente.

Le graphe représente une application ou un programme et les noeuds représentent des fonctions qui doivent être appliquées aux données.

Les files d'attente transportent les données entre les fonctions connectées.

Dans la plupart des cas, chaque opérateur dans le graphe reçoit des données à partir de sa file d'attente d'entrée, applique sa fonction aux données, et délivre en sortie les résultats à sa file d'attente de sortie.

Dataflow


Algo_IntraPred_LU...
Y_LEFT
MPR
Y_UP
Y_UP_LEFT
Y_UP_RIGHT
AVAIL
PRED_MODE
Algo_Add_8x8
X
Z
Y
Algo_Merge_8x8_t...
X
Y
MBPred_intra_8x8
Mgnt_Intra8x8
PRED_MODE
AVAIL
EDGE
PRED_MODE8
MB_8X8
Y_LEFT
NEIGHBOUR
Y_UP
Y_UP_LEFT
Y_UP_RIGHT
PredMode
Buffer_Neighbour_...
MB_TYPE
EDGE
MB_IN
MB_LOCATION
MbType
MbIn
MbLocation
NeighbourAvail
Algo_Split_16x16_...
X
Y
Coef_ACR

Programmation par flux de données, principe de base

La composition d'un graphe de flux de données
= processus de création d'instances d'opérateurs et de leurs liaisons entre elles par « couture » de la sortie d'un opérateur à l'entrée d'un autre.

Souplesse dans la façon dont les utilisateurs relient les opérateurs pour créer des applications.

En pensant à chaque opérateur en termes de

- consommer des données et
- produire une sortie,

le paradigme dataflow fournit le mécanisme d'ordre supérieur (composition) pour l'organisation de ces fonctions.

Exemple : le pipe shell UNIX

En UNIX les commandes shell peuvent être raccordées l'une à l'autre pour produire la sortie désirée.

Chaque commande shell est indépendante mais toutes vont consommer l'entrée standard et envoient des données sur la sortie standard.

Opérateur `|`

```
cat "monfichier.txt" | grep "toto" | wc -l
```

Files d'attentes

Les opérateurs Dataflow utilisent les files d'attente.

Ils peuvent avoir plusieurs files d'attente d'entrée et de sortie.

Ils ont également des propriétés qui permettent la manipulation de leur comportement.

Le contrôle de flux est appliqué aux files d'attente pour permettre différents taux de consommation entre les opérateurs.

La détection de Deadlock peut également être appliquée.

Dataflow et parallélisme

Le modèle de flux de données fournit un pipeline dédié au parallélisme par sa nature même.

Les opérateurs normalement ne sont pas obligés de voir la totalité de leur entrée avant de produire la sortie.

Pendant que chaque opérateur travaille, un pipeline de données fait son chemin à travers le graphe. Chaque opérateur peut être représenté par un fil, ou peut-être un ensemble de fils.

Avec de nombreux opérateurs dans un graphe de flux de données fonctionnant en parallèle, le dataflow tire facilement parti des processeurs multicœurs.

Partitionnement horizontal

Le dataflow prend également en charge le partitionnement horizontal, permettant aux données d'être segmentées et appliquées à une section répliquée d'un graphe.

Ce partitionnement peut être dynamique, en ajustant le nombre de ressources informatiques disponibles au moment de l'exécution.

Le partitionnement horizontal est un excellent moyen de « diviser pour régner » des problèmes où la dépendance de données n'est pas un problème.

Il peut à la fois s'appliquer à des processeurs multicœurs ou plusieurs nœuds dans un cluster.

Origine du paradigme

Une définition de flux de données fréquemment donnée dans les manuels de science informatique décrit une architecture selon laquelle la modification de la valeur d'un des résultats entraîne le recalcul automatique d'autres variables dépendantes.

Modèle «tableur»

Le fondateur du paradigme dataflow, Gilles Kahn, définit le modèle de programmation Kahn Networks Process (KPNs).

Selon Wikipedia, KPN définit un modèle de programmation « . . . où un groupe de processus séquentiels déterministes communiquent à travers des canaux FIFO non bornés. »

Origine du paradigme

Les KPNs ont été initialement conçus pour la programmation distribuée.

Ils sont également très utilisés pour les systèmes de modélisation de traitement du signal (opérateurs = filtres).

Le concept original de dataflow (feuilles de calcul) et KPNs ont, au fil du temps, évolué vers l'architecture logicielle de flux de données.

Qualités du dataflow

Les concepts de flux de données sont faciles à saisir conduisant à « l'évolutivité de conception. »

Le dataflow se prête bien à des environnements graphiques, ce qui apporte au calcul haute performance un niveau de convivialité très accessible.

Le Dataflow fournit un moyen de composer des applications évolutives utilisant une approche modulaire. Il le fait tout en faisant abstraction des problèmes de bas niveau tels que les fils, la synchronisation de la mémoire, et les conditions de parcours.

Qualités du dataflow

Dataflow empreinte les bonnes qualités de la programmation fonctionnelle.

Par exemple, les files d'attente de flux de données sont immuables, ce qui élimine les préoccupations au sujet de la synchronisation de la mémoire ou des effets secondaires communs des opérateurs en amont.

L'immuabilité permet également aux files d'attente d'avoir plusieurs lecteurs pour une bonne réutilisation fonctionnelle.

Il n'y a pas d'effet de bord.

Il est facile de regarder un graphe de dataflow et deviner comment il fonctionne et ce qu'il fait.

Soutien aux Big Data

Le modèle permet le chevauchement des opérations d'Entrées / Sorties avec calcul.

C'est une approche de la parallélisation totalement fonctionnelle

Cela répond à un problème majeur dans le traitement des « big data » pour les processeurs multi-coeurs de base d'aujourd'hui : alimentation suffisament rapidement les processeurs avec des données.

Le Dataflow passe facilement à l'échelle vers, contrairement à Hadoop ou Map Reduce, qui ne passent pas à l'échelle vers le bas en raison de leur complexité intrinsèque.

Soutien aux Big Data

L'architecture de flux de données exploite naturellement les processeurs multi-coeurs.

Les mêmes principes peuvent être appliqués à des grappes multi-noeuds par l'extension des files d'attente de flux de données sur les réseaux avec un graphe de dataflow exécuté sur plusieurs systèmes en parallèle.

Le modèle de la composition de la construction des graphes de flux de données permet la réplication de morceaux du graphe à travers plusieurs nœuds.

La portée du dataflow s'étend donc aux problèmes de big data.

Dataflow et Acteurs

Le modèle de l'acteur a été popularisée par des langages tels que Erlang et Scala.

Dans ces modèles, des acteurs indépendants communiquent par messages.

Quand un acteur reçoit un message, il agit sur lui comme défini par des fonctions à l'intérieur de l'acteur.

Chaque acteur définit ses paramètres de communication qui permettent aux autres acteurs de le trouver et lui envoyer des messages.

Dans certaines mises en œuvre, les messages sont sous forme libre et les acteurs utilisent des expressions régulières pour distinguer les messages et la façon dont ils sont traités.

Le modèle Acteurs, exemple en Scala

Ce code est un fragment car il ne comprend pas les importations ou les définitions des classes de messages.

La classe MessageHandler étend la classe de bibliothèque Acteur, redéfinissant la méthode `act()`.

La méthode `act()` est appelée pour gérer les messages entrants.

L'exemple de code reçoit des messages dans une boucle jusqu'à ce qu'un message d'arrêt soit reçu.

Une fois qu'un message arrêt est reçu, l'échantillon invoque `exit()`, pour mettre fin à l'instance de l'acteur.

Le modèle Acteurs, exemple en Scala

```
class MessageHandler extends Actor {
    def act() {
        while (true) {
            receive {
                case Message =>
                    // Do some work to handle the message
                    // Send an acknowledgement
                    sender ! Ack
                case Stop =>
                    // Terminates the actor
                    exit()
            }
        }
    }
}
```

Le modèle Acteurs

Les modèles de flux de données et acteur sont très similaires.

Ils s'appuient tous deux sur un style de programmation fonctionnelle, sans partage de la mémoire.

Ils sont faciles à étendre en écrivant de nouveaux opérateurs ou acteurs.

Le multithreading est transparent pour le développeur.

Pas de mémoire partagée signifie pas de soucis de synchronisation.

Le modèle Acteurs, autre exemple

Voici un exemple écrit plus proche du style dataflow.

C'est un opérateur simple qui prend des chaînes comme entrées et renvoie la longueur de chaque chaîne en sortie. La construction de boucle est similaire à un acteur, mais gère un type de données plus statique.

Le modèle Acteurs, autre exemple

```
public class StringLength extends DataflowProcess {

    private final StringInput input;
    private final IntOutput output;

    public StringLength(Dataflow source) {
        // Handle properties, set up inputs and outputs ...
    }
    @Override
    protected void execute() {
        while (input.stepNext()) {
            output.push(input.asString().length());
        }
        output.pushEndOfData();
    }
}
```

Dataflow vs. modèle Acteurs

Il existe des différences entre les deux modèles.

Le modèle Acteurs ne garantit pas l'immutabilité ou l'ordonnancement des messages, contrairement au Dataflow.

Le modèle Acteurs n'est pas non plus optimisé pour un grand nombre de messages ou des messages avec de grandes quantités de données.

Dataflow fournit le contrôle de flux. Sans contrôle de flux, soit les messages risquent d'être abandonnés, ou des problèmes d'utilisation de la mémoire se produisent quand les ports de communication sont pleins.

Dataflow vs. modèle Acteurs

Dataflow fait généralement la liaison des composants statiquement au moment de la composition, alors que le modèle Acteurs le fait dynamiquement. La liaison statique permet une analyse graphique pour optimiser plus facilement.

Le modèle Acteurs semble beaucoup mieux adapté au parallélisme basé sur les tâches pour lesquelles la concurrence est importante.

Par exemple, les programmes Erlang contrôlent les équipements de télécommunications avec des milliers de demandes par minute.

Performance

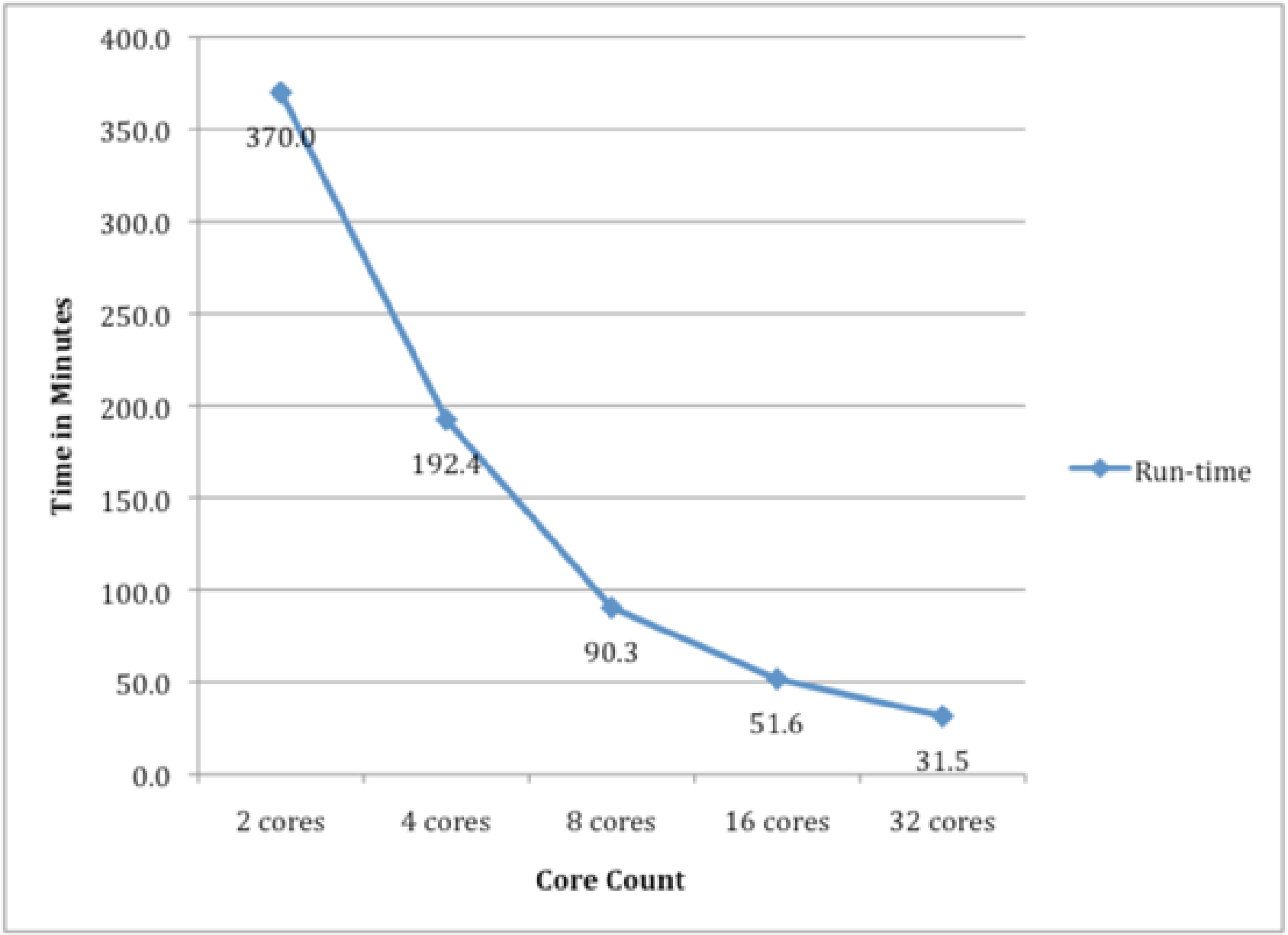
Benchmark MalStone B10 : 10 milliards de lignes de données de log dans cinq domaines, pour mesurer la performance des frameworks de flux de données.

Presque un téraoctet de données.

Machine à nœud unique avec 32 cœurs.

Le code extrait l'année et la semaine à partir d'un champ timestamp et agrège un indice qui est calculé à partir de l'id du site, l'année et la semaine.

Performance



Dataflow, implémentation

De nombreux langages existent pour implémenter le modèle dataflow.

Exemple de framework récent, open-source et facile à intégrer dans Eclipse : Orcc

http ://orcc.sourceforge.net/

On peut également citer Oz, qui répond à de nombreux paradigmes, dont le dataflow.