UNIVERSITÉ DU QUÉBEC À MONTRÉAL

# TRAITEMENT EFFICACE DE FLUX DE DONNÉES SUR MACHINES MULTI-CŒURS À L'AIDE DE FASTFLOW

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

IULIAN CIOBANU

MOIS DU DÉPÔT ANNEE DU DÉPÔT

# REMERCIEMENTS

blah blah

TABLE DES MATIÈRES

LISTE DES FIGURES

# LISTE DES TABLEAUX

Tableau Page

# RÉSUMÉ

Un bref résumé du mémoire/thèse...

# CHAPITRE I

# INTRODUCTION

Blah blah ...

Signalons que l'utilisation du package `inputenc` permet d'utiliser des caractères accentués normaux, plutôt que des caractères accentués dans l'ancien style (commandes pour les accents). [1]

On remarquera aussi, dans la note en bas de page, l'utilisation des guillemets français, appelés aussi parfois des «chevrons», qu'on doit utiliser plutôt que les "guillemets anglais". Ces guillemets français s'obtiennent à l'aide des caractères <<...>>.

Une autre façon est de définir la macro suivante, puisqu'avec certains *packages*, les caractères <<...>> ne semblent pas fonctionner :

```
\newcommand{\QUOTE}[1]{\og #1 \fg{}}
```

1. Le même texte écrit avec les accents en commande serait alors le suivant : «Signalons que l'utilisation du package `fontenc` permet d'utiliser des caractères accentués normaux, plutôt que des caractères accentués dans l'ancien style (commandes pour les accents).»

# CHAPITRE II

# ANALYSE DES OUTILS CONNUS

# CHAPITRE III

# DESCRIPTION DE L'API

Ce chapitre présente la description complète de l'API. Sa conception permet aux utilisateurs de tirer parti de la simplicité d'utilisation tout en cachant la complexité concernant les mécanismes concurrents utilisés. La figure 3.1 montre une vue d'ensemble de l'architecture du système. L'API est composée de quatre composants principaux : l'Interface avec lequel le développeur interagit, le Pipeline le coeur de l'API, les Stages et les Operateurs. Le rôle de chaque composant dans l'API est présenté dans la section suivante. La dernière section de ce chapitre décrit en détail les plus importantes méthodes implémentées dans l'interface.

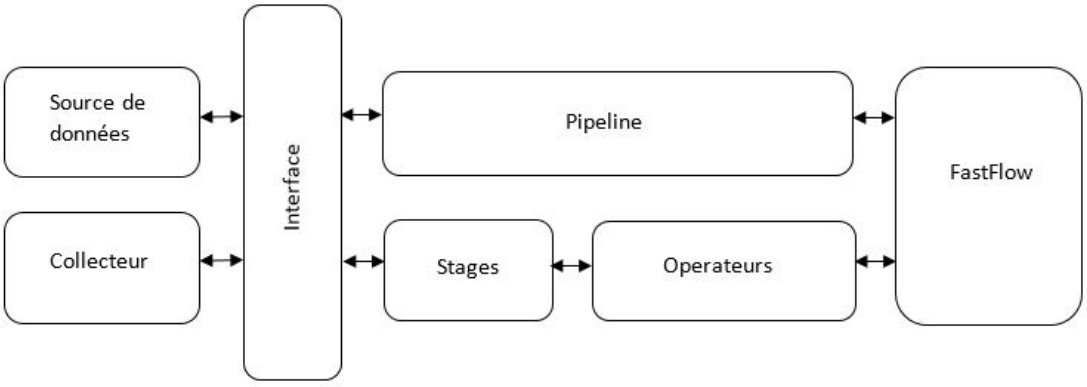


**Figure 3.1** Components de l'API.

## 3.1 Les composants de l'API

### 3.1.1 Interface

L'interface proposée en PpFf consiste en un ensemble de méthodes qui permettent á l'utilisateur de manipuler des flux de données de manière simple et efficace. L'interface suit de près l'interface introduite dans la version Java Stream 8. Le tableau 3.1 décrit brièvement toutes les méthodes implémentés dans l'API.

Comme on peut le voir dans le tableau 3.1, la déclaration des méthodes utilise la programmation générique de C++ qui sont les templates. Cela permet aux utilisateurs d'avoir une interface unique, de sorte qu'il peut être réutilisé pour n'importe quel type de données.

Un autre point clé dans cette interface le représente l'expressivité. Même avant de la conception, on s'est proposé de fournir un système suffisamment intuitif et expressif pour le traitement de flux de données. Le pseudocode 3.1 montre un extrait de code pour donner un premier aperçu de l'expressivité de l'interface. D'autres exemples seront fournis plus tard dans ce document. Dans l'exemple fourni, on sélecte seulement les employées qui ont un salaire plus grand que 35 milles. Les employées représentées par un conteneur STL sont filtrées en enchainant trois opérations : source qui a le rôle d'envoyer dans le flux les objets de type Employee, filtre qui filtre les employées selon la condition fournie en paramètre et la dernière opération collect qui a le rôle de collecter les employées une fois filtrées dans un conteneur STL. Dans cet exemple les employées sont ramassées dans un conteneur de type vecteur. On note que le type de conteneur est donné par le type fourni en paramètre template de la méthode collect.

**Pseudocode 1** Léxpressivité de l'API.

```
std::vector<Employee> sourceEmployees;

std::vector<Employee> result =

        Pipe()

        .source<Employee>(sourceEmployees.begin(), sourceEmployees.end())

        .find<Employee>([](Employee *e) ->bool {return e->salary > 35000;})

        .collect<Employee, std::vector>();
```

### 3.1.2 Opérateurs

Les opérateurs sont la base de notre système. L'API fournit un ensemble d'opérateurs qui augmentent la productivité de l'utilisateur. Les opérateurs sont structurés en deux catégories : les opérateurs sans état et opérateurs avec l'état.

Les opérateurs sans état sont les opérateurs qui ne disposent pas d'informations sur l'itération en cours et ne transmettent pas les informations intermédiaires des étapes de traitement précédentes. Si on prend comme exemple le filtre représenté par la méthode find de le tableau 3.1, il traite le flux de données élément par élément. Lorsque la fonction fournie en paramètre de la méthode find ne satisfait pas la condition de filtrage, le filtre ne retournera rien. Un opérateur sans état, par contre il peut utiliser des données historiques stockées dans la mémoire locale ou sur le disque.

Les opérateurs avec l'état sont les opérateurs qui maintiennent une structure de données interne appelée l'état. Cette structure préserve l'historique des opérations passées et affecte la logique de traitement dans les calculs ultérieurs. Par exemple l'opérateur Sum calcule la somme des éléments du flux. Son état contient la valeur de l'élément en cours et la valeur de la somme de tous les éléments précédents celui-ci.

### 3.1.3 Stages

Le traitement du flux de données est modélisé en utilisant une chaîne d'étapes. Dans notre API, une étape est représentée par un stage. Ce module n'est pas visible à l'utilisateur. Chaque stage est composé d'un ou plusieurs opérateurs.

### 3.1.4 Pipeline

Le pipeline est le composant principal de l'API. Un Pipeline est une chaîne de traitement composée d'un ou de plusieurs opérateurs groupés dans des stages. La figure 3.2 montre une vue détaillée de pipeline en action. Une éétape de la chaîne de traitement de ce modèle traite les données produites par l'étape précédente dans le flux et fournit les résultats à l'étape suivante dans le flux. Un pipeline P avec n étapes peut être formellement défini comme :

```
P = O1 + O2 + O3 + ... + On;
```

Dans l'expression ci-dessus O est représenté par le n-em opérateur dans le Pipeline.

L'utilisation de Pipeline introduit une couche d'abstraction sur une chaîne complexe d'opérateurs. De plus il n'expose pas à l'extérieur que les entrées et les sortirs du Pipeline. Une telle conception modulaire ajoute beaucoup de flexibilité au système tout en simplifiant la mise en œuvre. Par exemple, le parallélisme du flux pourrait être facilement réalisé en ayant plusieurs Pipelines identiques connectés à la même entrée et sortie respectivement.

| Méthods | Retourn type | Description |
|---|---|---|
| template<typename T><br>allMatch(std : :function<bool(T*)> predicate) | bool | Retourne vrais si tous les éléments de ce flux correspondent au prédicat fourni. |
| template<typename T><br>anyMatch(std : :function<bool(T*)> predicate) | bool | Retourne vrais si au moins un élément de ce flux correspondent au prédicat fourni. |
| template < typename T, template <typename ELEM, class ALLOC = std : :allocator<ELEM» class TContainer > collect() | TContainer<T> | Retourne un conteneur de type STL sur les éléments de ce flux. |
| count() | unsigned int | Retourne le nombre d'éléments dans ce flux. |
| template<typename In><br>find(std : :function<bool(In*)> const& taskFunc) | Pipe& | Renvoie dans le flux tous les éléments qui satisfont la condition fournie en paramètre. |
| template<typename In, typename Out, typename OutContainer><br>flatMap(std : :function<OutContainer*(In*)><br>const& taskFunc) | Pipe& | Renvoie dans le flux le résultat produit en appliquant la fonction de mappage fournie en paramètre à chaque élément. |
| template<typename In, typename Out, typename OutContainer = In><br>flatMap() | Pipe& | Renvoie dans le flux les éléments du conteneur si celui-ci est un élément du flux. |
| template<typ name In, typename K = In, typename V = In, typename MapType><br>groupByKey(std : :function<K*(In*)> const& taskFuncOnKey, std : :function<V*(In*)><br>const& taskFuncOnValue = identity<In,V>) | MapType | Retourne un map avec les éléments du flux groupés par clé. |

**Figure 3.2** Les méthodes exposées aux utilisateurs par l'API.

# CHAPITRE IV

# LA MISE EN OUVRE DE L'API

# CHAPITRE V

# ÉTUDES DE CAS ET EXPÉRIENCES

# CHAPITRE VI

# CONCLUSION

# APPENDICE A

## TITRE DE L'ANNEXE

— note.dtd

# APPENDICE B

## TITRE DE CETTE ANNEXE

stuff ....