NOUVELLE INTERFACE DE PROGRAMMATION PARALLELE : APPLICATION A UN DSL POUR LA PARALLÉLISATION DES ALGORITHMES DE MACHINE LEARNING

SCIENCE EVENTS CAMEROON 2022

Présenté par

Nel Gerbault NANVOU TSOPGNY,

Co-dirigé par

Pr. Alain TCHANA

Dr. Etienne KOUOKAM

Dr. Thomas MESSI NGUELE

Yaoundé, 20 Décembre 2022

Définitions

DSL

- Langage machine binaire, besoin de langages de programmation.
- Langage de programmation scindés en deux catégories :
  1. GPL
  2. DSL (externes et internes)

Machine Learning

Contexte et problème

Contexte

- Temps d'exécution des algorithmes élevé au passage à l'échelle.
- Limites de la solution consistant à augmenter la fréquence des processeurs.
- Avènement des Architecture multi-many coeurs.

Problème

- Machines à disposition sous-exploitées : difficultés de parallélisation.
- Méthodes de parallélisation disponibles figées.

Travaux existants

	Chu et al (2007)[2]	OptiML (2011)[4]	Qjam (2012)[1]
Implémenté	non	oui	oui
disponibilité du code	-	oui	non
Prototypage	-	non	oui
Implemenété en langage haut niveau	-	oui (scala)	oui(python)
Facilité d'Utilisation	-	Non	-
Nombre de lignes de code	-	peu	-

Aucune implémentation n'a été fait dans un langage de bas de niveau tel que le C.

Question de Recherche

Comment écrire simplement, les algorithmes de Machine Learning, s'exécutant automatiquement et efficacement de façon parallèle sur des architectures Multi-coeurs et sur architectures distribuées ?

FastML

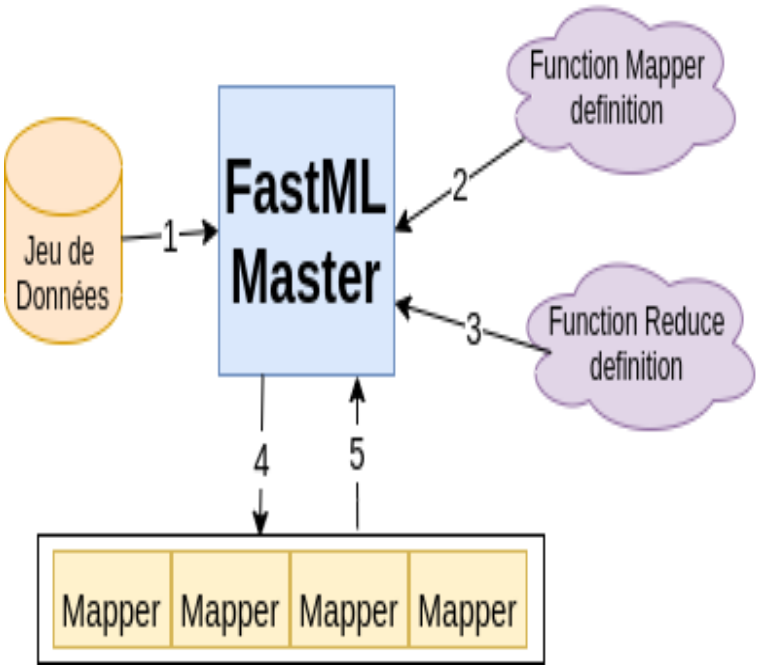


Figure – Architecture de notre DSL FastML : *https ://gitlab/gnelnanvou/fastml.git*

1. Définition du jeu de données.
2. Définition de l'opération *fonctionMapper*.
3. Définition de l'opération *fonctionReducer*
4. Division du jeu de données et création des mappers.
5. Synchronisation et agrégation des résultats intermédiaires.

Interface de programmation FastML

Types de base

- matrice
- matricechar

Définition du jeu de données

- readMatrice(X, filename, separator)
- readMatrice_char(X_char, filename, separator)
- readDatas(X, Y, filename, n_colums, separator)
- readDatas_char(X_char, Y_char, filename, n_colums, separator)

Performances obtenues

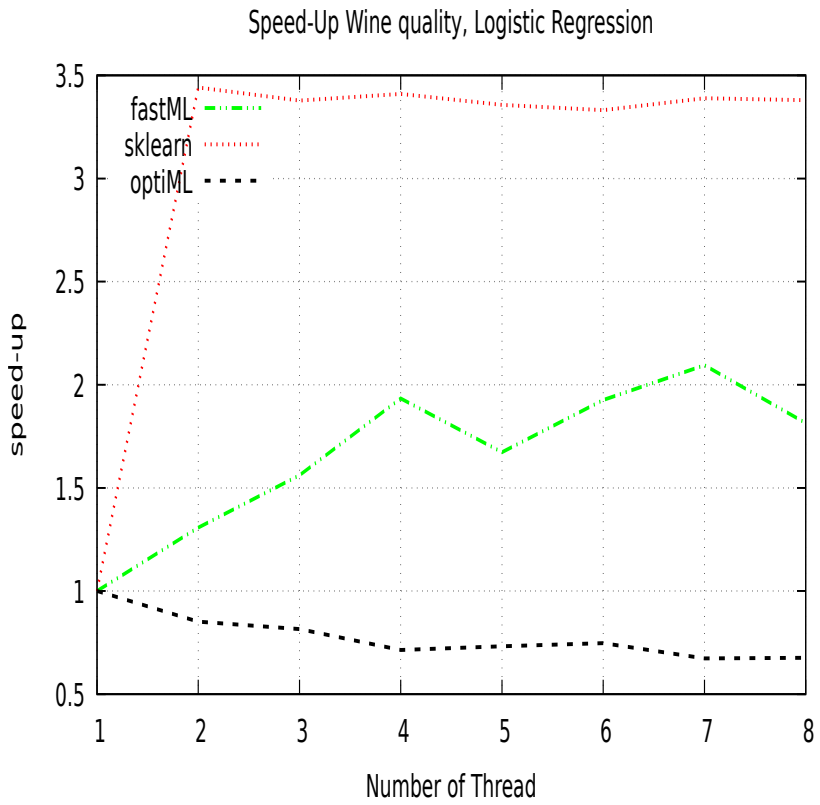


Figure – Logistic regression Speedup

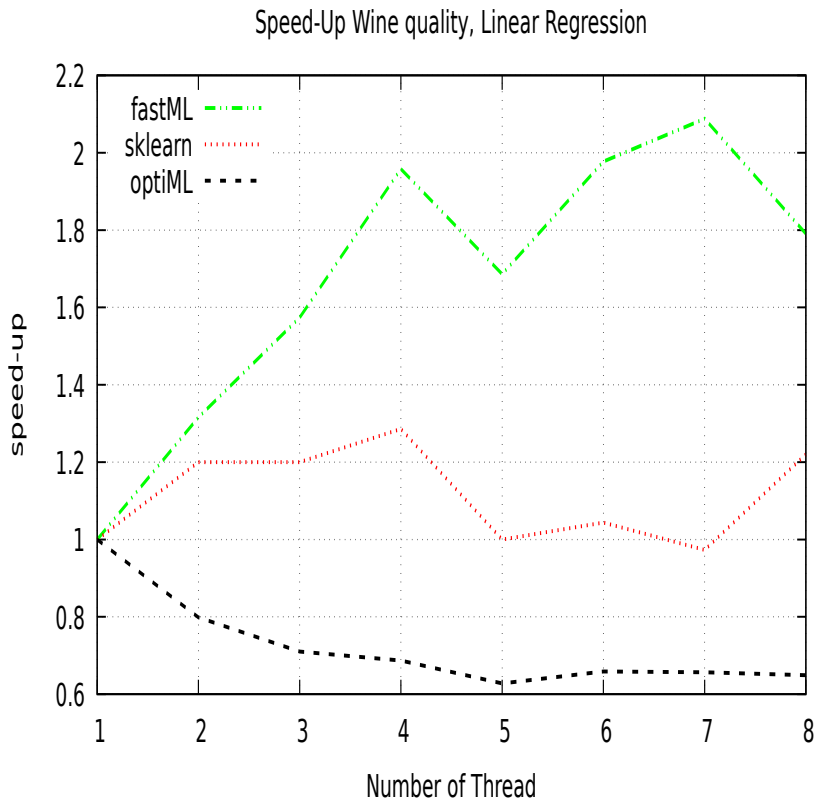


Figure – Linear regression Speedup

Performances obtenues

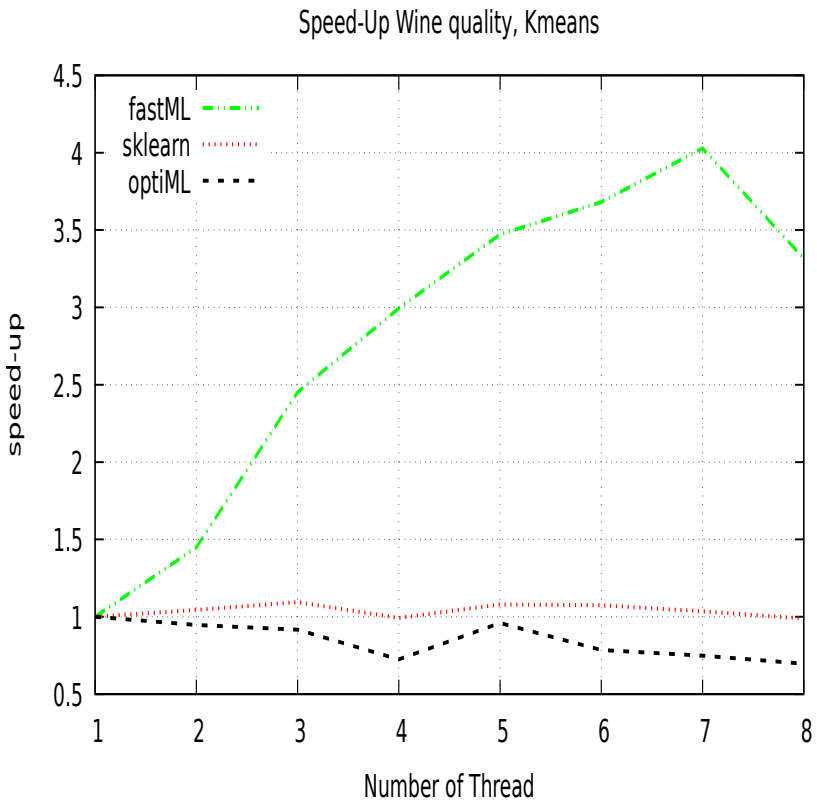


Figure – Kmeans speedup

Discussion

- Résultats obtenus avec FastML promoteurs (jusqu'à 4x pour FastML, 1x pour Kmeans et 0.73x pour sklearn)
- Travail présenté au CARI 2022
- Notre travail de thèse porte principalement sur l'amélioration de FastML

Travail à faire dans la suite

Amélioration suivant deux axes :

Optimisation de la stratégie de parallélisation

- Peu d'algorithmes dans l'étude actuellement ;
- Nécessité d'inclure d'autres algorithmes dans l'étude pour valider l'efficacité de FastML ;
- juste une implémentation multi-coeur ;
- **Proposer une version distribuée.**

Déconstruction des concepts de thread et processus

- Utilisation des threads ou processus ;
- Conséquences sur les performances et la tolérance aux pannes ;
- Choix entre forte isolation mais faible observabilité (Processus) et forte observabilité mais faible isolation (thread).
- trouver une interface qui génère automatiquement du code efficace sur multi-coeur et sur architectures distribuées.

Optimisation de la stratégie de parallélisation

- Analyse de la parallélisation des algorithmes basés sur les CNNs, RNNs, GNNs, ...
- Construction d'une stratégie générale de parallélisation d'algorithmes de Machine Learning
- Déployement de FastML (le rendre disponible en ligne au même titre que les frameworks comme Scikit-learn ou Pytorch).

Déconstruction des concepts de thread et processus

Exemple d'un travail qui déconstruit les concepts de threads et processus :

Travil de Yuzhuo Jing et al. (2022) : Orbit [3]

- Support au SE pour l'exécution des Tâches auxiliaires (TA)
- Les TA sont utiles pour l'auto maintenance
- Propriétés de Orbit :
  - Forte isolation
  - Forte observabilité
  - Synchronisation automatique des états
  - Modification sécurisée du programme principal

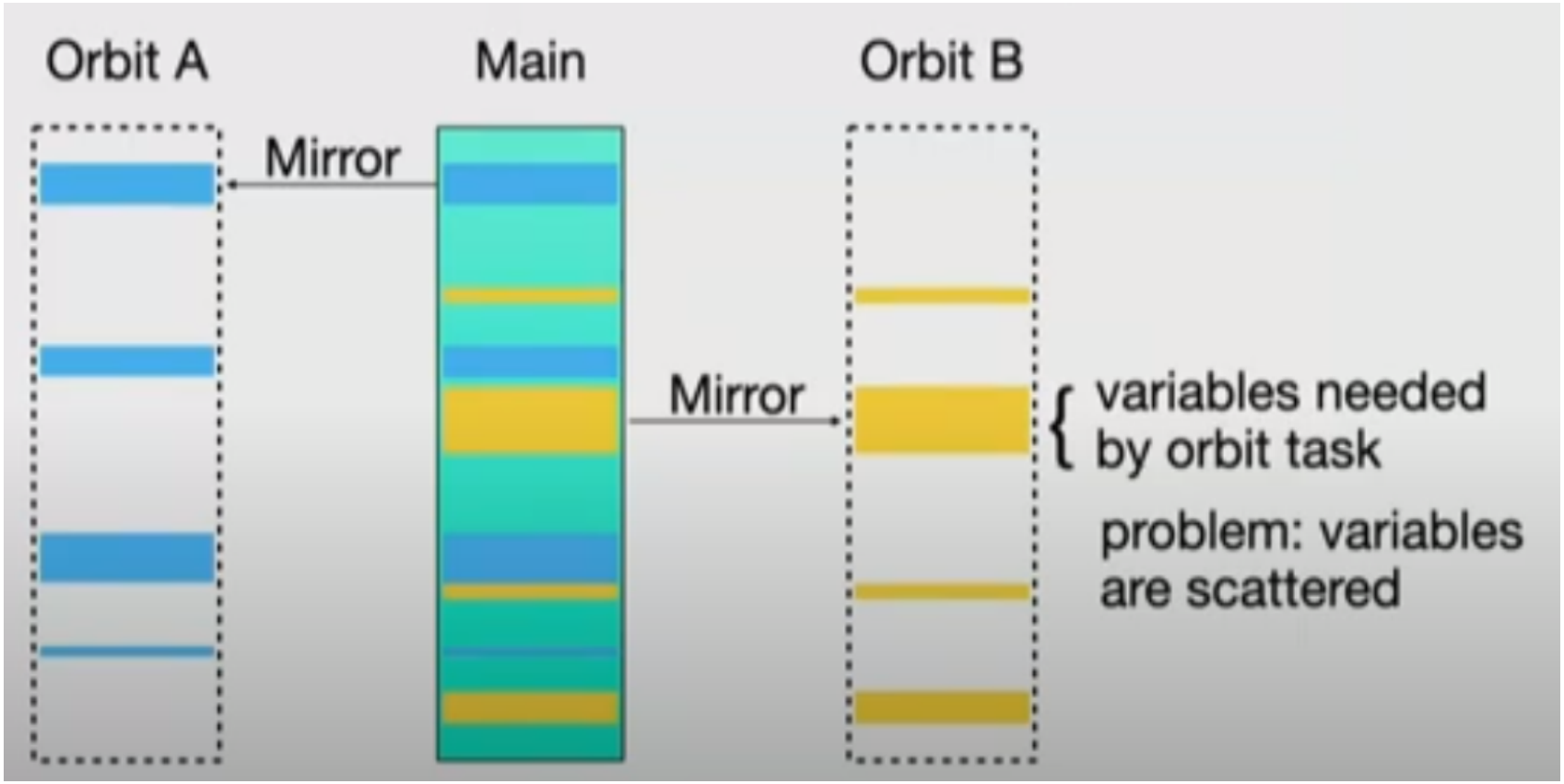
Vue d'ensemble d'orbit

Fournit sous forme d'un ensemble de fonctions

```
orbit *orbit_create(const char *name, orbit_entry entry,
                    void* (*init)(void))
int orbit_destroy(orbit *ob)
orbit_area *orbit_area_create(size_t init_size, orbit *ob)
void *orbit_alloc(orbit_area *area, size_t size)
```

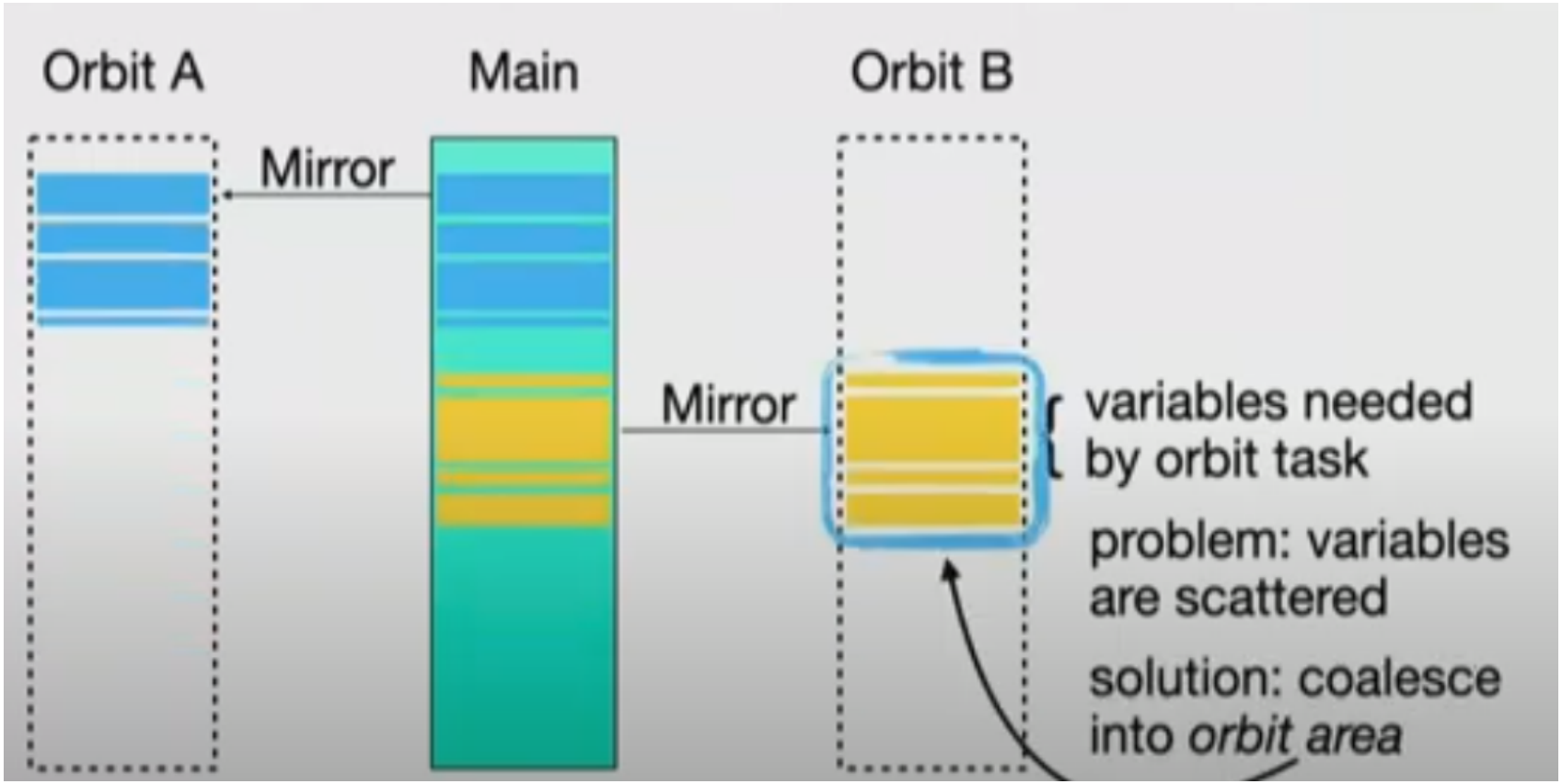
```
long orbit_call(orbit *ob, size_t narea, orbit_area** areas,
                orbit_entry func_once, void *arg, size_t argsize)
orbit_future *orbit_call_async(orbit *ob, int flags, size_t narea,
                orbit_area** areas, orbit_entry func_once, ...)
long pull_orbit(orbit_future *f, orbit_update *update)
long orbit_push(orbit_update *update, orbit_future *f)
```

Synchronosation d'état



Synchronosation d'état

- Analyse statique pour trouver les points d'allocation utiles pour la TA ;
- Mécanisme d'écriture par copie ;
- La copie est faite si la TA doit modifier un état ;



Place de Orbit dans notre travail

Cette article nous donne une idée sur comment on peut traverser les limites posées par les threads et les processus

- Concevoir une interface de programmation parallèle pour prendre en compte la spécification du type de domaine de protection ;
- Concevoir un compilateur capable de choisir et d'adapter dynamiquement le mécanisme de communication ;
- Compléter le compilateur pour qu'il puisse choisir dynamiquement le type de domaine de protection (parmi ceux autorisés par le développeur) adapté à la situation ;
- Concevoir l'interface du système d'exploitation pour faciliter l'intégration des types de domaine de protection.

Au final

- Proposer une nouvelle interface de programmation parallèle ;
- Inclure cette nouvelle dans FastML ;
- Évaluer et comparer.

Conclusion

- **Objectif** : Fournir un cadre pour la parallélisation automatique des algorithmes du Machine Learning ;
- **Travail dans cette thèse** : Améliorer FastML
  1. Optimisation de la stratégie de parallélisation ;
  2. Nouvelle interface de programmation parallèle.
- **Travail actuel** : Revue de la littérature sur la mise sur pied d'une nouvelle interface.


Juan Batiz-Benet, Quinn Slack, Matt Sparks, and Ali Yahya.
Parallelizing machine learning algorithms.
In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, Pittsburgh, PA, USA*, pages 25–27, 2012.


Cheng Chu, Sang Kyun Kim, Yian Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun.
Map-reduce for machine learning on multicore.
Advances in neural information processing systems, 19 :281, 2007.


Yuzhuo Jing and Peng Huang.
Operating system support for safe and efficient auxiliary execution.
In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 633–648, 2022.


Arvind K Sujeeth, HyoukJoong Lee, Kevin J Brown, Tiark Rompf, Hassan Chafi, Michael Wu, Anand R Atreya, Martin Odersky, and Kunle Olukotun.

Optiml : an implicitly parallel domain-specific language for machine learning.
In *ICML*, 2011.


Merci de votre aimable attention!