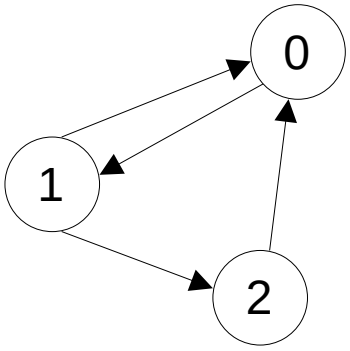
PageRank



Matrice d'adjacence A

0	1	0
1	0	1
1	0	0

Matrice d'adjacence transposée A^T

0	1	1
1	0	0
0	1	0

Matrice P = (A^T) normalisée sur les colonnes
= (A normalisée sur les lignes)^T

0	0,5	1
1	0	0
0	0,5	0

q vecteur résultat de longueur n
« nombre de nœuds dans le graphe »
Initialement :

$$\begin{bmatrix} 1/n \\ 1/n \\ . \\ . \\ 1/n \end{bmatrix}$$

(exemple ici)

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

À chaque itération du PageRank :

$$q_i = \beta(P \cdot q_{i-1}) + \frac{(1-\beta) \mid q_{i-1} \mid}{n}$$

L'opération délicate (en parallèle) est le produit matrice-vecteur $P \cdot q$

L'algorithme du PageRank est toujours le même, mais il y a plusieurs manières de l'appliquer. C'est ce qui est présenté par la suite.

Déroulé de l'algorithme du PageRank

Étape par étape

Choix du **beta** (amortissement), **epsilon** (erreur max)
Initialisation du **vecteur q_0**, et de la **somme totale de q_0**, et de l'**erreur** (pour entrer dans la boucle interne)

Tant que (erreur > epsilon) :

- **Produit matrice-vecteur** $q_{i+1} = P*q_i$: calcul du vecteur q_{i+1}, de manière répartie (en réalité : calcul de morceaux du vecteur q_{i+1} dans chaque processus)

- **Redistribution** : communication des nouveaux morceaux du vecteur q_{i+1} aux processus qui en auront besoin pour les calculs de l'itération suivante

- Amortissement et **Normalisation** du nouveau vecteur q_{i+1} (besoin de la somme totale des vecteurs q_i et q_{i+1})

- **Calcul de l'erreur** (norme de $q_{i+1} - q_i$)

Fin Tant que

Nous allons détailler par la suite les deux implémentations de cet algorithme (la version optimisant l'optimisation mémoire, et la version optimisant les communications pour Torus sur Fugaku)

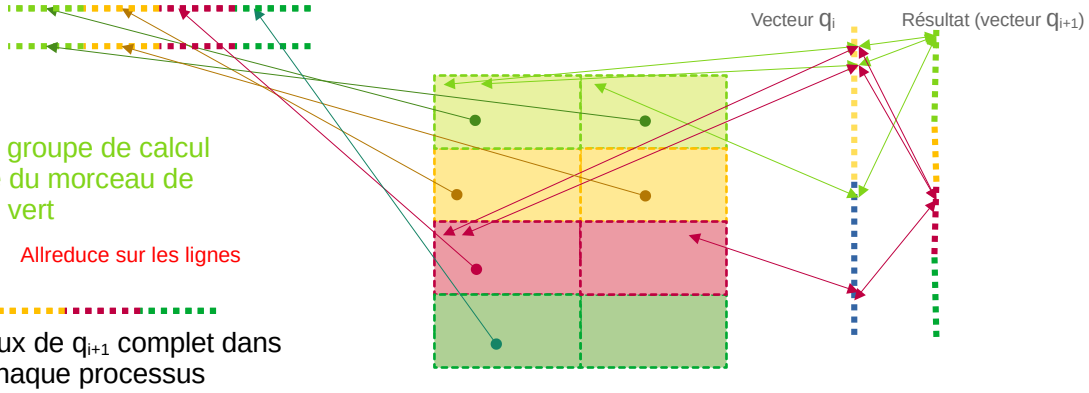
Déroulé du PageRank – Optimized Memory Usage

(effectué avec A^T non normalisée)

Étape par étape

Tant que (erreur > epsilon) :

- Produit matrice vecteur : calcul du vecteur résultat $q_{i+1} = P \cdot q_i$

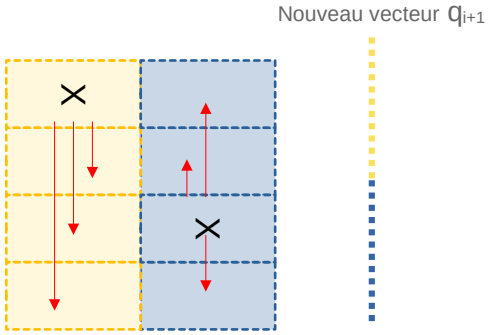


Chaque processus du groupe de calcul vert calcul une partie du morceau de vecteur q vert

Allreduce sur les lignes

Morceaux de q_{i+1} complet dans chaque processus

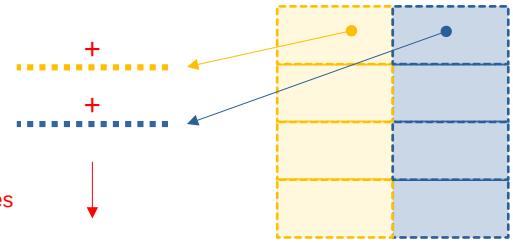
- Redistribution du vecteur résultat q_{i+1} : communication des morceaux vers les processus qui en auront besoin pour les calculs de l'itération suivante



Les processus racine (''X'') communiquent leurs morceau du nouveau vecteur q aux processus qui en ont besoin pour les calculs de l'itération suivante

Broadcast sur les lignes / colonnes selon des communicateurs spécifiques

- Amortissement et normalisation du vecteur q_{i+1}



Somme des éléments du morceau de q local

Allreduce sur les lignes

Somme totale de q_{i+1}

Division de chaque élément de q_{i+1} par la somme totale

- Calcul de l'erreur (q_{i+1} - q_i)

Somme locale des éléments du vecteur q_{i+1} - q_i

Allreduce sur les lignes

Somme totale = erreur

Fin Tant que

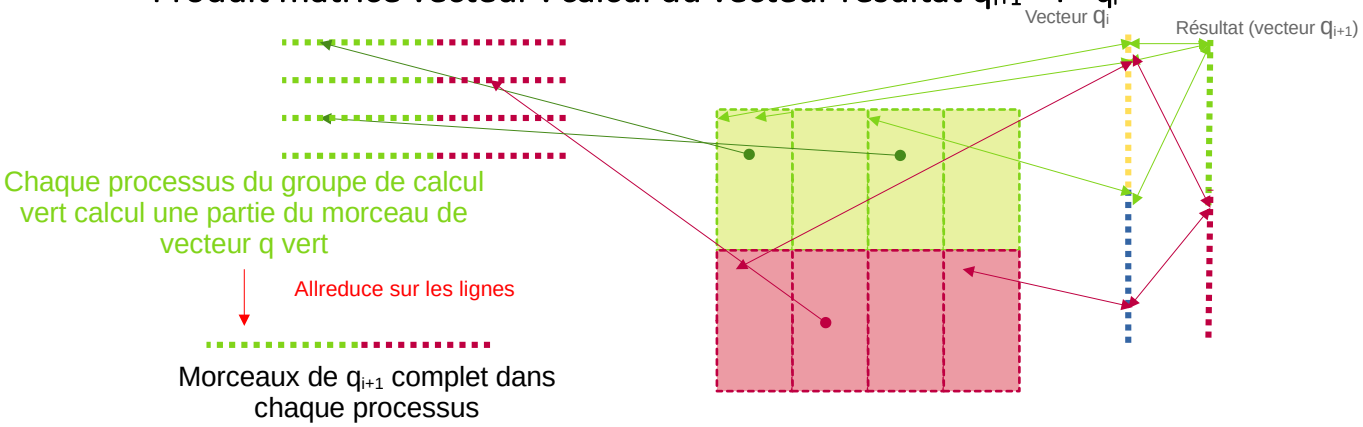
Déroulé du PageRank – Optimized Memory Usage

(effectué avec A^T non normalisée) – cas qui pose problème avec Torus

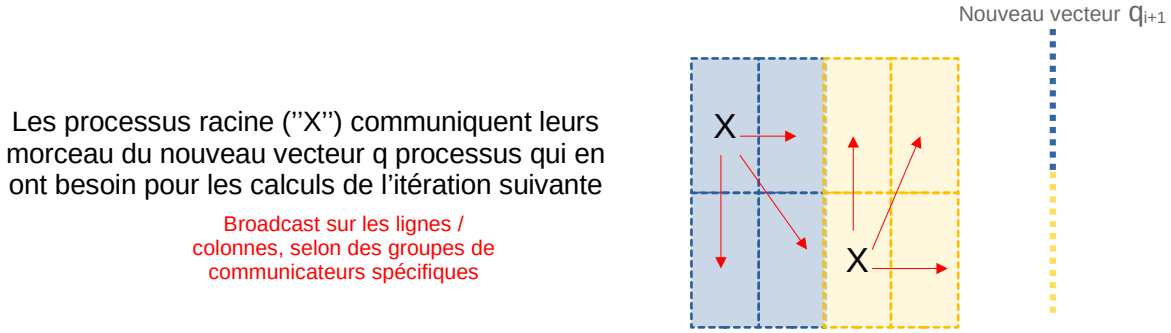
Étape par étape

Tant que (erreur > epsilon) :

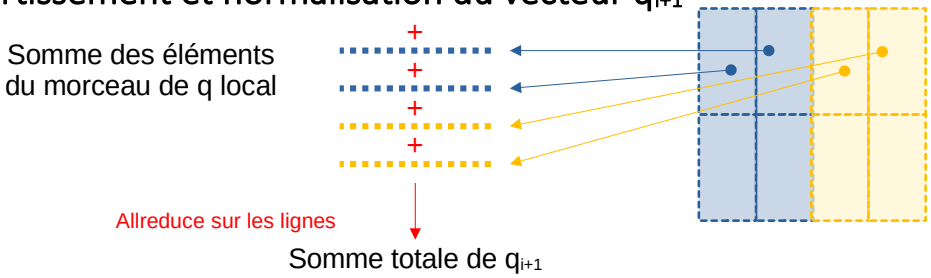
- Produit matrice vecteur : calcul du vecteur résultat $q_{i+1} = P \cdot q_i$



- Redistribution du vecteur résultat q_{i+1} : communication des morceaux vers les processus qui en auront besoin pour les calculs de l'itération suivante



- Amortissement et normalisation du vecteur q_{i+1}



Division de chaque élément de q_{i+1} par la somme totale

- Calcul de l'erreur (q_{i+1} - q_i)

Somme locale des éléments du vecteur q_{i+1} - q_i

↓ Allreduce sur les lignes

Somme totale = erreur

Fin Tant que

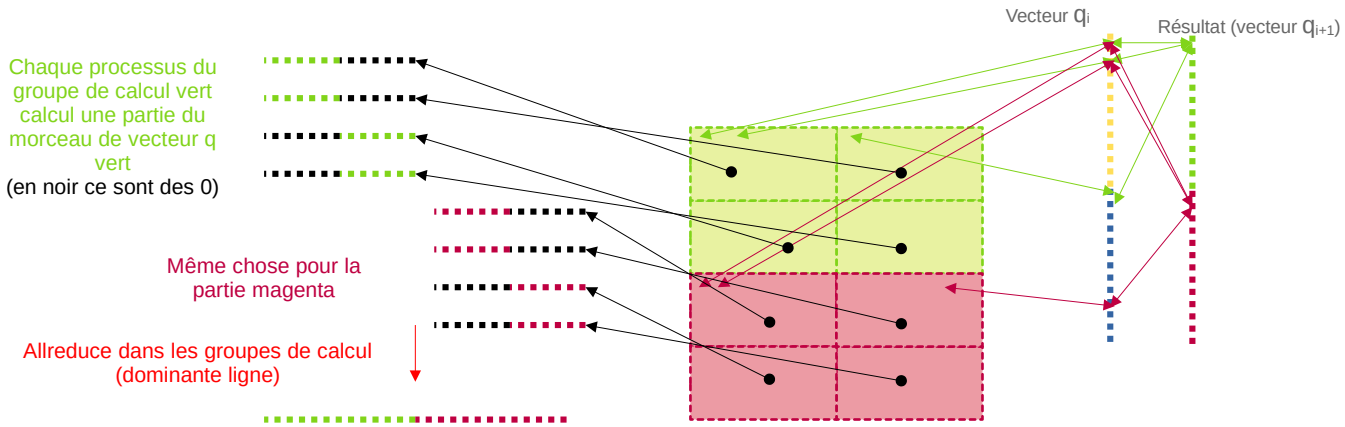
Déroulé du PageRank – Optimized Communications

(effectué avec A^T non normalisée)

Étape par étape

Tant que (erreur > epsilon) :

- Produit matrice vecteur : calcul du vecteur résultat $q_{i+1} = P \cdot q_i$

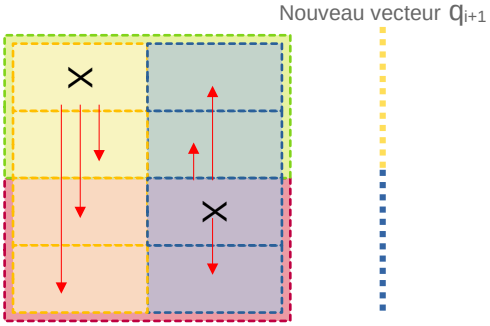


Morceau de q_{i+1} vert / magenta complet dans tout les processus respectivement vert / magenta

- Redistribution du vecteur résultat q_{i+1} : communication des morceaux vers les processus qui en auront besoin pour les calculs de l'itération suivante

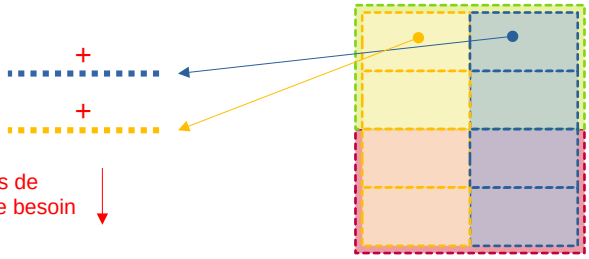
Les processus racine (''X'') communiquent leurs morceau du vecteur q_{i+1} aux autres de la même colonne

Broadcast sur les colonnes



- Amortissement et normalisation du vecteur q_{i+1}

Somme des éléments du morceau de q local



Allreduce dans les groupes de communication inter-groupe de besoin (dominante ligne)

Somme totale de q_{i+1}

Division de chaque élément de q_{i+1} par la somme totale

- Calcul de l'erreur (q_{i+1} - q_i)

Somme locale des éléments du vecteur q_{i+1} - q_i

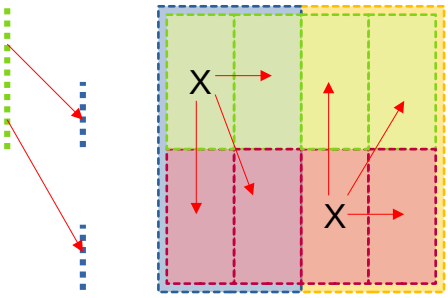
Allreduce dans les groupes de communication inter-groupe de besoin (dominante ligne)

Somme totale = erreur

Fin Tant que

PageRank avec optimisation de l'utilisation mémoire vs PageRank avec optimisation des communications

Optimized memory usage



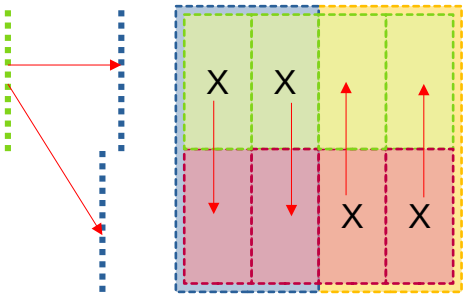
The ''root'' processes (''X'') communicate a part of their result vector to the others processes that needs them

Broadcast on the rows and columns

The stored vectors for the calculations can be smaller than the matrix-vector product result vector : less memory usage

Pose problème avec Torus (communication diagonale)

Optimized communications



The ''root'' processes (''X'') communicate their part of result vector to the others processes of the same column

Broadcast on the columns only (better for Torus)

The stored vectors for the calculations are the same size as matrix-vector product result vector : more memory usage

Pas de problème avec Torus, mais potentiellement plus de problèmes mémoire