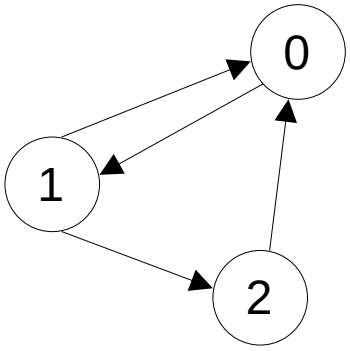
PageRank



Adjacency Matrix A

0	1	0
1	0	1
1	0	0

Transposed adjacency matrix A^T

0	1	1
1	0	0
0	1	0

P Matrix = (A^T) normalized on columns = (A normalized on rows)^T

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

result vector q of length n = "number of nodes in the graph"
Initially:

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

(example above)

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

At each PageRank iteration:

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

The tricky operation (when we are working in distributed environment) is the matrix-vector product $P \cdot q$

The PageRank algorithm is always the same, but there are several ways to apply it. This is what is presented next.

Summary of the PageRank algorithm
Step by step

Choice of **beta** (damping factor), **epsilon** (max error, to study the convergence)
Initialization of vector q_0, and total sum of q_0, and error (to enter inner loop)
While (error > epsilon) :

- **Matrix-vector product** $q_{i+1} = P \cdot q_i$: calculation of the vector q_{i+1}, in a distributed way (actually: calculation of pieces of the vector q_{i+1} in each process)

- **Redistribution**: communication of the new pieces of the q_{i+1} vector to the processes that will need them for the calculations of the next iteration

- **Damping and Normalization** of the new vector q_{i+1} (need the total sum of vectors q_i and q_{i+1})

- **Calculation of the error** (norm of $q_{i+1} - q_i$)

WhileEnd

We will detail, after this, the two implementations of this algorithm (the version optimizing memory optimization, and the version optimizing communications for Torus on Fugaku)

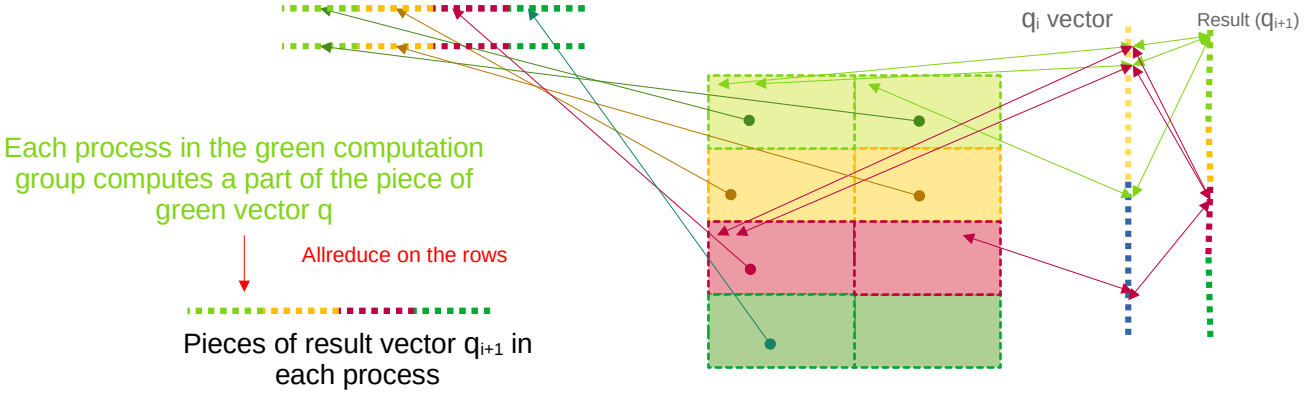
PageRank run – Optimized Memory Usage

(with A^T non-normalized)

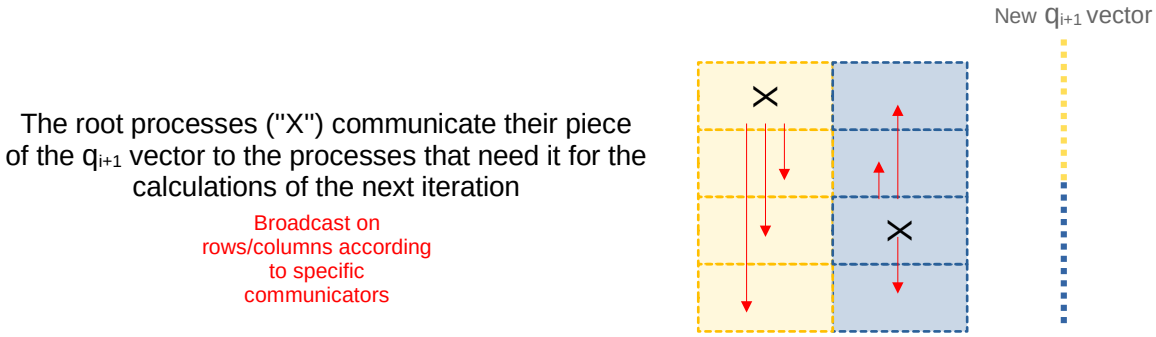
Step by step

While (error > epsilon):

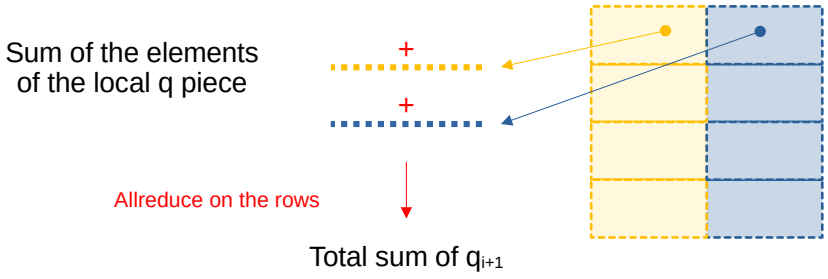
- Matrix-vector product: calculation of the result vector $q_{i+1} = P \cdot q_i$



- Redistribution of the result vector q_{i+1}: communication of the pieces to the processes that will need them for the calculations of the next iteration



- Damping and normalization of the vector q_{i+1}



Division of each element of q_{i+1} by the total sum

- Calculation of the error ($q_{i+1} - q_i$)

Local sum of vector elements $q_{i+1} - q_i$

Allreduce on the rows

error = total sum

WhileEnd

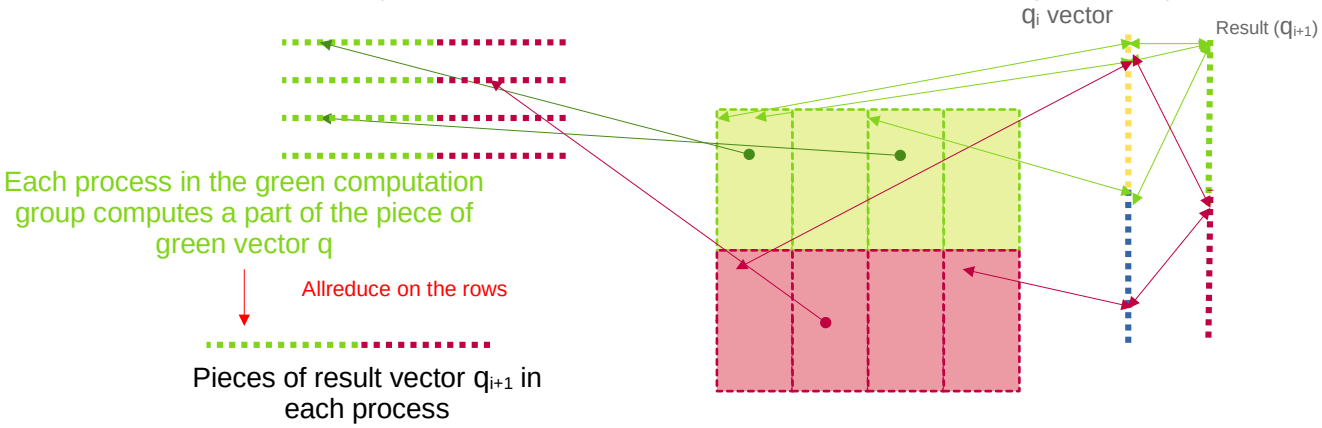
PageRank run – Optimized Memory Usage

(with A^T non-normalized) – case that causes problems with Torus

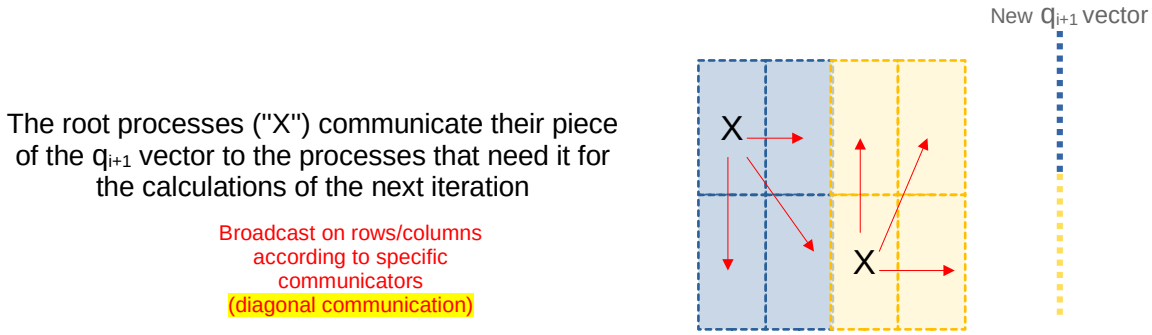
Step by step

While (error > epsilon):

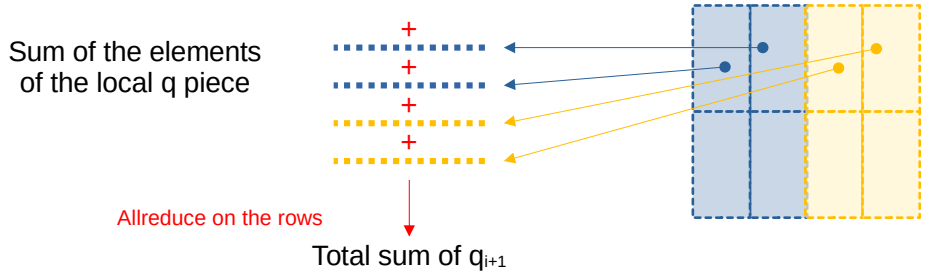
- Matrix-vector product: calculation of the result vector $q_{i+1} = P \cdot q_i$



- Redistribution of the result vector q_{i+1}: communication of the pieces to the processes that will need them for the calculations of the next iteration



- Damping and normalization of the vector q_{i+1}



Division of each element of q_{i+1} by the total sum

- Calculation of the error ($q_{i+1} - q_i$)

Local sum of vector elements $q_{i+1} - q_i$

Allreduce on the rows

error = total sum

WhileEnd

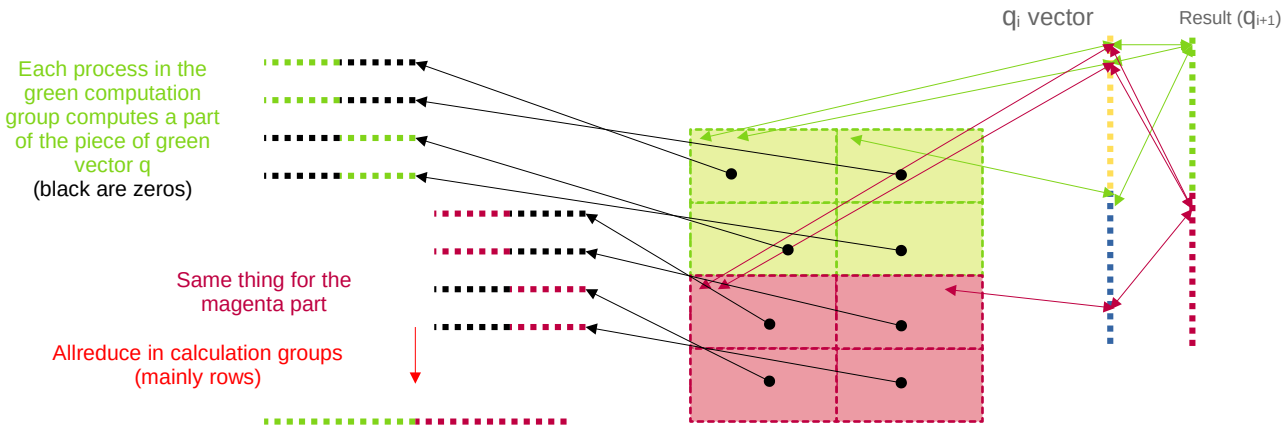
Déroulé du PageRank – Optimized Communications

(with A^T non-normalized)

Step by step

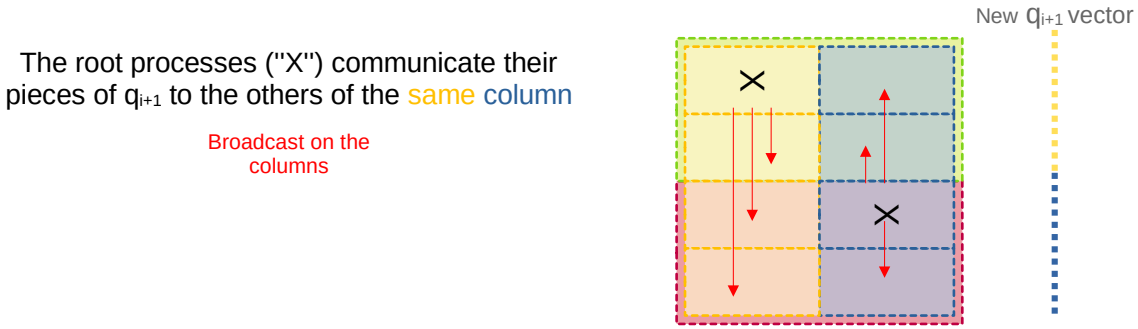
While (error > epsilon):

- Matrix-vector product: calculation of the result vector $q_{i+1} = P \cdot q_i$

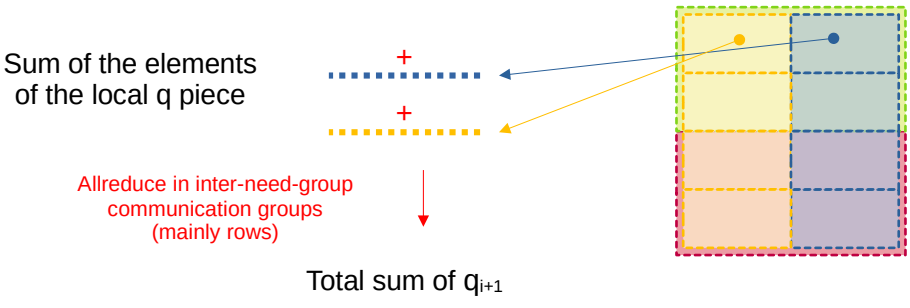


green / magenta pieces of q_{i+1} in all processes green / magenta respectively

- Redistribution of the result vector q_{i+1}: communication of the pieces to the processes that will need them for the calculations of the next iteration



- Damping and normalization of the vector q_{i+1}



Total sum of q_{i+1}

Division of each element of q_{i+1} by the total sum

- Calculation of the error (q_{i+1} - q_i)

Local sum of vector elements q_{i+1} - q_i

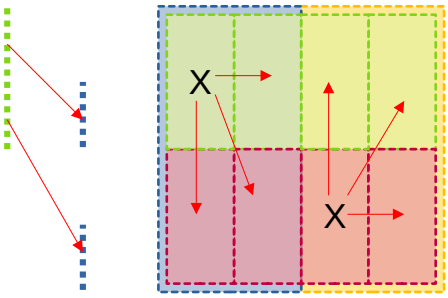
Allreduce in inter-need-group communication groups (mainly rows)

error = total sum

WhileEnd

PageRank optimized memory usage vs PageRank optimized communications comparison (result vector redistribution) :

Optimized memory usage



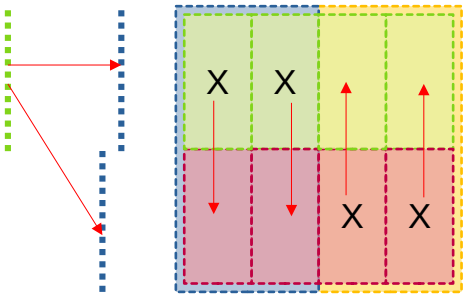
The ''root'' processes (''X'') communicate a part of their result vector to the others processes that needs them

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

Problem with Torus (diagonal communication)

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

No problem with Torus, but potentially more memory issues