- We establish a closed-from quantity model to analysis the speedup efficiency for regular, toroidal rectangle and hypercube network.
    a) Further, the data injection is on the corner, boundary processor and inner grid.
    b) With front end and Without front-end

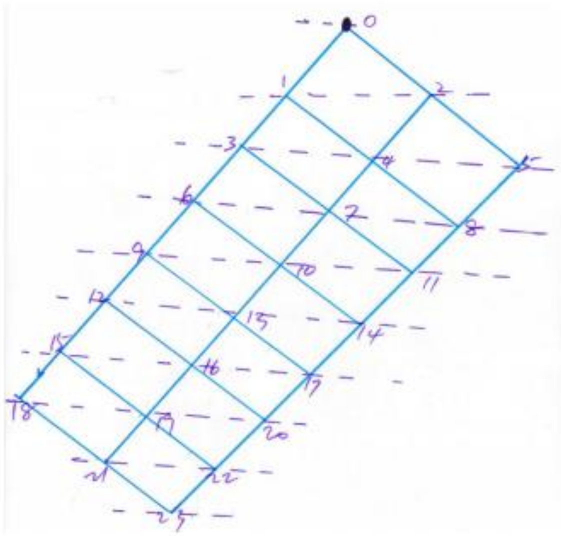

Figure 3.9: 3*8 regular network. The data injection position is P_0

Fig. 3.9:

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_9 \\ \alpha_{12} \\ \alpha_{15} \\ \alpha_{18} \\ \alpha_{21} \\ \alpha_{23} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.50}$$

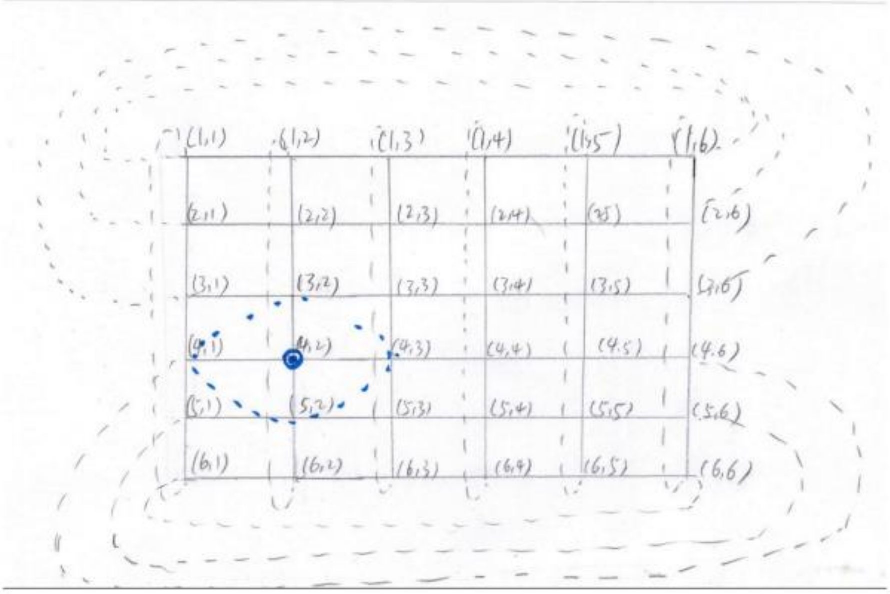

Figure 4.3: The m*n toroidal rectangle network and the data injection is $P_{4,2}$

The $D_{k,i}$ table is as follow in table Table 4.1

The flow matrix closed-form is

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

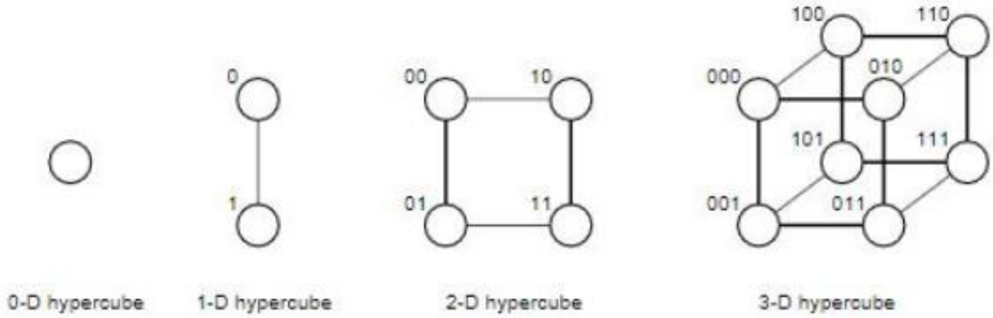

Figure 5.1: Hypercube in 0, 1, 2, 3 dimension. [1]

A general case, $D-$dimension network, the flow matrix is :

$$A = \begin{bmatrix} \binom{n}{0} & \binom{n}{1} & \binom{n}{2} & \cdots & \binom{n}{n-2} & \binom{n}{n-1} & \binom{n}{n} \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix}$$

- We establish a sensitivity analysis model for regular, toroidal rectangle and hypercube network.
  - Corner, boundary processor and inner grid processor.
  - Frontend and without frontend
  - Comparison between the regular network and toroidal network

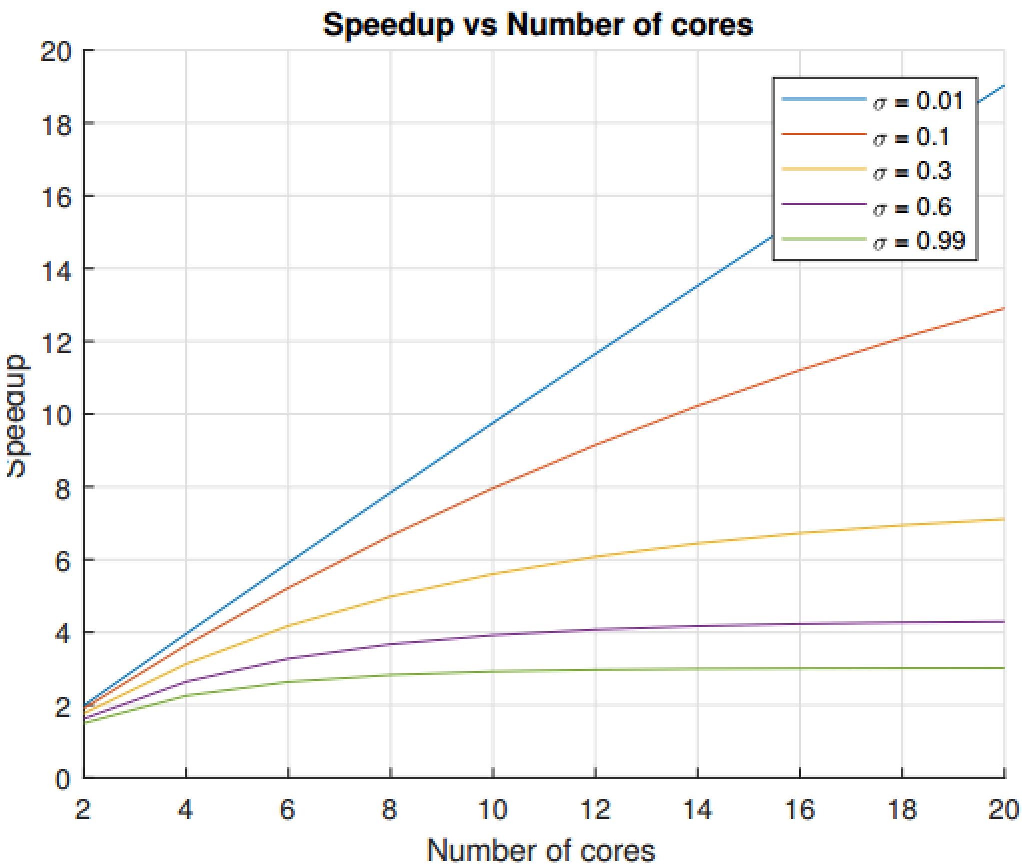

Figure 3.41: Speedup curve of 2*10 regular network result

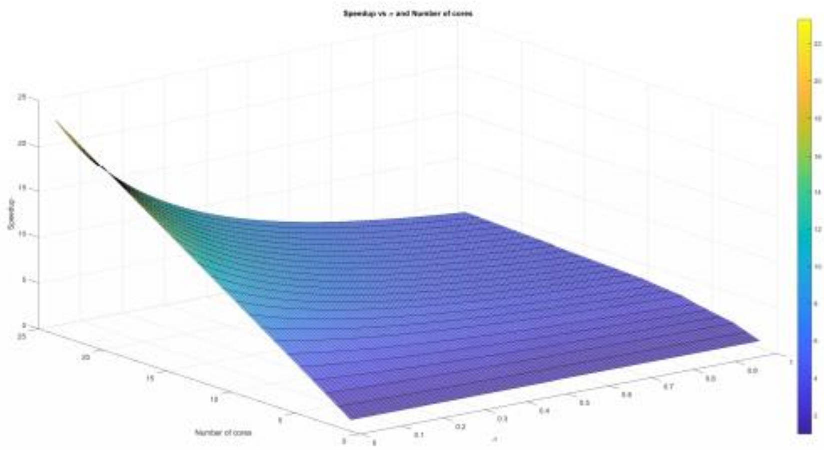

Figure 3.42: Sensitivity analysis result of 3*8 regular network result

- We propose a quantity model for multi-source optimal (Even data fraction) or suboptimal (different data fraction) regular, toroidal network and hypercube network.
- Further, we propose three algorithms to save processors to hit the same execution time.

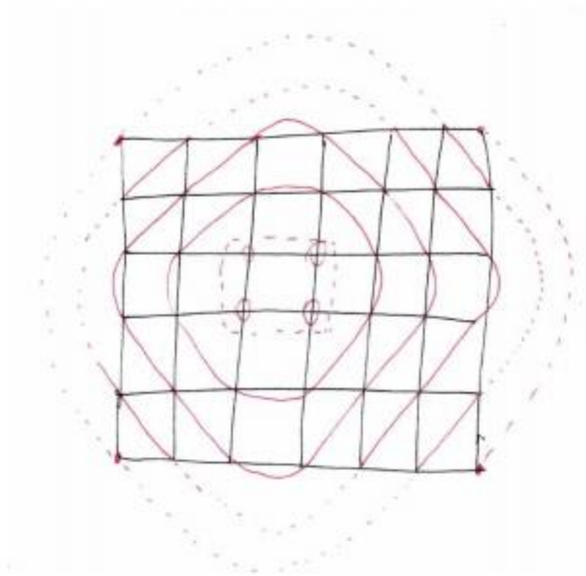

Figure 3.22: Data injection consists of a subgraph of G

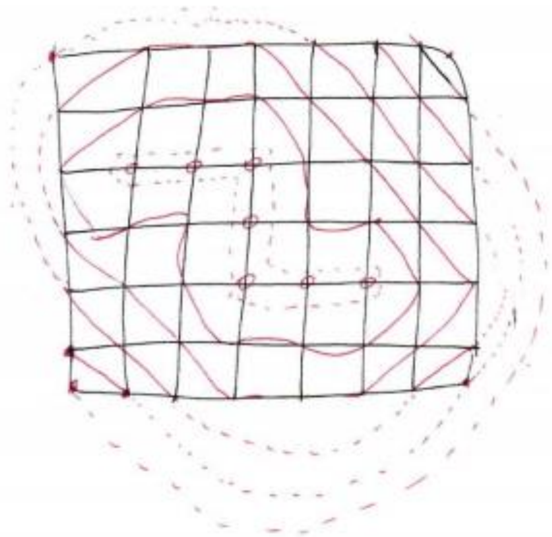

Figure 3.23: Data injection consists of a subgraph of G

Our objective is to propose a general algorithm framework to tackle this situation and give quantity model analysis. This algorithm is named as ***Two Phase Scheduling Algorithm (TPSA)***.

Algorithm 1 Two Phase Scheduling Algorithm (TPSA)

$global_s$:

Collapse the data injection processors into one "big" equivalent processor[3].

Calculate $m * n$ processor's D_i

Obtain the flow matrix A_i.

Calculate $m * n$ processors data fraction α_i

$local_s$:

Re-distributing workload between the data injection processors.

- The time complexity is $O(k * m * n)$.
- The time to calculate each determinant is $O(n^3)$ with Gaussian elimination or LU decomposition. So the total determinant time complexity is $O(k * n^3)$.
- The total time complexity is $O(k * n^3)$. Nonetheless, in real case, the time complexity is fast enough.

For example, Fig. 3.22's ***flow matrix*** is :

$$\begin{bmatrix} 4 & 8 & 12 & 10 & 6 & 2 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.87}$$

The simulation result illustrates as follows:

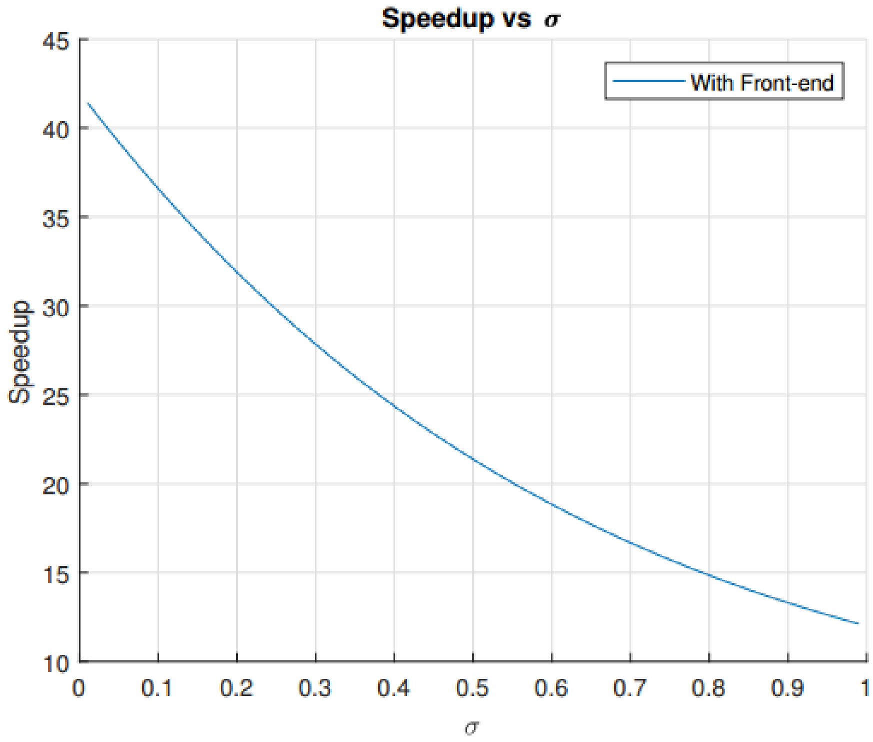

Figure 3.24: Speedup vs σ

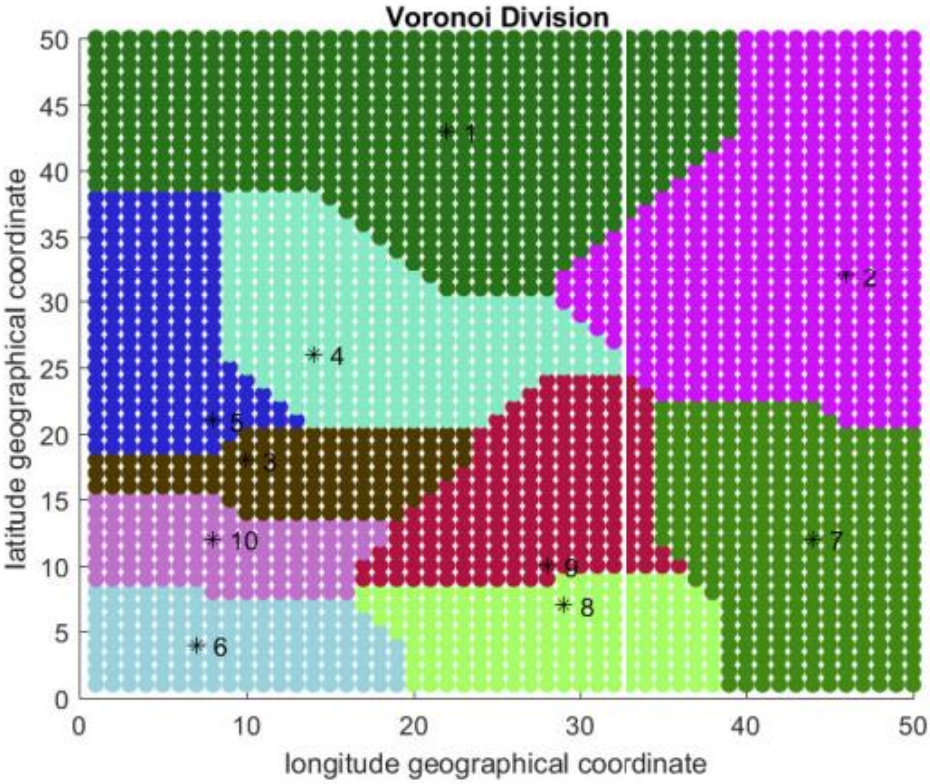

Figure 3.28: 10 Voronoi Cells

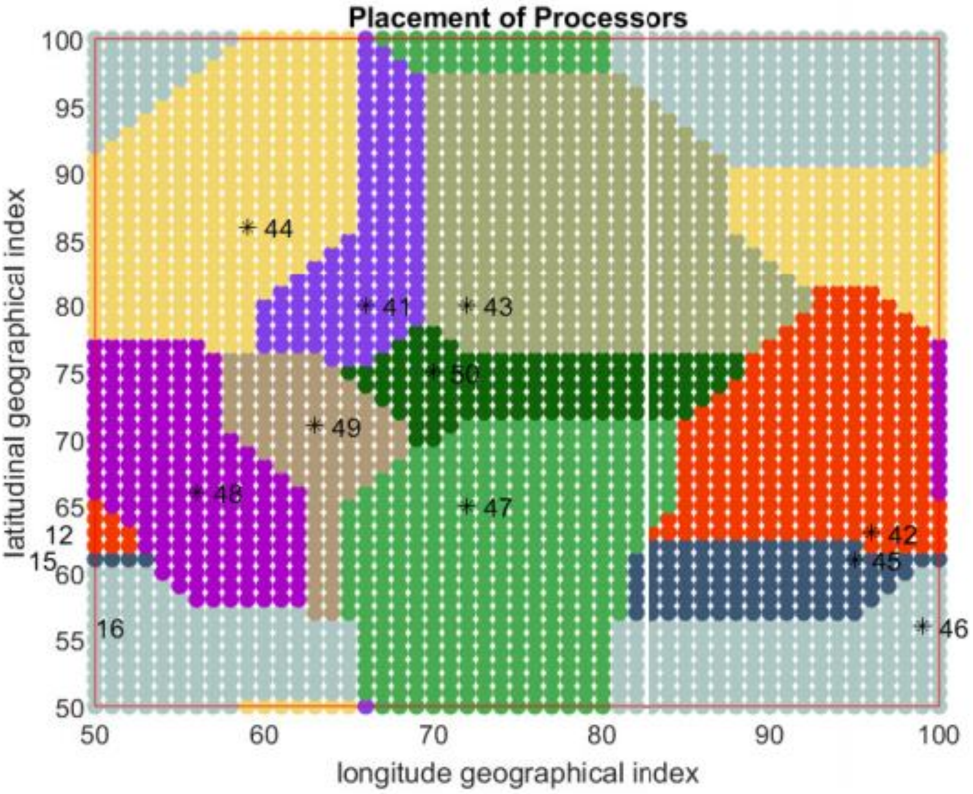

Figure 4.8: Torus Voronoi Diagram

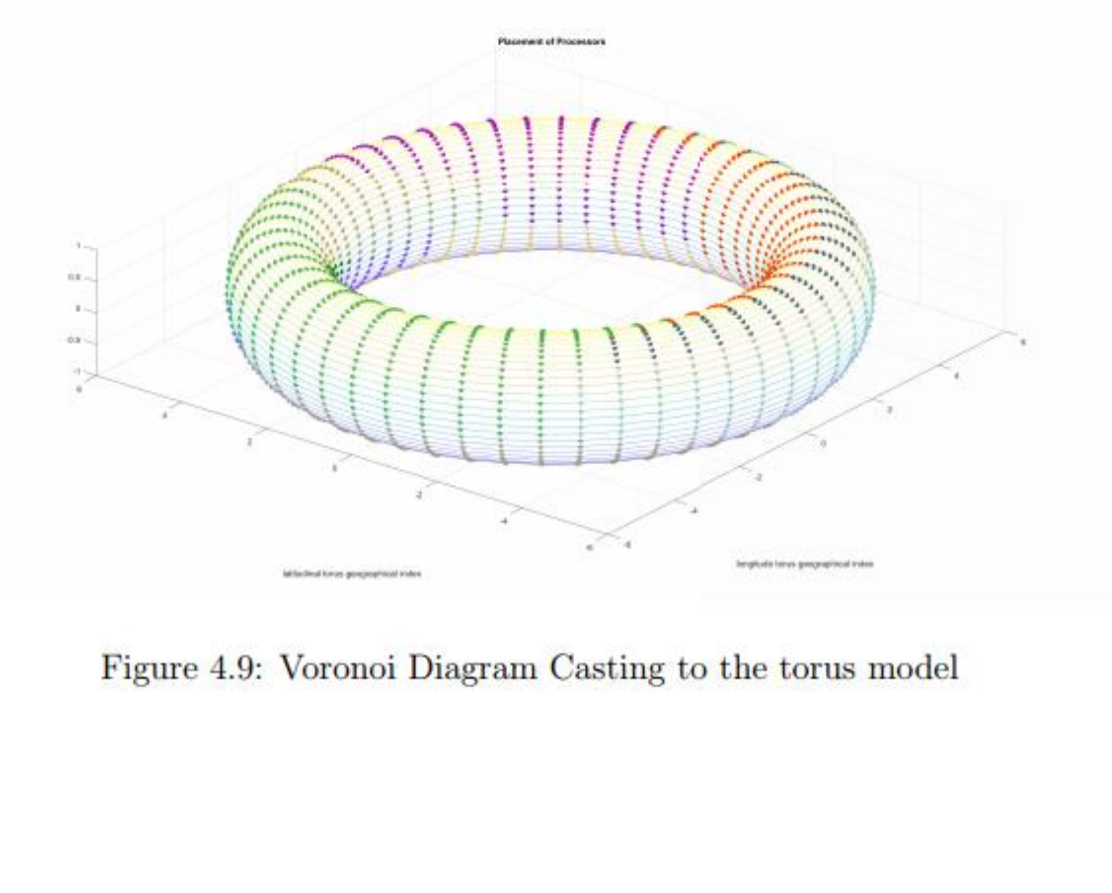

Figure 4.9: Voronoi Diagram Casting to the torus model

Algorithm 2 Reduced Voronoi Diagram Algorithm(RVDA)

$global_s$:

Define the cell's flow matrix column's number is $depth$.

Calculate k Voronoi cells with Manhattan distance.

Calculate k radius R_i of n Voronoi cells.

Calculate each cell's flow matrix A_i.

Set $depth_{min} = \min Speedup_i$'s R_i.

while $1 \leq i \leq k$ **do**

  $tempdepth_i = R_i$

  while $depth_{min} \leq j \leq tempdepth_i$ **do**

    Binary Search the value $\min(j)$, the flow matrix $\hat{A}_i$'s speedup $>$ $\min(speedup_{min})$.

    j = j + 1.

  end while

  Calculate the Reduced Voronoi cells by setting the $depth_i = depth_{min}$ in each cell.

  Calculate Voronoi cell's flow matrix A_i.

  $i = i + 1$

end while

$local_s$:

Display each reduced Voronoi cells.

Illustrate each reduced Voronoi cells' speedup curves

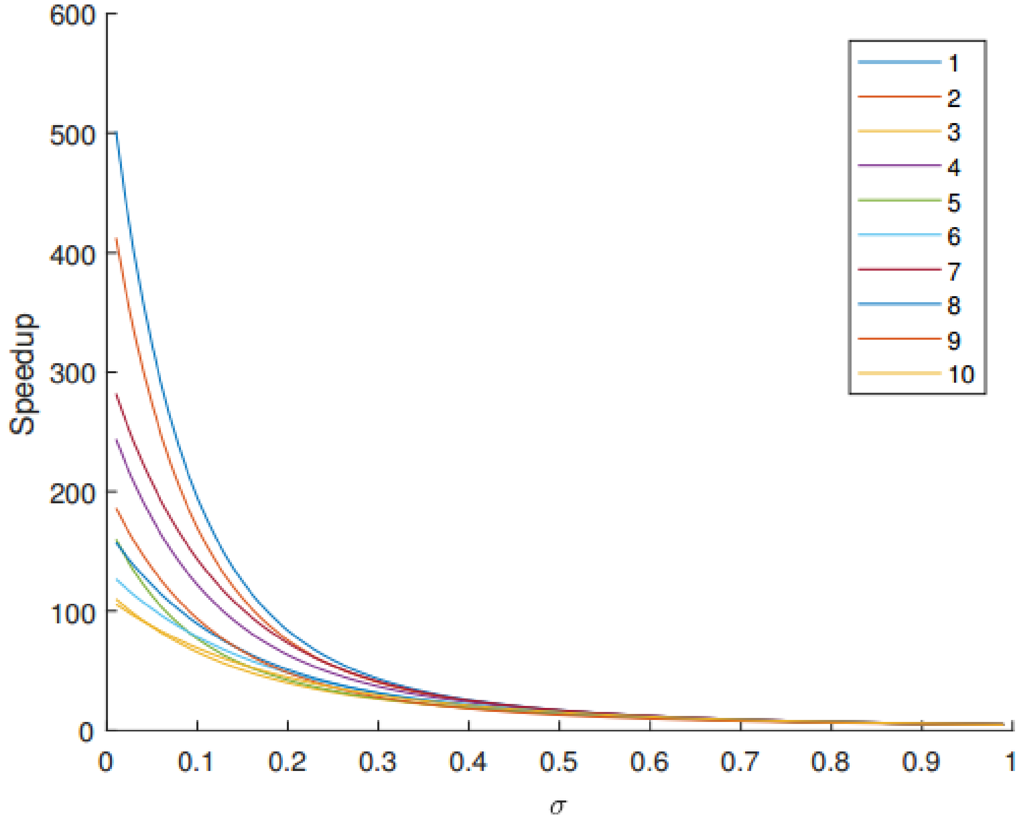

Figure 3.29: 10 Voronoi Cells speedup curves

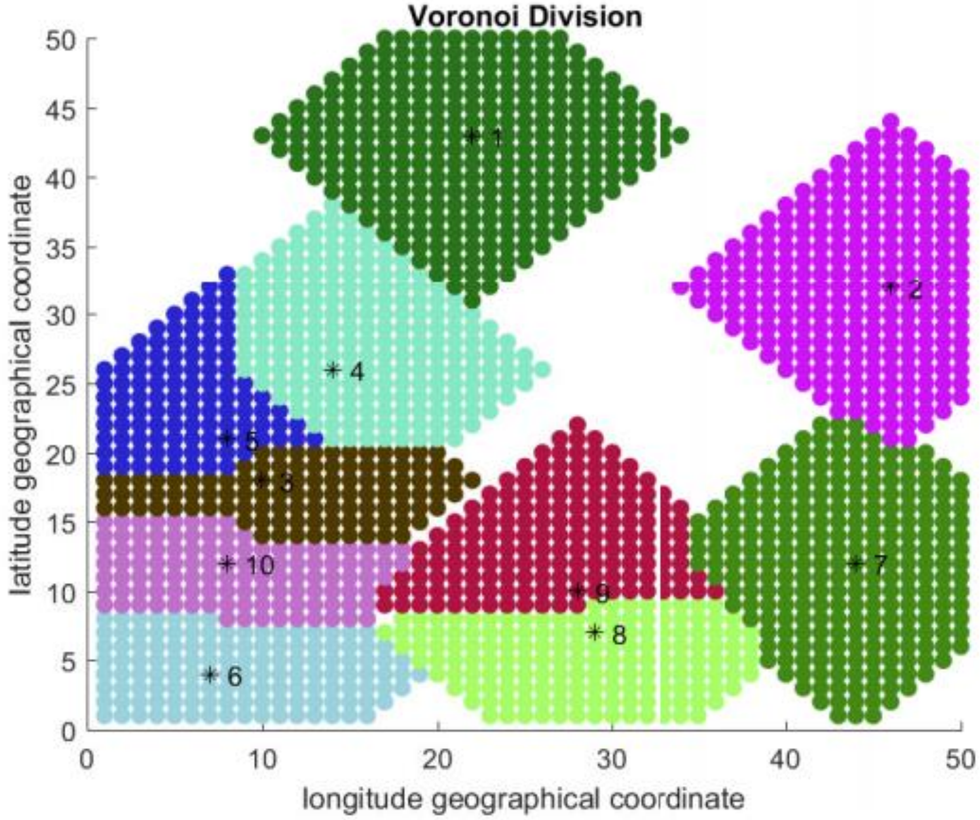

Figure 3.30: 10 reduced Voronoi cells

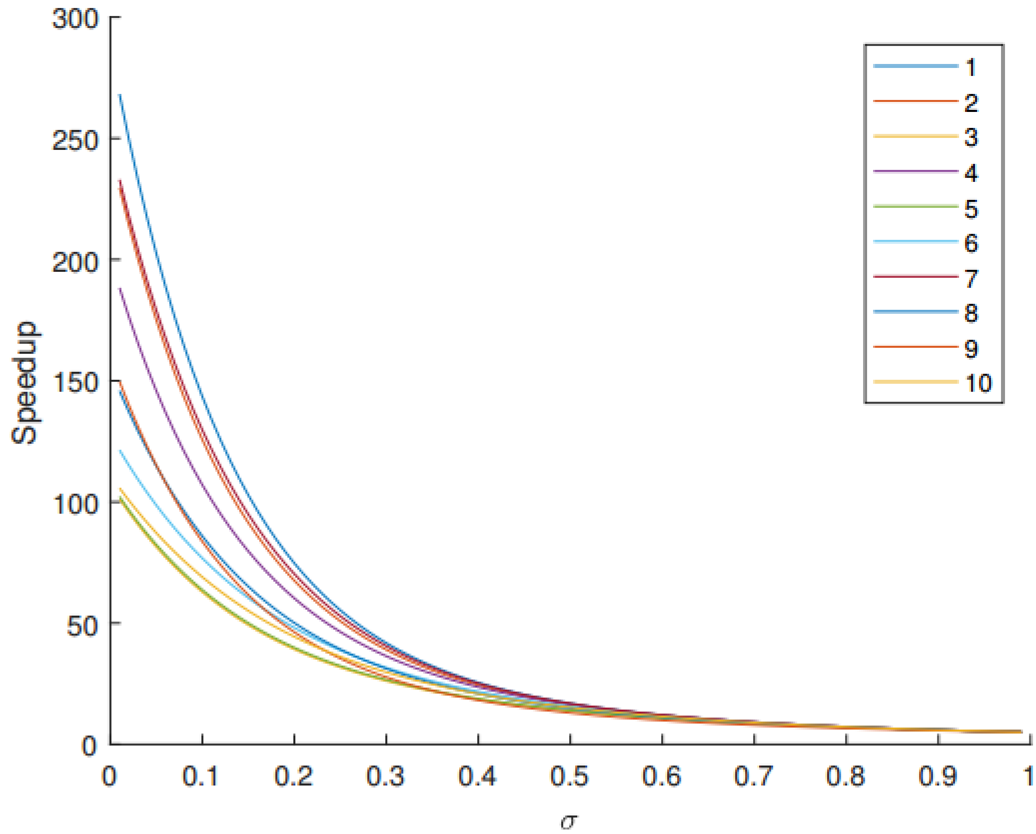

Figure 3.31: 10 reduced Voronoi cells's speedup curves

Algorithm 4 Improved Reduced Voronoi Diagram Algorithm(IRVDA)

$global_s$:

Define the cell's flow matrix column's number is $depth$.

Collapse the data injection processors into one "big" equivalent processor.

Calculate k constrained Voronoi cells[12] with Manhattan distance and get k cells.

Calculate k radius R_i of k Voronoi cells.

while $1 \leq i \leq k$ **do**

  Calculate the speedup $Speedup_i$ with flow matrix A_i.

  $i = i + 1$

end while

Set the $depth_{min} = \min(Speedup_i)$'s Radius

while $1 \leq i \leq n$ **do**

  $tempdepth_i = R_i$.

  while $depth_{min} \leq j \leq tempdepth_i$ **do**

    Binary Search the value $\min(j)$, the flow matrix $\hat{A}_i$'s speedup $>$ $\min(Speedup_{min})$.

    break.

    j = j + 1.

  end while

  Calculate Voronoi cell's flow matrix $\hat{A}_i$.

  $i = i + 1$

end while

$local_s$:

Display each reduced Voronoi cells.

Illustrate each reduced Voronoi cells speedup curves

- The Manhattan Voronoi cells' time complexity is $O(k * m * n)$;
- The binary search find the min j's time complexity is $O(k*(\log_2 \max(m, n)))$.
- So the total time complexity is $O(k * m * n)$.