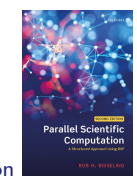
Parallel Sparse Matrix–Vector Multiplication

Section 4.3 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

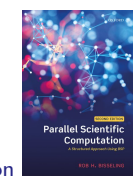
Utrecht University



Distribution option 1: represent first, distribute later

First build a global data structure to represent the sparse matrix, then distribute it over the processors:

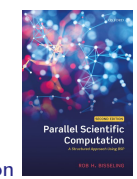
- A parallelizing compiler would do this.
- It requires global collaboration between the processors.
- Simple operations become complicated: in a linked list, 3 processors must work together and communicate to insert a new nonzero.



Distribution option 2: distribute first, represent later

Distribute the matrix first, and then let each processor represent the local nonzeros in a local data structure:

- This assigns subsets of nonzeros to processors.
- The subsets form a partitioning of the nonzero set: subsets are disjoint and together they contain all nonzeros.
- Sequential sparse data structures can be used.
- Simple operations remain simple: insertion and deletion are local operations without communication.
- This is the preferred approach.

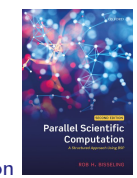


Most general matrix distribution

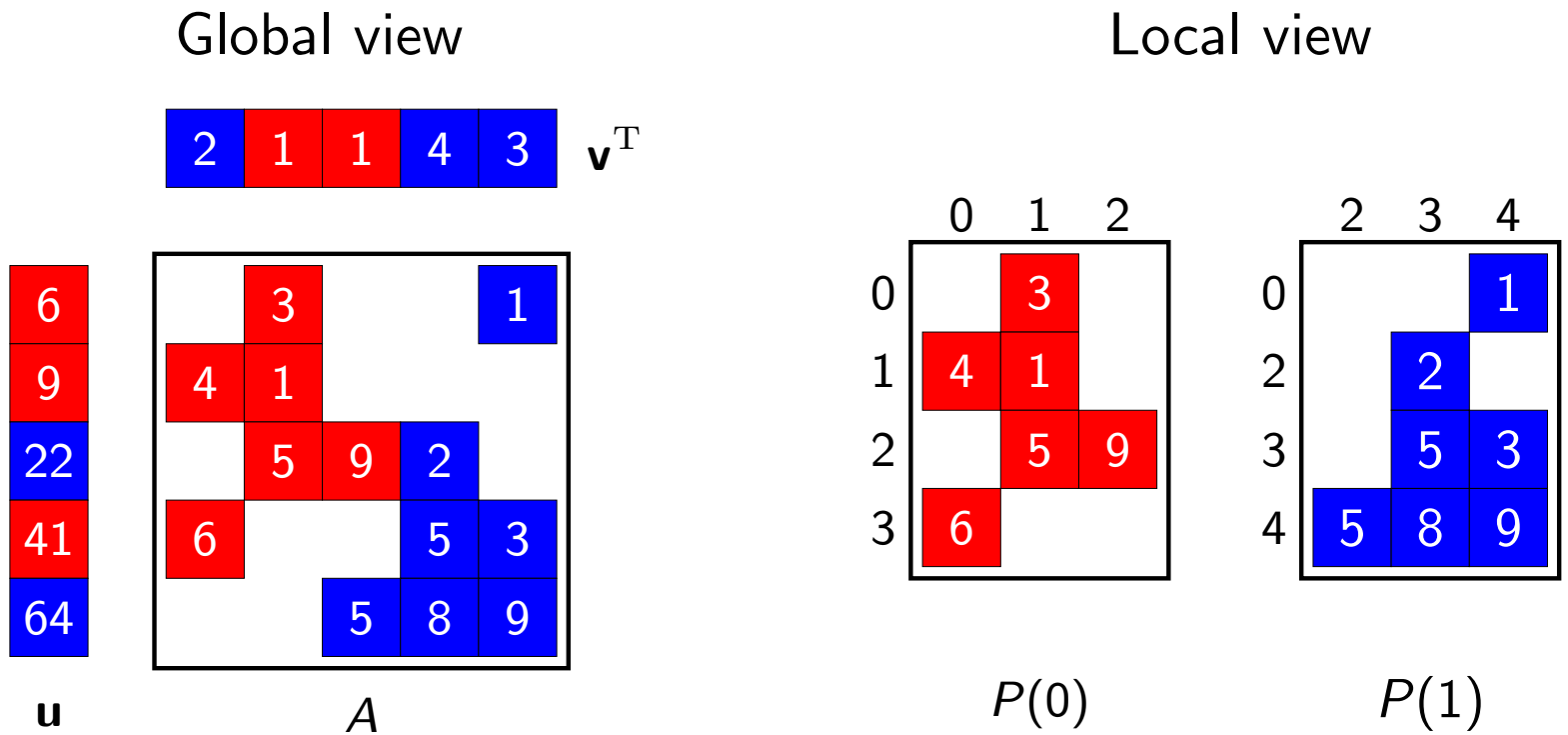
- The most general scheme maps nonzeros to processors,

$$a_{ij} \longmapsto P(\phi(i,j)), \text{ for } 0 \leq i,j < n \quad \text{and} \quad a_{ij} \neq 0,$$

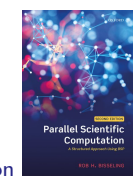
  where $0 \leq \phi(i,j) < p$.
- Zeros are not assigned to processors. For convenience, we define $\phi(i,j) = -1$ if $a_{ij} = 0$.
- Here, we use a 1D processor numbering.



Distribution of matrix and vectors



- $p = 2$, $n = 5$, $nz = 13$.
- $P(0)$ red cells, $P(1)$ blue cells.
- Matrix distribution is non-Cartesian.

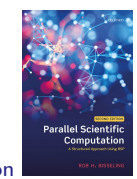


Where to compute $a_{ij} \cdot v_j$?

- Usually, there are many more nonzeros than vector components,

$$nz(A) \gg n,$$

  so we should move the vector components v_j to the nonzeros a_{ij}, not the other way round.
- Add the local products $a_{ij}v_j$ belonging to the same row i. The resulting partial sum on $P(s)$ is the local contribution u_{is} to u_i.
- Send u_{is} to the owner of u_i.
- Thus, we do not communicate elements of A, but only components of $\mathbf{v}$ and contributions to components of $\mathbf{u}$.

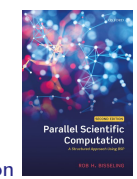


How to distribute the vectors?

- We map vector components to processors by

$$u_i \longmapsto P(\phi_{\mathbf{u}}(i)), \text{ for } 0 \leq i < n.$$

- In many iterative solvers, the same vector is repeatedly multiplied by a matrix A.
- Usually, a few vector operations are interspersed, such as DAXPYs $\mathbf{y} := \alpha\mathbf{x} + \mathbf{y}$ or inner products $\alpha := \mathbf{x}^{\mathrm{T}}\mathbf{y}$.
- All vectors should then be distributed in the same way: $\mathrm{distr}(\mathbf{u}) = \mathrm{distr}(\mathbf{v})$ for the operation $\mathbf{u} := A\mathbf{v}$.

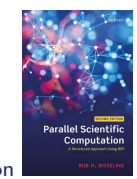


Different distributions for input and output vectors

- The Hyperlink-Induced Topic Search (HITS) algorithm by Jon Kleinberg repeatedly computes $A^{\mathrm{T}}A\mathbf{v}$ instead of $A\mathbf{v}$, where A is a web link matrix, to obtain two page-ranking values:
  - the authority value u_i which says how authoritative page i is as a source of information,
  - the hub value v_i which says in how far its hyperlinks point to authorities.
- An authority vector $\mathbf{u}$ can be computed from a hub vector $\mathbf{v}$ by $\mathbf{u} := A\mathbf{v}$, and vice versa by $\mathbf{v} := A^{\mathrm{T}}\mathbf{u}$.
- In this case, the output vector for A is taken as the input vector for A^{T}, and we do not need to revert immediately to the input distribution, allowing

$$\mathrm{distr}(\mathbf{u}) \neq \mathrm{distr}(\mathbf{v}).$$

J. M. Kleinberg, *Journal of the ACM*, **46**(5) (1999) pp. 604–632.

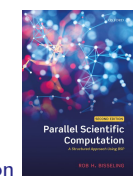


Deriving a parallel algorithm

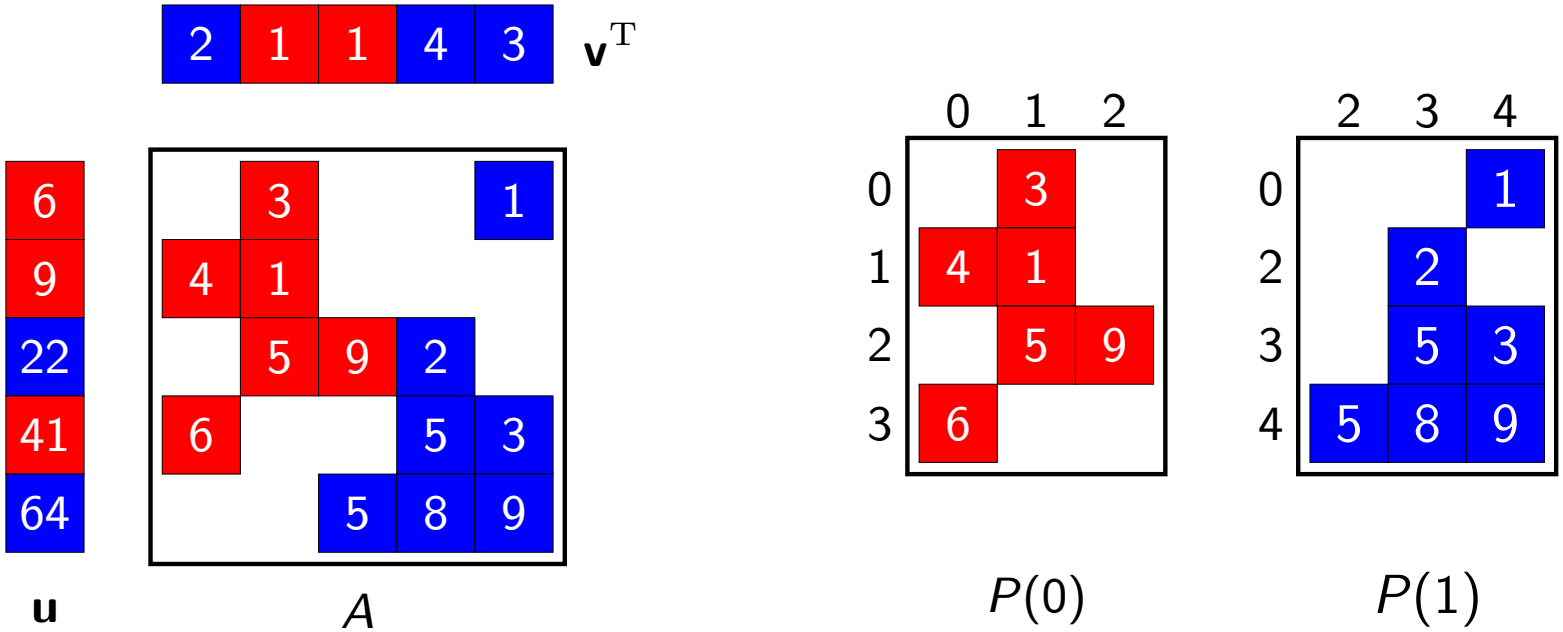
- Once we have chosen the data distribution and decided to compute the products $a_{ij}v_j$ on the processor that owns a_{ij}, the parallel algorithm follows naturally.
- The main computation for processor $P(s)$ is multiplying each local nonzero element a_{ij} by v_j and adding the result into a local partial sum,

$$u_{is} = \sum_{\substack{j=0 \\ \phi(i,j)=s}}^{n-1} a_{ij}v_j,$$

- Sparsity is exploited because
  - only terms with $a_{ij} \neq 0$ are summed;
  - only local partial sums u_{is} are computed for which $\{j : 0 \leq j < n \wedge \ \phi(i,j) = s\} \neq \emptyset$.



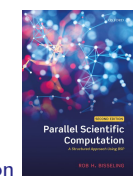
Row index set



- The index set of rows that are locally nonempty in $P(s)$ is

$$I_s = \{i : 0 \leq i < n \wedge (\exists j : 0 \leq j < n \wedge \phi(i,j) = s)\}.$$

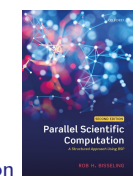
- On $P(0)$, row 4 is empty, so $I_0 = \{0, 1, 2, 3\}$
- On $P(1)$, row 1 is empty, so $I_1 = \{0, 2, 3, 4\}$.



Local sparse matrix–vector multiplication

$$I_s = \{i : 0 \leq i < n \wedge (\exists j : 0 \leq j < n \wedge \phi(i,j) = s)\}$$

{ Local sparse matrix–vector multiplication } ▷ Superstep (1)
for all $i \in I_s$ **do**
    $u_{is} := 0$;
    for all $j : 0 \leq j < n \wedge \phi(i,j) = s$ **do**
        $u_{is} := u_{is} + a_{ij} v_j$;

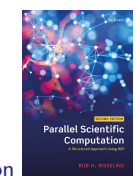


Data structure for local sparse matrix

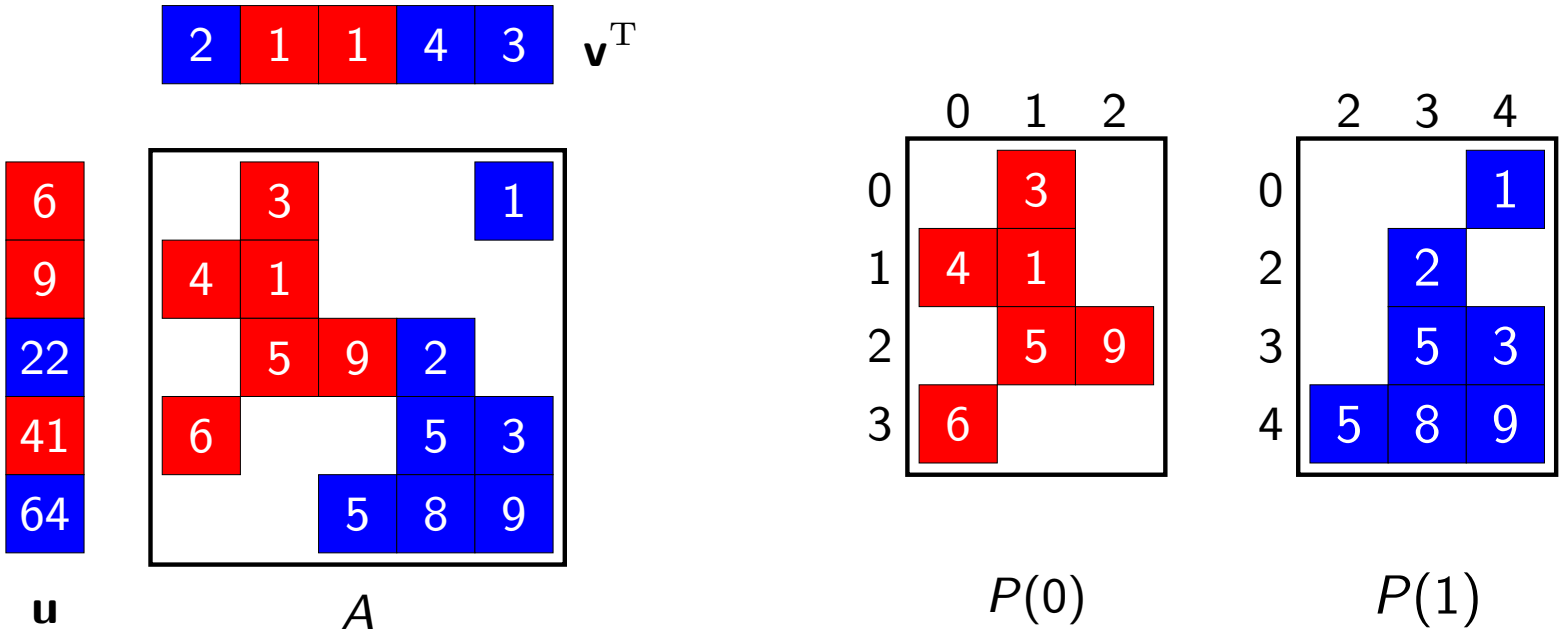
- Compressed row storage (CRS) suits row-oriented local sparse matrix–vector multiplication.
- CRS must be adapted to avoid overhead of many empty rows, which typically occurs if $c \ll p$.
- We number the nonempty local rows from 0 to $|I_s| - 1$. The corresponding indices `i` are the local indices.
- The original global indices from the set I_s are stored in increasing order in an array *rowindex* of length $|I_s|$:

$$i = \mathit{rowindex}[\mathtt{i}].$$

- The address of the first local nonzero of row i is $\mathit{start}[\mathtt{i}]$.



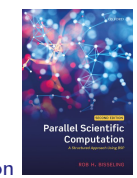
Column index set



- The index set J_s of columns that are locally nonempty in $P(s)$ is

$$J_s = \{j : 0 \leq j < n \wedge (\exists i : 0 \leq i < n \wedge \phi(i,j) = s)\}.$$

- $J_0 = \{0,1,2\}$ and $J_1 = \{2,3,4\}$.

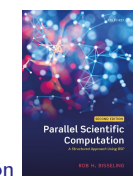


Fanout

$$J_s = \{j : 0 \leq j < n \wedge (\exists i : 0 \leq i < n \wedge \phi(i,j) = s)\}$$

{ Fanout } ▷ Superstep (0)
for all $j \in J_s$ **do**
    get v_j from $P(\phi_{\mathbf{v}}(j))$;

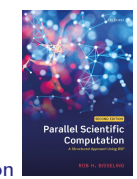
- The receiver knows (from its index set J_s) that it needs the vector component v_j. The sender is unaware of this. Thus, the receiver initiates the communication by using a 'get'.
- In dense algorithms, communication patterns are predictable and thus known to every processor, so that we only need 'put' primitives.
- In sparse algorithms, we also have to use 'get' primitives.
- In sparse programs, we must know the exact address where to get the data. This may require additional preprocessing.



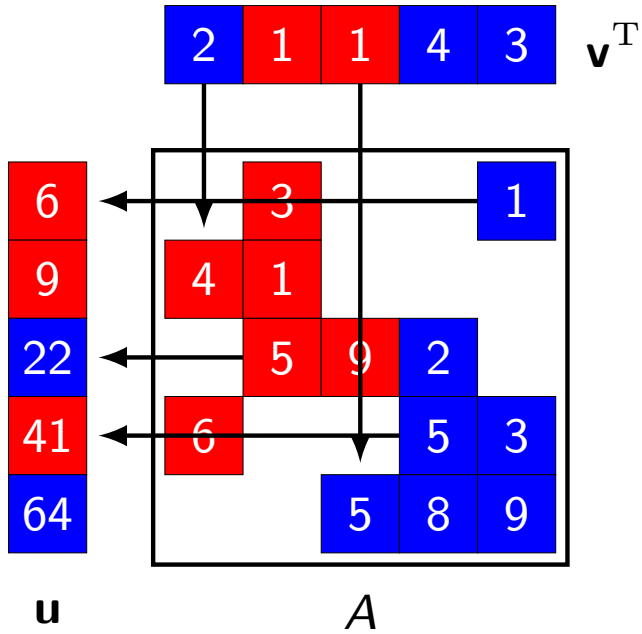
Fanin and final summation

{ Fanin } ▷ Superstep (2)
for all $i \in I_s$ **do**
    put u_{is} in $P(\phi_{\mathbf{u}}(i))$;

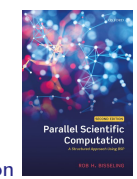
{ Summation of nonzero partial sums } ▷ Superstep (3)
for all $i : 0 \leq i < n \wedge \phi_{\mathbf{u}}(i) = s$ **do**
    $u_i := 0$;
    for all $t : 0 \leq t < p \wedge u_{it} \neq 0$ **do**
        $u_i := u_i + u_{it}$;



Communication



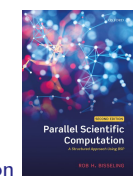
- Vertical arrows: communication of vector components v_j.
- v_0 is sent from $P(1)$ to $P(0)$, because of the nonzeros $a_{10} = 4$ and $a_{30} = 6$ owned by $P(0)$.
- v_1, v_3, v_4 need not be sent.
- Horizontal arrows: communication of partial sums u_{is}.



Cost analysis

Wat kost het? (Dutch for: How much does it cost?)

- The answer depends on the matrix A and the distributions $\phi, \phi_{\mathbf{v}}, \phi_{\mathbf{u}}$.
- We can obtain an upper bound on the BSP cost, assuming that:
  - the matrix nonzeros are evenly spread over the processors, each processor having $\frac{cn}{p}$ nonzeros;
  - the vector components are also evenly spread, each processor having $\frac{n}{p}$ components.



Cost of supersteps (0), (1)

- Superstep (0): $P(s)$ must receive at most all n components v_j, but not the $\frac{n}{p}$ local components, so that

$$h_{\mathrm{r}} = n - \frac{n}{p}.$$

Furthermore,

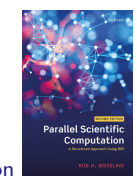
$$h_{\mathrm{s}} = \frac{n}{p}(p-1),$$

because the $\frac{n}{p}$ local components must be sent to all the other $p-1$ processors. Therefore,

$$T_{(0)} = (1 - \frac{1}{p})ng + l.$$

- Superstep (1): 2 flops are needed for each local nonzero.

$$T_{(1)} = \frac{2cn}{p} + l.$$



Cost of supersteps (2), (3)

- Superstep (2): Similar to (0).

$$T_{(2)} = (1 - \frac{1}{p})ng + l.$$

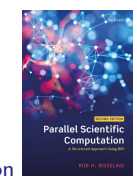
- Superstep (3): Each of the $\frac{n}{p}$ local vector components is computed by adding at most p partial sums.

$$T_{(3)} = n + l.$$

- The total BSP cost under the equal-spreading assumptions is bounded by

$$T_{\mathrm{MV}} \leq \frac{2cn}{p} + n + 2(1 - \frac{1}{p})ng + 4l.$$

- This bound may be overly pessimistic for particular distributions that are well-tailored to the matrix.

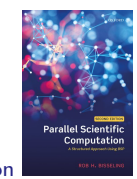


Efficient computation

$$T_{\mathrm{MV}} \leq \frac{2cn}{p} + n + 2(1 - \frac{1}{p})ng + 4l.$$

- Computation can be considered efficient if $\frac{2cn}{p} > 2ng$, i.e., $c > pg$. But this happens rarely: only for very dense matrices.
- The number of nonzeros per row c, and not the density d, determines the efficiency directly.
- To make the computation efficient for smaller c, we can:
  - use a Cartesian distribution and exploit its 2D nature;
  - refine the general distribution using an automatic procedure to detect the underlying matrix structure;
  - exploit properties of specific matrix classes, such as random sparse matrices and Laplacian matrices.

  This will be done later on.

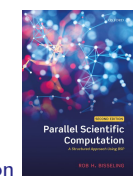


Communication volume

- The communication volume V of an algorithm is the total number of data words sent.
- For the parallel SpMV, V depends on ϕ, $\phi_{\mathbf{u}}$, $\phi_{\mathbf{v}}$.
- For a given ϕ, we can count
  - the number of processors λ_i with a nonzero a_{ij} in matrix row i (at least $\lambda_i - 1$ processors must send a contribution u_{is});
  - the number of processors μ_j with a nonzero a_{ij} in matrix column j (at least $\mu_j - 1$ processors must receive v_j).
- Thus, we obtain a lower bound V_ϕ on V,

$$V_\phi = \sum_{\substack{i=0 \\ \lambda_i \geq 1}}^{n-1} (\lambda_i - 1) + \sum_{\substack{j=0 \\ \mu_j \geq 1}}^{n-1} (\mu_j - 1).$$

- An upper bound is $V_\phi + 2n$, because in the worst case all n components u_i are owned by processors without a nonzero in row i, and similar for the components v_j.



Summary

- Distribute first, represent later.
- The most general mapping of nonzeros and vector components to processors is:

$$a_{ij} \longmapsto P(\phi(i,j)), \text{ for } 0 \leq i,j < n \quad \text{and} \quad a_{ij} \neq 0,$$
$$u_i \longmapsto P(\phi_{\mathbf{u}}(i)), \text{ for } 0 \leq i < n.$$

- We have derived a parallel algorithm with 4 supersteps: fanout, local matrix–vector multiplication, fanin, summation of partial sums.
- The row index set I_s and column index set J_s are used for exploiting the sparsity in the algorithm.
- A lower bound on the total communication volume is

$$V_\phi = \sum_{\substack{i=0 \\ \lambda_i \geq 1}}^{n-1} (\lambda_i - 1) + \sum_{\substack{j=0 \\ \mu_j \geq 1}}^{n-1} (\mu_j - 1).$$