Data Distribution

Albert-Jan Yzelman

15th of November, 2013

Derived from the book slides by Prof. dr. Rob H. Bisseling, found at

`www.math.uu.nl/people/bisseling/Education/PA/pa.html`

Data dependencies

- Data must be distributed for scalability. **Two objectives**:
  1. load balance
     (scalability for computation supersteps)
  2. minimise h-relations
     (i.e., minimise the overhead of communication supersteps).
- If data is mostly independent, then no communication between processes is necessary and optimising for load-balance is all we need to do (**embarrasingly parallel**).
- If computation on one data element requires input depending on all other data elements, then high communication costs (**all-to-all**s) are unavoidable.

Data dependencies

- Data must be distributed for scalability. **Two objectives**:
  1. load balance
     (scalability for computation supersteps)
  2. minimise h-relations
     (i.e., minimise the overhead of communication supersteps).
- If data is mostly independent, then no communication between processes is necessary and optimising for load-balance is all we need to do (**embarrasingly parallel**).
- If computation on one data element requires input depending on all other data elements, then high communication costs (**all-to-all**s) are unavoidable.
- Many applications are in-between these two extremes. Then load balancing and minimisation of h-relations, the two objectives of data distribution, **are in conflict** with each other.

Sparse matrix–vector multiplication

Parallel sparse matrix–vector multiplication $\mathbf{u} := A\mathbf{v}$, with

- A sparse $m \times n$ matrix,
- $\mathbf{u}$ dense m-vector,
- $\mathbf{v}$ dense n-vector;

computes

$$u_i := \sum_{j=0}^{n-1} a_{ij} v_j$$

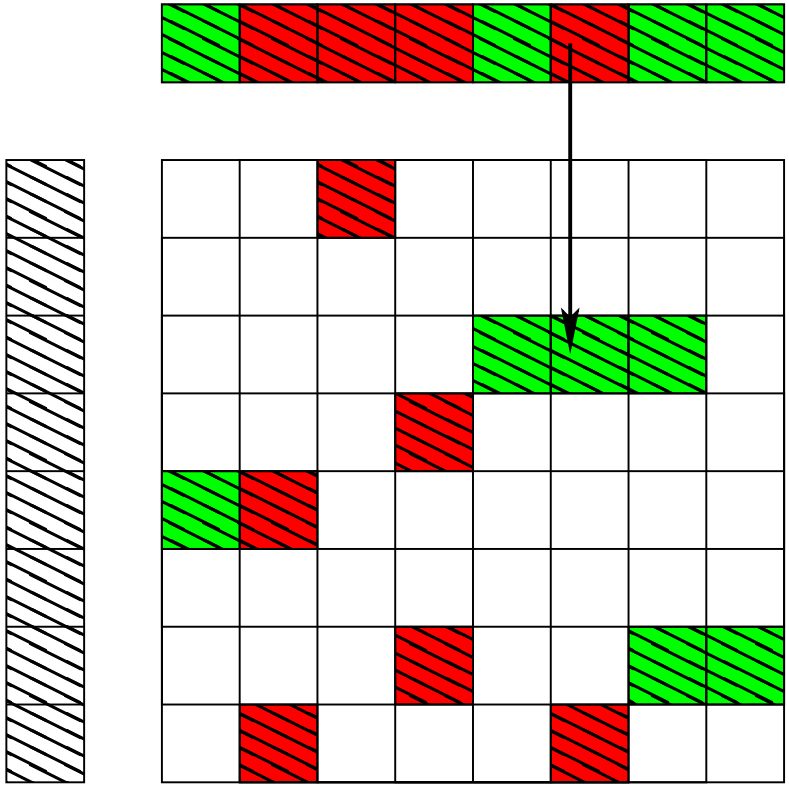
Four supersteps:

1. communicate (fan-out),
2. compute (local SpMV),
3. communicate (fan-in),
4. compute (handle remote contributions).

Sparse matrix–vector multiplication

Four supersteps:

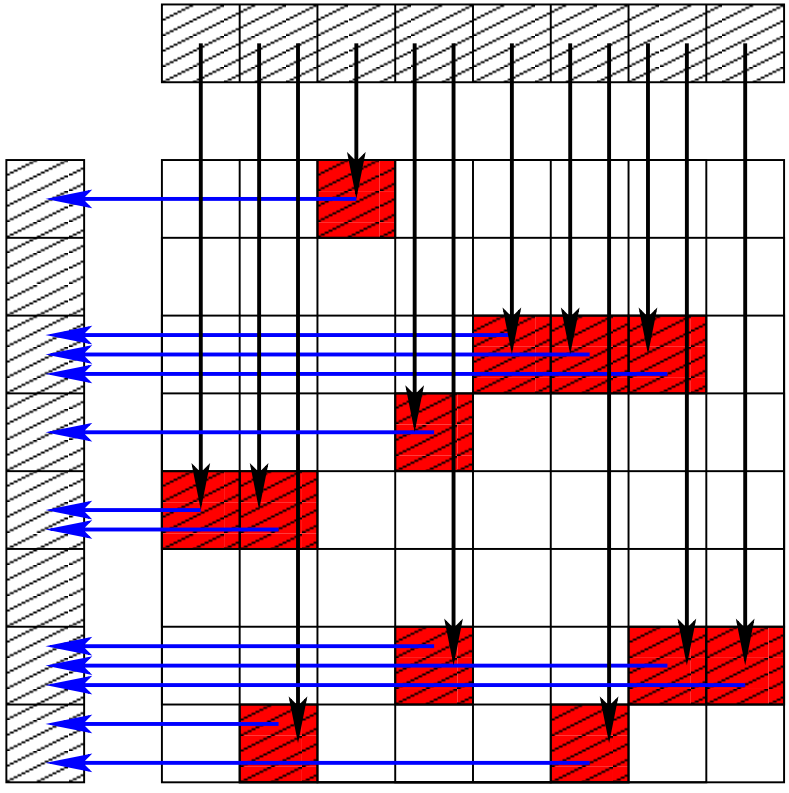
1. fan-out,
2. local SpMV,
3. fan-in,
4. handle remote contributions.



Sparse matrix–vector multiplication

Four supersteps:

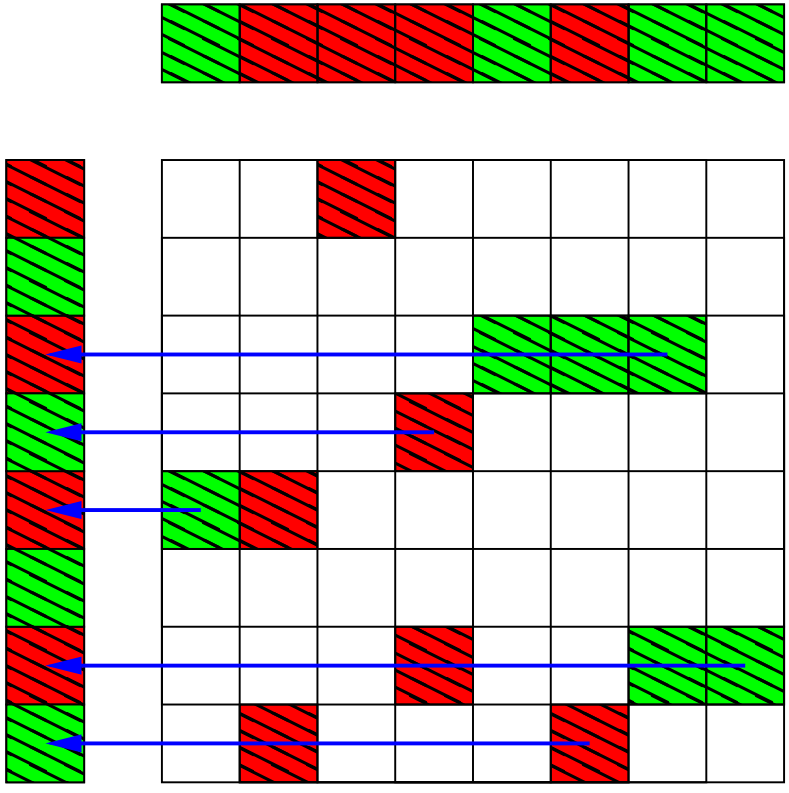
1. fan-out,
2. local SpMV,
3. fan-in,
4. handle remote contributions.



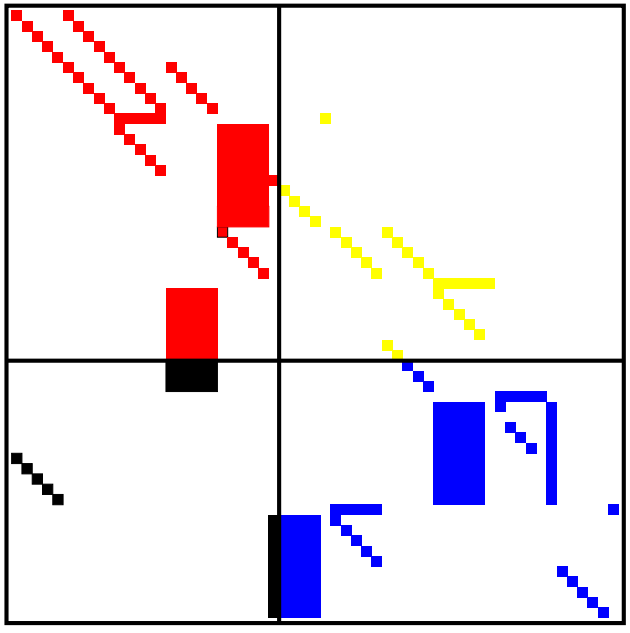
Sparse matrix–vector multiplication

Four supersteps:

1. fan-out,
2. local SpMV,
3. fan-in,
4. handle remote contributions.

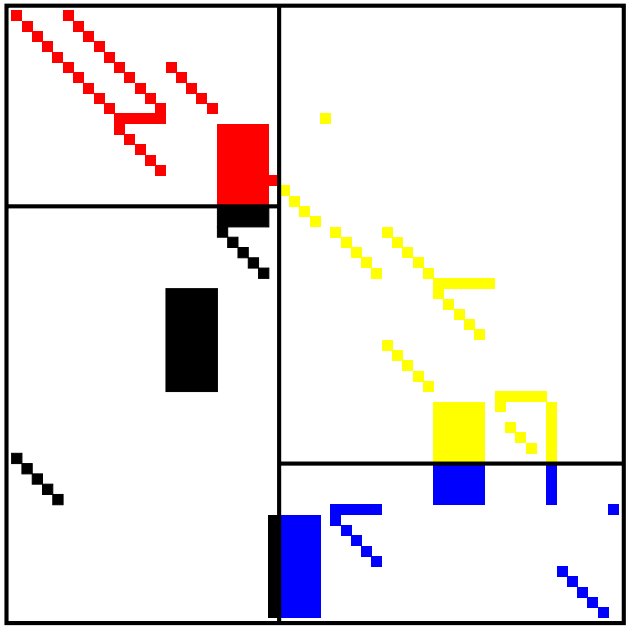


Cartesian matrix partitioning



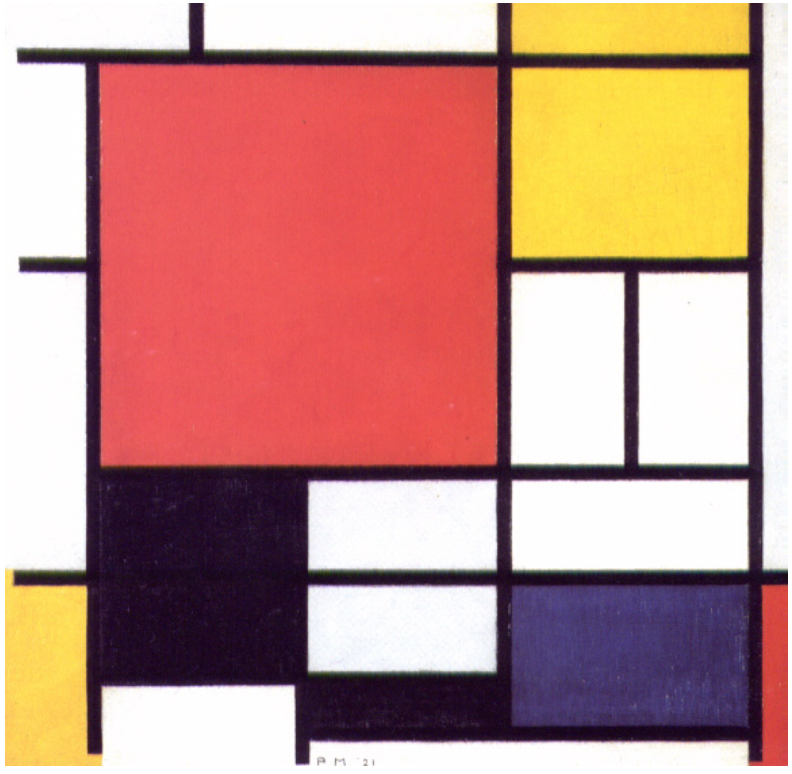
- Block distribution of 59×59 matrix `impcol_b` from Harwell–Boeing collection with 312 nonzeros, for $p = 4$
- #nonzeros per processor: 126, 28, 128, 30
- Each separate split has optimal balance (for blocks)

Non-Cartesian matrix partitioning



- Block distribution of 59×59 matrix `impcol_b` from Harwell–Boeing collection with 312 nonzeros, for $p = 4$
- #nonzeros per processor: 76, 76, 80, 80
- Each separate split has optimal balance (for blocks)

Composition with Red, Yellow, Blue and Black



Piet Mondriaan 1921

Sparse matrix partitioning

Matrix distributions

Definition (Matrix distribution)

Let A an $m \times n$ sparse matrix, $I = \{0, 1, \ldots, m-1\}$, and $J = \{0, 1, \ldots, n-1\}$. A distribution of this matrix over p processes is a function

$$\phi : I \times J \to \{0, 1, \ldots, p-1\}.$$

Definition (Matrix distribution over a process grid)

Let A, I, J as before. Let M, N be integers. If the p processes are organised in an $M \times N$ grid such that $p = M \cdot N$, then a matrix distribution over this grid is a function

$$\phi : I \times J \to \{0, 1, \ldots, M-1\} \times \{0, 1, \ldots, N-1\}.$$

Matrix distributions

You have seen matrix distributions before (dense LU decomposition):

Definition (2D cyclic distribution over a process grid)

Let $p = MN, A, I, J$ as before. A **2D cyclic distribution** is given by

$$\phi(i,j) = (i \bmod M, j \bmod N).$$

Any distribution on a process grid can be reduced to a distribution unrelated to a process grid, e.g., by mapping (s_i, s_j) to $s = s_i N + s_j M$.

Definition (2D cyclic distribution)

Let p, A, I, J as before. Assume additionally that $p = M \cdot N$ for integer M, N. Then, a 2D cyclic distribution is given by

$$\phi(i,j) = (i \bmod M) \cdot N + j \bmod N.$$

Matrix distributions

Definition (Cartesian distribution over a process grid)

Let $p = MN, A, I, J$ as before. Let $\phi_i : I \to \{0, 1, \ldots, M-1\}$ and $\phi_j : J \to \{0, 1, \ldots, N-1\}$. Then, a 2D Cartesian distribution over an $M \times N$ process grid has the following form:

$$\phi(i, j) = (\phi_i(i), \phi_j(i)).$$

Again, there is no difference with the following definition:

Definition (Cartesian distribution)

Let $p = MN, A, I, J, \phi_i, \phi_j$ as before. A 2D Cartesian distribution has the form

$$\phi(i, j) = \phi_i(i)N + \phi_j(j).$$

p-way sparse matrix partitioning

For sparse matrix partitioning, we identify:

- **nonzero** $\equiv$ index pair;
- **sparse matrix** $\equiv$ set of index pairs.

Instead of thinking about functions, we can also think about sets:

- Define the **process-local sparse matrix** A_s

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

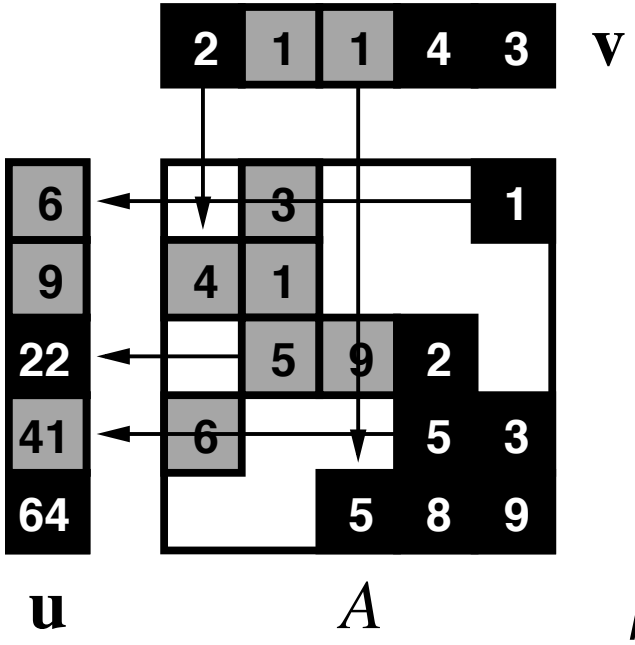
  as the set of nonzeroes local to process s, $0 \leq s < p$.
- The sets $A_0, \ldots, A_{p-1}$ form a p-**way partitioning** of

$$A = \{(i,j) : 0 \leq i,j < n \wedge a_{ij} \neq 0\}$$

  if all parts are mutually disjoint and include all nonzeroes:
  - $\cup_{k=0}^{p-1} A_k = A$, and
  - $\forall 0 \leq q, r < p$ we have that $A_q \cap A_r = \emptyset$.

Aims of sparse matrix partitioning

Parallel sparse matrix–vector multiplication $\mathbf{u} := A\mathbf{v}$

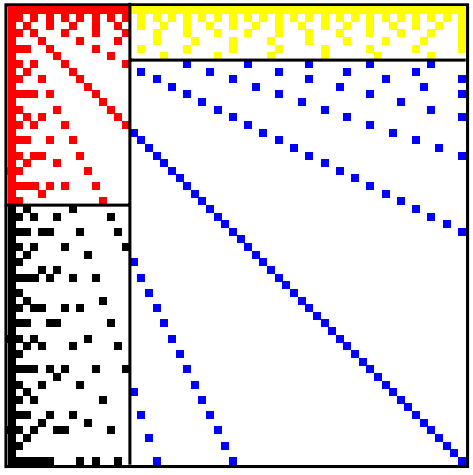


4 supersteps: communicate, compute, communicate, compute. Aims:

1. balance main computation step, and
2. minimise communication volume.

Here, the total communication volume V equals 5.

Communication volume for partitioned matrix



$$V(A_0, A_1, A_2, A_3) = V(A_0, A_1, A_2 \cup A_3) + V(A_2, A_3)$$

- $V(A_0, A_1, A_2, A_3)$ is the total matrix–vector communication volume corresponding to the partitioning A_0, A_1, A_2, A_3.
- $V(A_2, A_3)$ is the volume corresponding to the partitioning A_2, A_3 of the matrix $A_2 \cup A_3$.

Motivation of the Mondriaan splitting

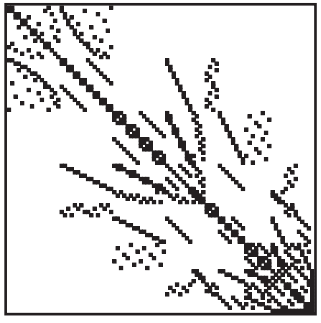
Theorem. Given an $m \times n$ sparse matrix A, and mutually disjoint subsets $A_0, \ldots, A_k$ of A, where $k \geq 1$, it holds that

$$V(A_0, \ldots, A_k) = V(A_0, \ldots, A_{k-2}, A_{k-1} \cup A_k) + V(A_{k-1}, A_k).$$

Meaning: k parts $\Rightarrow$ $k+1$ parts can be done locally, independently, by looking at just one split. This greedily minimises the total communication volume.

Computational load balance

- Paint all nonzeros black:



No communication, but no parallelism. No pain, no gain!

- A load balance criterion must therefore be satisfied:

$$\max_{0 \le s < p} nz(A_s) \le (1+\epsilon)\frac{nz(A)}{p}.$$

- ϵ is specified allowable imbalance;
  ϵ' is imbalance achieved by partitioning.

BSP cost determines ϵ

We now have a parameter ϵ. What is its best value?

- Communication cost is $\frac{Vg}{p}$, **assuming balanced communication**.
- Total BSP cost is

$$2(1+\epsilon')\frac{nz(A)}{p} + \frac{Vg}{p} + 4l.$$

- To get a good trade-off between computation imbalance and communication, we require

$$2\epsilon'\frac{nz(A)}{p} \approx \frac{Vg}{p}, \quad \text{i.e.,} \quad \epsilon' \approx \frac{Vg}{2nz(A)}.$$

- If necessary, we adjust ϵ and run the partitioner again.

Bipartitioning: splitting into 2 parts

$$A = \begin{bmatrix} 0 & 3 & 0 & 0 & 1 \\ 4 & 1 & 0 & 0 & 0 \\ 0 & 5 & 9 & 2 & 0 \\ 6 & 0 & 0 & 5 & 3 \\ 0 & 0 & 5 & 8 & 9 \end{bmatrix}.$$

- The number of possible 2-way partitionings is $2^{nz(A)-1} = 2^{12} = 4096$. (Symmetry saved a factor of 2.)
- Finding the best solution by enumeration, trying all possibilities and choosing the best, works only for small problems. Thus, we need heuristic methods.
- Splitting by columns restricts the search space to $2^{n-1} = 2^4 = 16$ possibilities. An optimal column split for $\epsilon = 0.1$ is $\{0, 1, 2\}$ — $\{3, 4\}$, with $V = 4$.

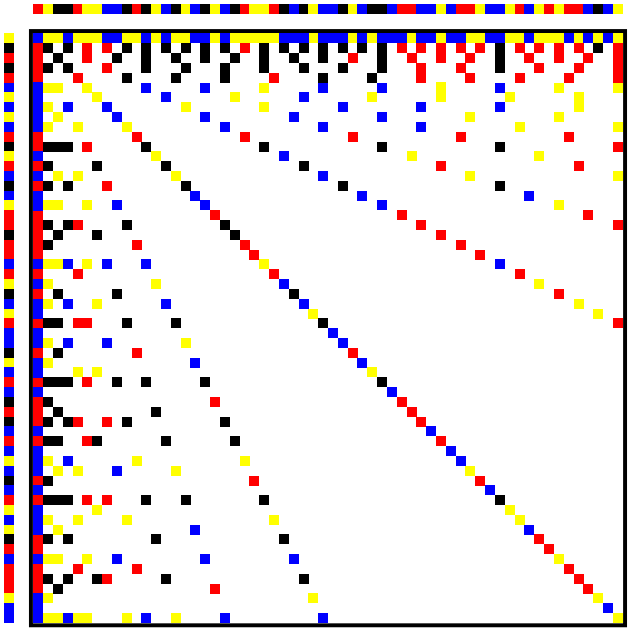
Repeated splits

Recursive bipartitioning: starts with a complete matrix, splits it into 2 submatrices, and recurses on each submatrix (until p parts are created). The maximum number of nonzeroes in one part is at most $(1+\epsilon)\frac{nz}{2}$. The 1:1 load balance ratio might shift if $p \neq 2^q$!

- Rows and columns in the submatrix need not be consecutive.
- A split in the column in direction can cause empty rows to appear in the submatrix (and vice versa).
- The final result for processor $P(s)$ is a local matrix A_s. This matrix is a submatrix of A that corresponds to the rows and columns of $\bar{I}_s \times \bar{J}_s$.
- Removing empty rows and columns from $\bar{I}_s \times \bar{J}_s$ gives $I_s \times J_s$. Thus

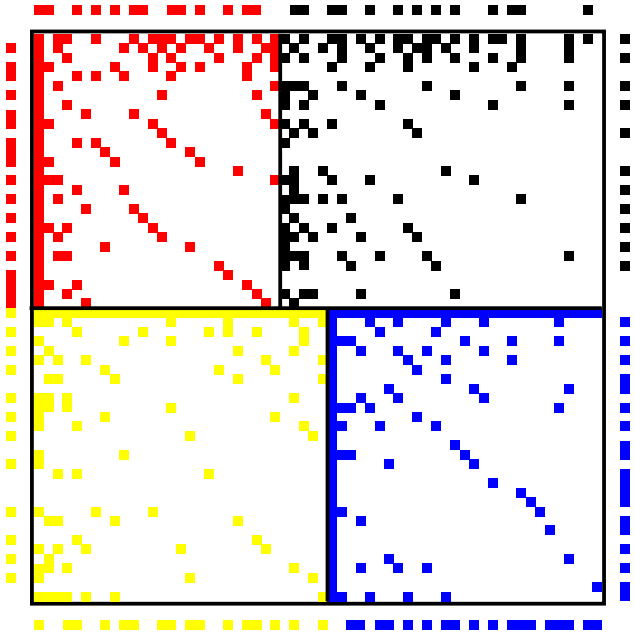
$$A_s \subset I_s \times J_s \subset \bar{I}_s \times \bar{J}_s.$$

Global view of matrix prime60



- Distribution of 60×60 matrix prime60 with 462 nonzeros, for $p = 4$, obtained by Mondriaan partitioning with $\epsilon = 3\%$.
- Maximum number of nonzeros per processor is 117; average is 462/4=115.5. Achieved imbalance is $\epsilon' \approx 1.3\%$.
- Communication volume is: fanout 51; fanin 47; $V = 98$.

Local view of matrix prime60



- The local submatrix $\bar{I}_s \times \bar{J}_s$ of processor $P(s)$ has size:
  - 29×26 for $P(0)$; 29×34 for $P(1)$
  - 31×31 for $P(2)$; 31×29 for $P(3)$
- Note that $\bar{I}_1 \times \bar{J}_1$ has 6 empty rows and 9 empty columns, giving a size of 23×25 for $I_1 \times J_1$.

Growth of load imbalance by splitting

- If the growth factor at each recursion level is $1 + \delta$, the overall growth factor is $(1 + \delta)^q \approx 1 + q\delta$. Here, $p = 2^q$. This motivates starting with $q\delta = \epsilon$, i.e., $\delta = \epsilon/q$.
- After the first split, one part has at least half the nonzeros, and the other part at most half. We recompute the ϵ values for both halves based on the new situation.
- The less-loaded part can increase the allowed load imbalance as its farther from its maximum load. This results in more freedom for the partitioner to reduce communication.

Recursive, adaptive bipartitioning algorithm

MatrixPartition(A, p, ϵ)

input: $p = 2^q$, $\epsilon =$ allowed load imbalance, $\epsilon > 0$.

output: p-way partitioning of A with imbalance $\leq \epsilon$.

if $p > 1$ **then**

$maxnz := (1 + \epsilon)\frac{nz(A)}{p}$;

$(B_0^{\text{row}}, B_1^{\text{row}}) := split(A, \text{row}, \frac{\epsilon}{q})$;

$(B_0^{\text{col}}, B_1^{\text{col}}) := split(A, \text{col}, \frac{\epsilon}{q})$;

if $V(B_0^{\text{row}}, B_1^{\text{row}}) \leq V(B_0^{\text{col}}, B_1^{\text{col}})$ **then**

$(B_0, B_1) := (B_0^{\text{row}}, B_1^{\text{row}})$;

else

$(B_0, B_1) := (B_0^{\text{col}}, B_1^{\text{col}})$;

...

Recursive, adaptive bipartitioning algorithm

MatrixPartition(A, p, ϵ)

input: $p = 2^q$, $\epsilon =$ allowed load imbalance, $\epsilon > 0$.

output: p-way partitioning of A with imbalance $\leq \epsilon$.

if $p > 1$ **then**

$\ldots$

$\epsilon_0 := \frac{maxnz}{nz(B_0)} \cdot \frac{p}{2} - 1$; $\epsilon_1 := \frac{maxnz}{nz(B_1)} \cdot \frac{p}{2} - 1$;

$(A_0, \ldots, A_{p/2-1}) :=$ MatrixPartition$(B_0, \frac{p}{2}, \epsilon_0)$;

$(A_{p/2}, \ldots, A_{p-1}) :=$ MatrixPartition$(B_1, \frac{p}{2}, \epsilon_1)$;

else

$A_0 := A$;

The magic *split* function

This clarifies the limitations of what the split can do;

- either rowwise or columnwise splits, and
- cannot return a bipartitioning that deviates more than ϵ from
- an ideal 1:1 split in terms of load-balance.

But how does it work, and how does it minimise communication?

Graphs and hypergraphs

To solve this multi-constraint optimisation problem, we use **hypergraphs**. We first introduce graphs:

Definition (Graph)

Let V be a set of **vertices** and $E = \{ \{v_i, v_j\} \mid v_{i,j} \in V \}$ a set of **edges**. Then $G = (V, E)$ is an **undirected graph**.

Graphs and hypergraphs

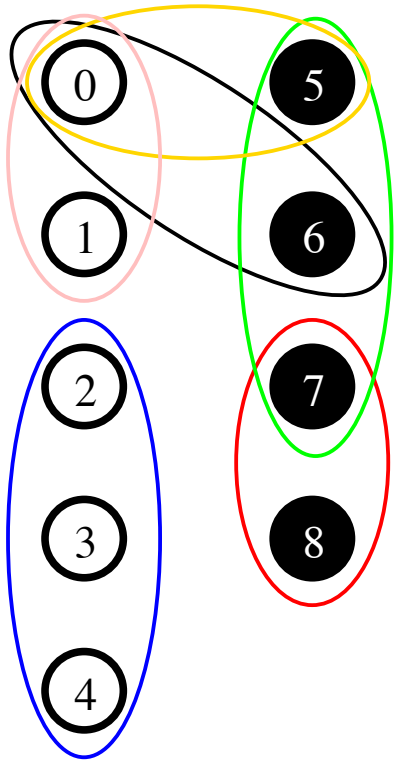
Each edge $e \in E$ of a graph connects but two vertices. Hypergraphs allow for **larger connectivities**.

Definition (Hypergraph)

Let $\mathcal{V}$ be a set of vertices and $\mathcal{N} \subseteq \mathcal{P}(\mathcal{V})$ a set of **hyperedges** (also called **nets**). Then $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a hypergraph.

Note that indeed $n \subseteq \mathcal{V}$, so $|n| > 2$ is possible.

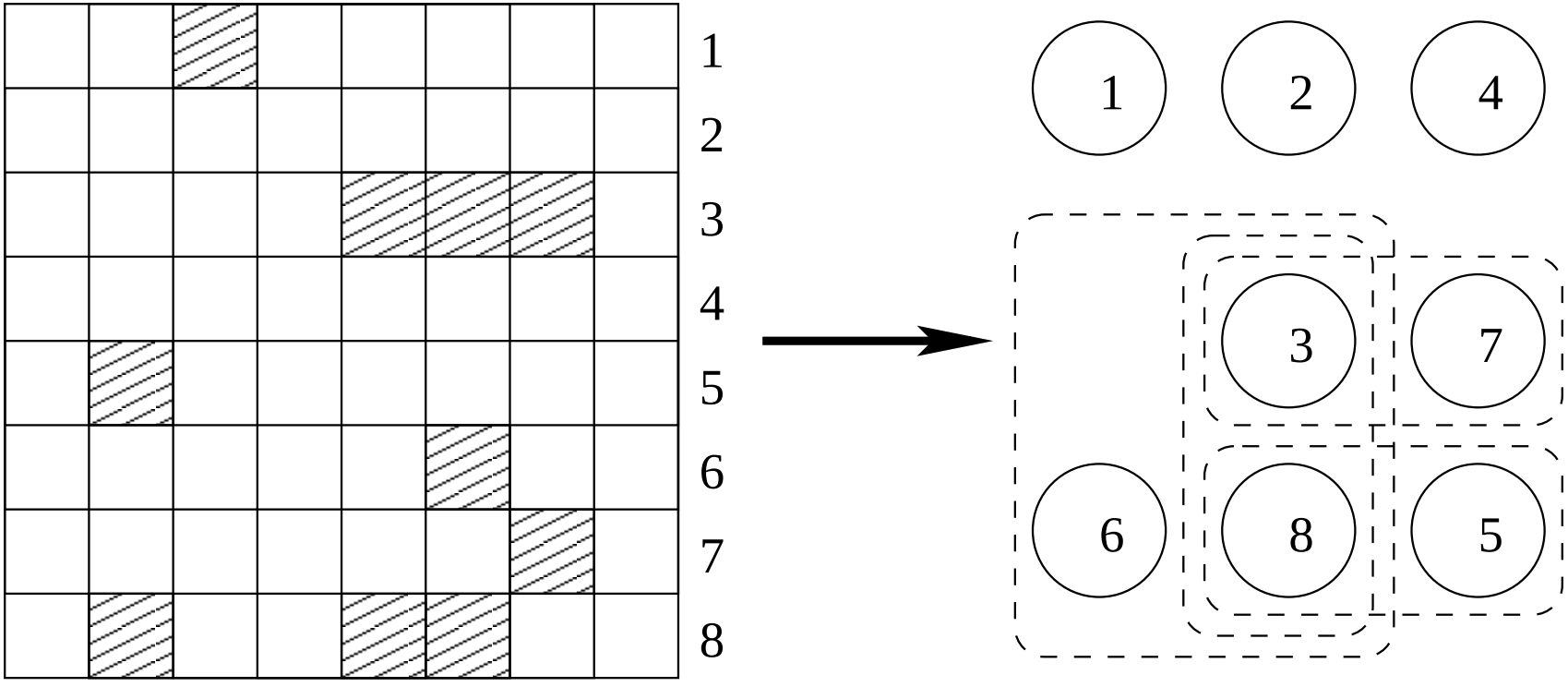
Example hypergraph



Hypergraph with 9 vertices and 6 hyperedges (nets), partitioned over 2 processors

From matrix to hypergraph

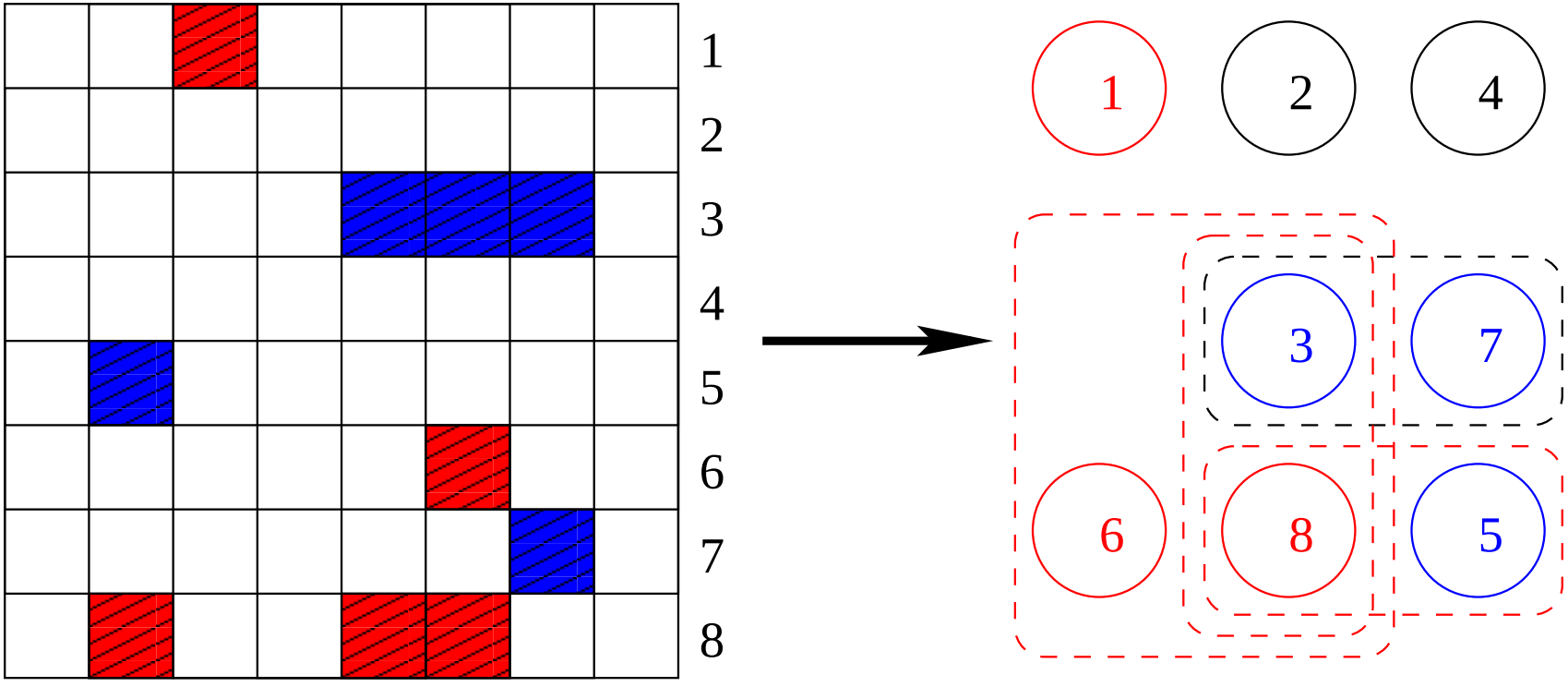
"Shared" columns: communication during fan-out



Column-net model; a cut net means a shared column

From matrix to hypergraph

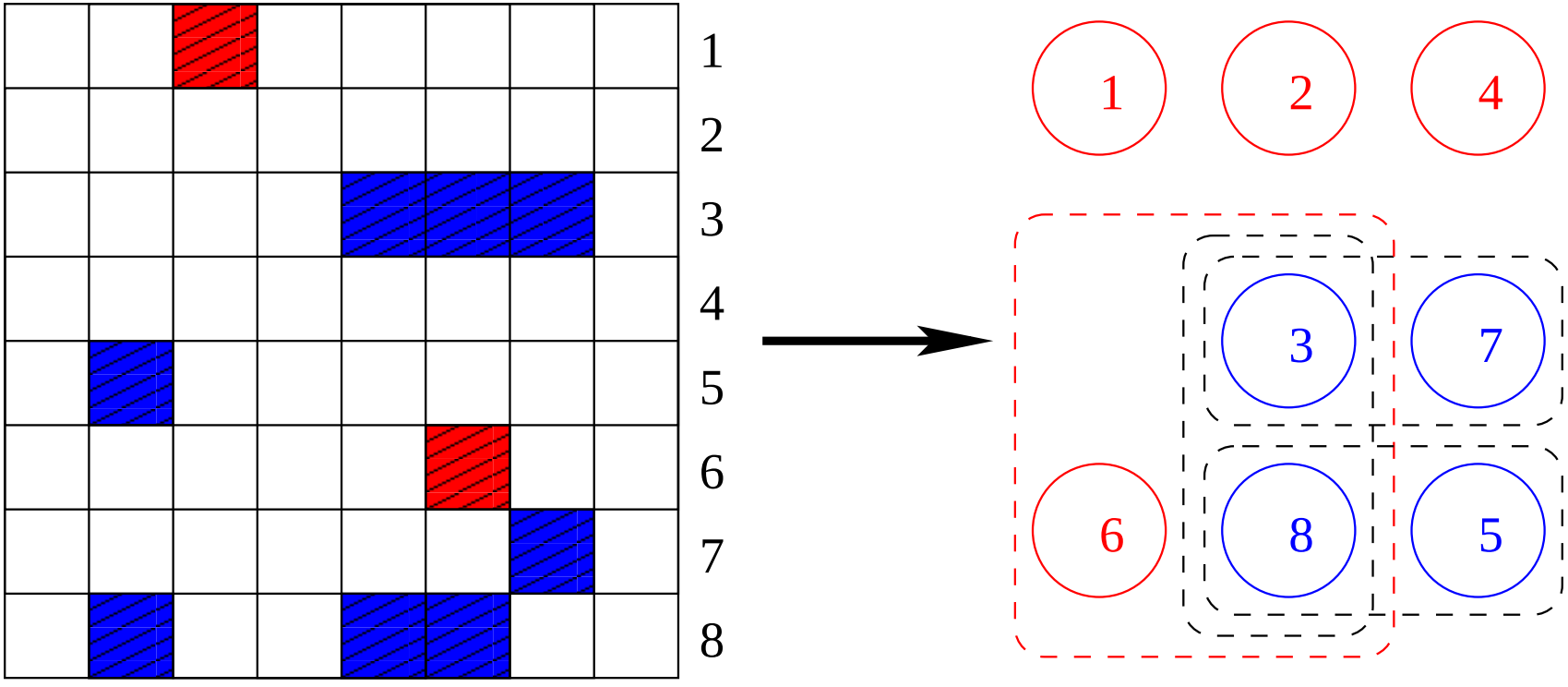
"Shared" columns: communication during fan-out



Column-net model; a cut net means a shared column

From matrix to hypergraph

"Shared" columns: communication during fan-out



Column-net model; a cut net means a shared column

From matrix to hypergraph

Definition (Column-net model of a sparse matrix)

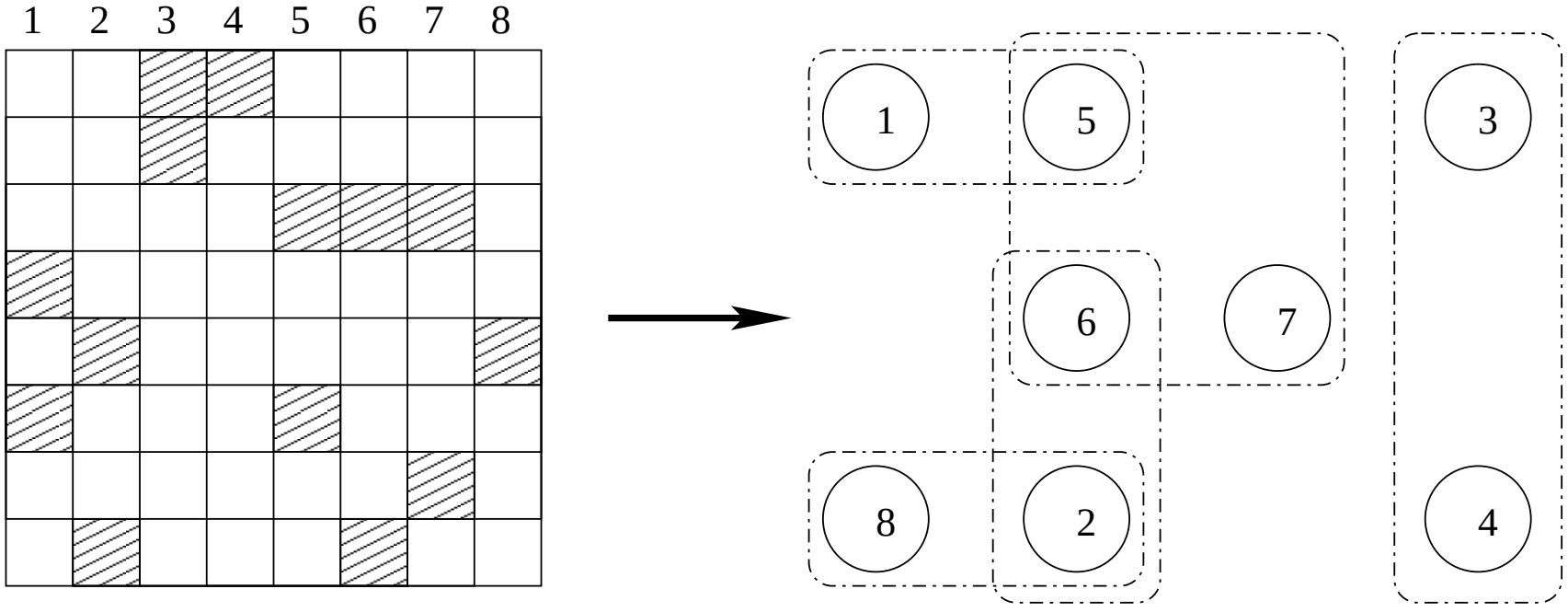
Let A be an $m \times n$ sparse matrix, $I = \{0, 1, \ldots, m-1\}$, and $J = \{0, 1, \ldots, n-1\}$. Define $\mathcal{V} = I$, and $\forall i \in I$ define a net $n_i \in \mathcal{N}$ with

$$n_i = \{j \in J \mid a_{ij} \neq 0\}.$$

Then $(\mathcal{V}, \mathcal{N})$ is the **column-net model** of A.

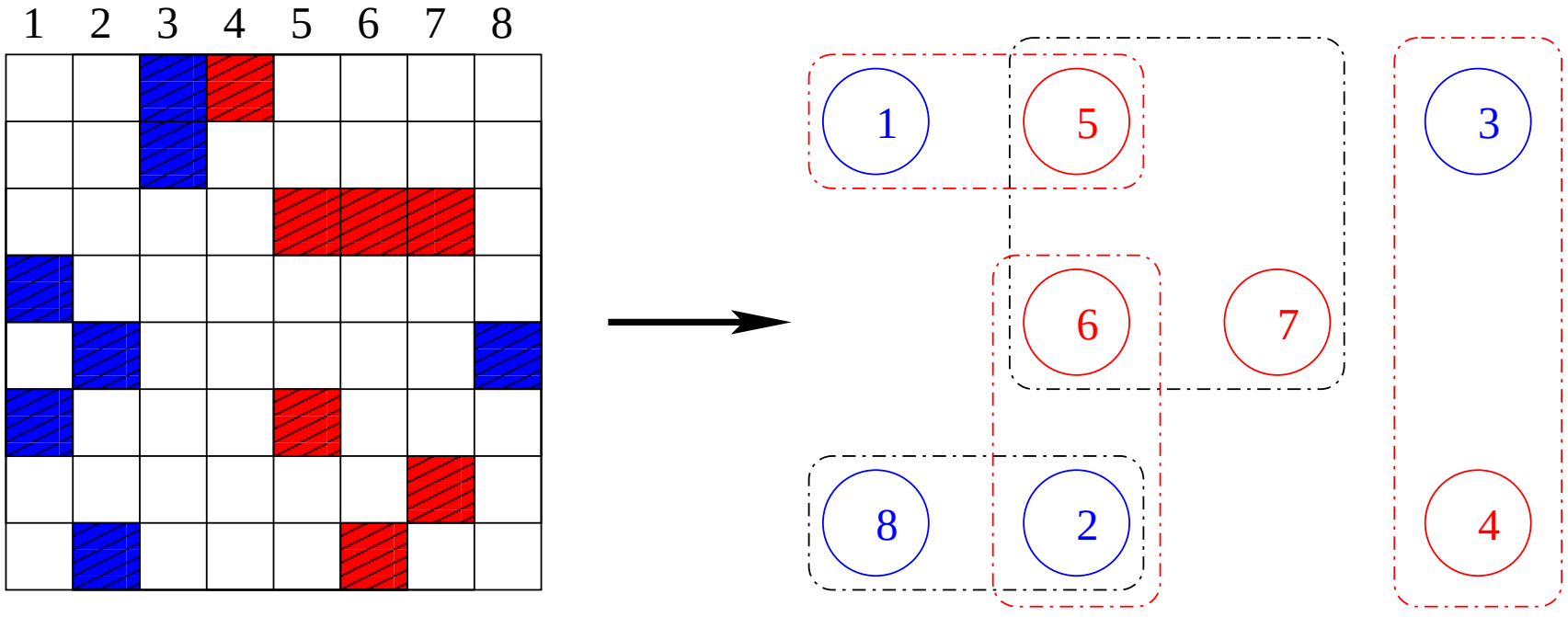
From matrix to hypergraph

"Shared" rows: communication during fan-in



From matrix to hypergraph

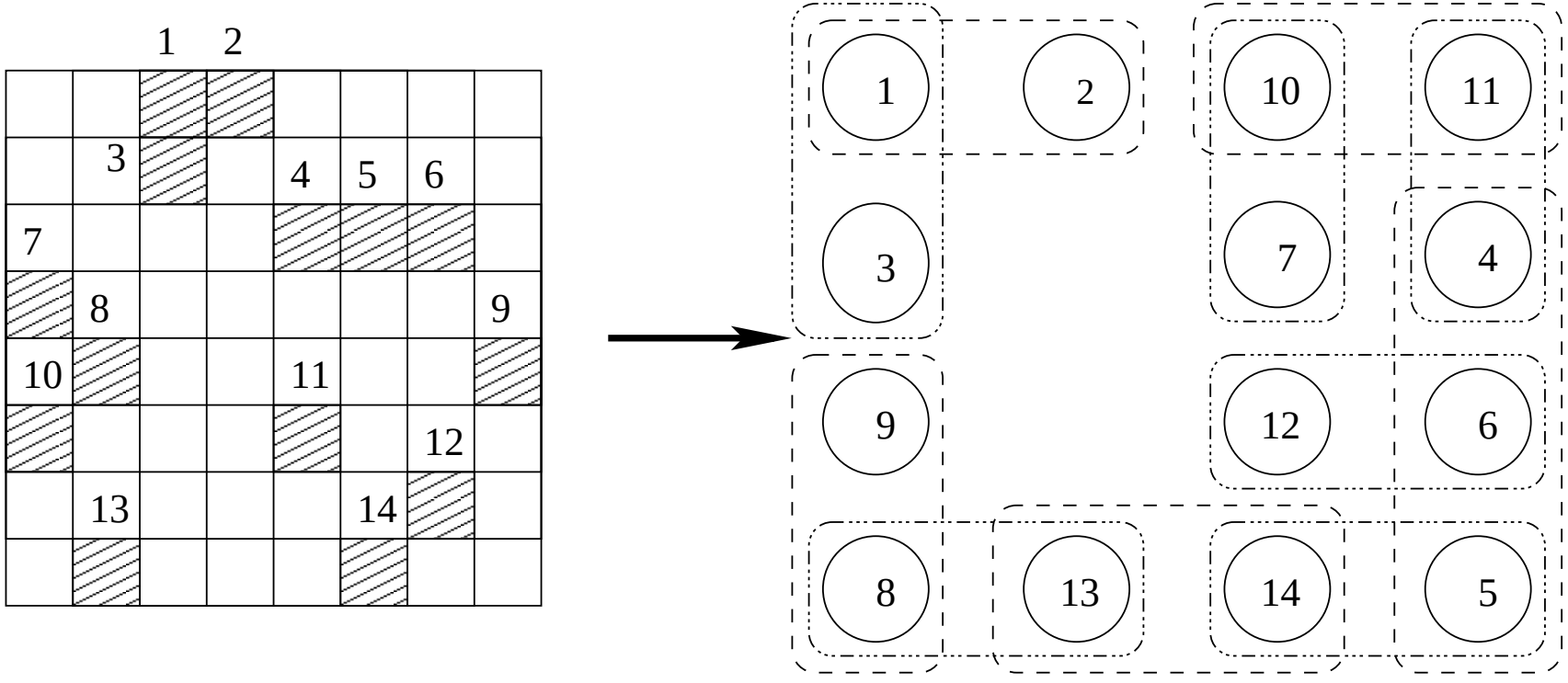
"Shared" rows: communication during fan-in



Row-net model; a cut net means a shared row. Definition is analogous to that of the column-net model, but with the roles of I and J switched.

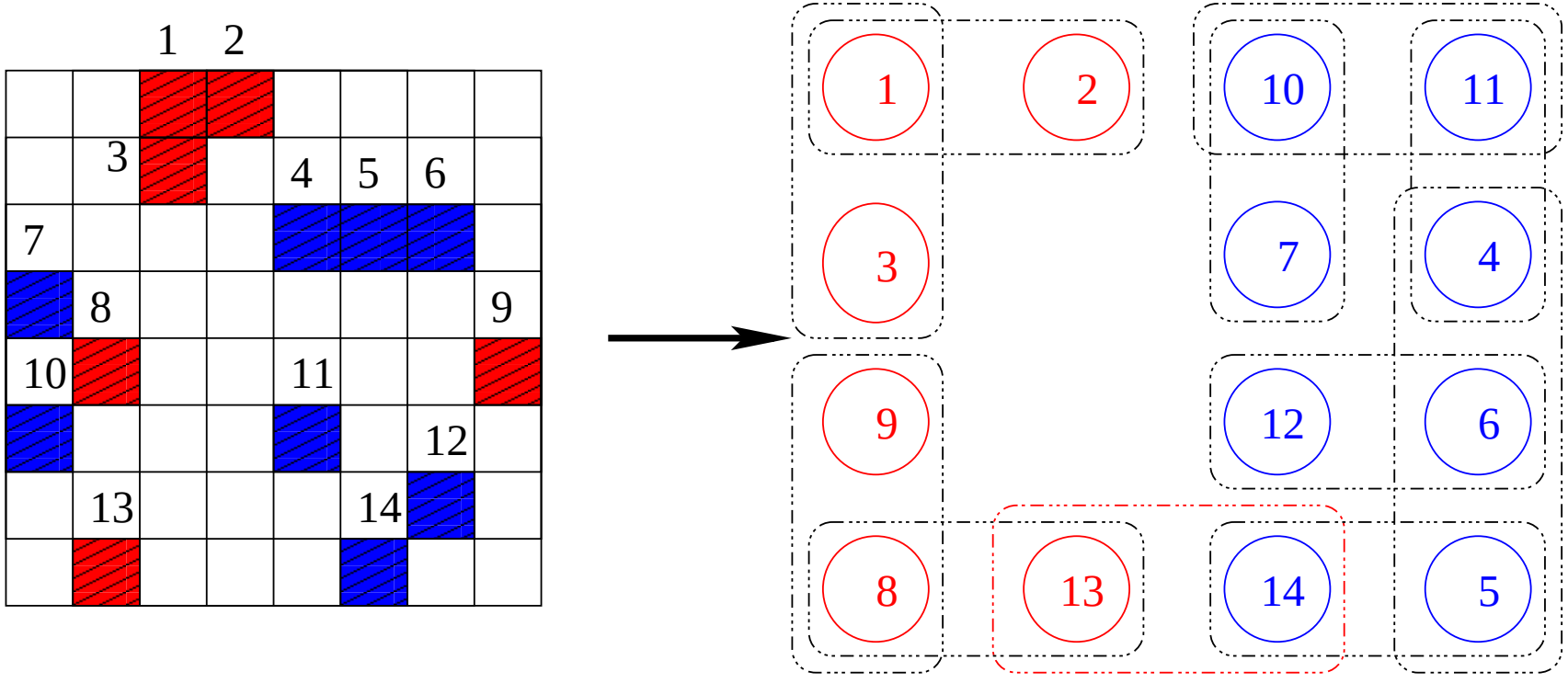
From matrix to hypergraph

"Catch" all communication:



From matrix to hypergraph

"Catch" all communication:



Fine-grain model; a cut net means **either** fan-out or fan-in.

Matrix communication to hypergraph costs

Do hypergraph models allow precise modeling of the communication volumes? (And if not, what other cost metrics make sense?)

Definition (Connectivity of a hyperedge)

Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ be a hypergraph. Let $\mathcal{P}_k = \{\mathcal{V}_0, \ldots, \mathcal{V}_{k-1}\}$ be a k-way partitioning of $\mathcal{H}$. Then, the connectivity λ_i of the hyperedge $n_i \in \mathcal{N}$ is given by

$$\lambda_i = |\{\mathcal{V}_i \in \mathcal{P}_k \mid n_i \cap \mathcal{V}_i \neq \emptyset\}|.$$

Matrix communication to hypergraph costs

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ and a partitioning $\mathcal{P}_k$ of $\mathcal{V}$.

- A net is **cut** precisely if its connectivity is larger than 1.

Definition (Cut-net metric)

The cost of a partitioning according to the cut-net metric is given by

$$\sum_{n_i \in \mathcal{N}} \begin{cases} 1, & \text{if } \lambda_i > 1 \\ 0, & \text{otherwise.} \end{cases}$$

Question: does this model the communication volume?

Matrix communication to hypergraph costs

Answer: **no**. But we can:

Definition ($(\lambda - 1)$-metric)

Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\mathcal{P}_k$, and λ_i be as before. Then, the $(\lambda - 1)$-metric is given by

$$\sum_{n_i \in \mathcal{N}} (\lambda_i - 1) .$$

This models the communication model exactly. It counts

- the amount of fan-out communication in the column-net model,
- the amount of fan-in communication in the row-net model, and
- the total amount of communication in the fine-grain model.

Hypergraph partitioning

General data partitioning

Definition (Hypergraph partitioning)

Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. A **partitioning** of $\mathcal{H}$ into p parts, is a partitioning $\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-1}$ of $\mathcal{V}$ into p parts such that

$$(1) \quad \mathcal{V} = \cup_{s=0}^{p-1} \mathcal{V}_s, \text{ and}$$

$$(2) \quad \forall i, j \in \{0, 1, \ldots, p-1\}, \mathcal{V}_i \cap \mathcal{V}_j = \emptyset.$$

Hypergraphs models of sparse matrices, combined with hypergraph partitioning, directly results in sparse matrix partitionings (in the sense of partitioning A into local matrices A_s).

- A partitioning of a column-net model of A corresponds to a partitioning of the rows of A (a 1D row-wise distribution).

General data partitioning

Definition (Hypergraph partitioning)

Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. A **partitioning** of $\mathcal{H}$ into p parts, is a partitioning $\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{p-1}$ of $\mathcal{V}$ into p parts such that

$$(1) \quad \mathcal{V} = \cup_{s=0}^{p-1} \mathcal{V}_s, \text{ and}$$

$$(2) \quad \forall i, j \in \{0, 1, \ldots, p-1\}, \mathcal{V}_i \cap \mathcal{V}_j = \emptyset.$$

Hypergraphs can model more than just sparse matrices; it can model any collection of data dependencies. But

- partitioning is not the only relevant operation (matching?),
- much effort goes into finding correct cost metric.

Sometimes a mathematical analysis can find optimal partitionings a priori. Here we present an automatic but **heuristic**, procedure.

Hypergraph partitioner

Following the example of sparse matrix partitioning:

- Model the sparse matrix using a hypergraph.
- **Partition the vertices** of that hypergraph (in two).

That is, first design a hypergraph bipartitioner.

Hypergraph partitioner

Following the example of sparse matrix partitioning:

- Model the sparse matrix using a hypergraph.
- **Partition the vertices** of that hypergraph (in two).

That is, first design a hypergraph bipartitioner.

State-of-the-art hypergraph partitioning is a multi-level scheme:

1. First **coarsen** the input hypergraph.
2. If the hypergraph remains too large, call this multi-level scheme recursively.
3. Otherwise, do random partitioning or optimal partitioning.
4. Undo coarsening.
5. **Refine** the resulting partitioning refinement (e.g., local search).

Partitioning: coarsening hypergraphs

Step 1: hypergraph coarsening

Definition (Coarsened hypergraph)

Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ be a hypergraph. Let $V_0, V_1, \ldots, V_{k-1}$ be a k-way partitioning of $\mathcal{V}$. Let $\mathcal{V}_c = \{V_0, \ldots, V_{k-1}\}$. For each $n_i \in \mathcal{N}$, there is a $n_i^c \in \mathcal{N}_c$ with

$$n_i^c = \{V_i \in \mathcal{V}_c \mid n_i \cap V_i \neq \emptyset\}.$$

Then $\mathcal{H}_c = (\mathcal{V}_c, \mathcal{N}_c)$ is a **coarsened hypergraph** of $\mathcal{H}$.

Coarsened hypergraphs should be structurally 'similar' to the original hypergraph.

Partitioning: coarsening hypergraphs

Step 1: hypergraph coarsening

Wanted: a measure for similarity.

- Assume a row-net model, where
- coarsening means combining matrix columns into 'supercolumns'.
- Hence, 'similar' columns should be combined:

Definition (structural inner product)

Let A, m, n, and I as before. Write $A_j^{\text{col}}, A_k^{\text{col}}$ for the jth and kth column of A, respectively. The **structural inner-product** $\langle j, k \rangle_{I_A}$ is

$$|\{i \in I \mid a_{ij} \neq 0 \text{ and } a_{ik} \neq 0\}|.$$

Partitioning: coarsening hypergraphs

Step 1: hypergraph coarsening

A related problem:

Definition (Graph matching)

Let $G = (V, E)$ be a graph. A **matching** $M \subseteq E$ has that for each $\{v, w\} \in M$, there is no other edge $e \in M$ for which $v \in e$ or $w \in e$.

- Let $\mathcal{M}$ be the set of all possible matchings. Then $M \in \mathcal{M}$ is **maximal** if for all $e \in E \backslash M$, $M \cup \{e\} \notin \mathcal{M}$.
- $M \in \mathcal{M}$ is **maximum** if there is no other matching $M_0 \in \mathcal{M}$ s.t. $|M_0| > |M|$.

Partitioning: coarsening hypergraphs

Step 1: hypergraph coarsening

A related problem:

Definition (Weighted graph matching)

Suppose a graph $G = (V, E)$ has an associated weight function $w : E \to \mathbb{R}$. Then (V, E, w) is a **weighted graph**.

We define the **weight** $w(M)$ of a matching M as $\sum_{\{j,k\} \in M} w_{jk}$.

A matching M is a **weighted maximum matching** if there is no other $M_0 \in \mathcal{M}$ such that $w(M_0) > w(M)$.

Partitioning: coarsening hypergraphs

Step 1: hypergraph coarsening

- Let $V = \{A_0^{\text{col}}, A_1^{\text{col}}, \ldots, A_{n-1}^{\text{col}}\}$.
- Let the edge $e_i(A_i^{\text{col}}, A_j^{\text{col}}) \in E$ have weight $w(i)$ equal to the similarity measure of the two columns.
- $G = (V, E, w)$ forms a fully connected edge-weighted graph.
- Let M be a weighted maximum matching of G.

Then a valid coarsening strategy is to merge all column pairs in M.

Partitioning: coarsening hypergraphs

Step 1: hypergraph coarsening

Merging similar columns in pairs to reduce the problem size (repeat this until the problem is small):

$$\begin{bmatrix} \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot \\ 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & 1 \end{bmatrix} \xrightarrow{merge} \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ 1 & \cdot & 1 & 1 \\ 1 & 1 & \cdot & 1 \\ 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & \cdot & 1 \end{bmatrix}$$

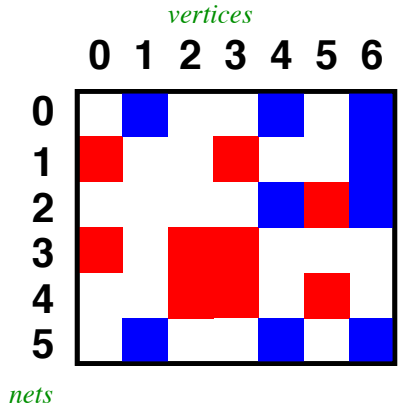
Partitioning: HKLFM

Step 5: refinement

After uncoarsening, reduce communication through local search methods.

- E.g., Kernighan-Lin, with improved implementation by Fiduccia and Mattheyses (KLFM), is often used.
- The cost function to minimise during local search is the $(\lambda - 1)$-metric.
- Moves that violate the load-balance criterion are marked invalid.

Partitioning: HKLFM



Example of refinement using the row-net model:

- HKLFM tries to improve initial uncoarsened partitioning by moving vertices (columns) to the other part.
- The vertex with the largest gain (communication reduction) is moved. If the best possible move increases the communication, it is still accepted.
- Several passes are carried out. Vertices are never moved twice in a pass. Best solution encountered is kept.

Partitioning

References:

Catalyürek & Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel Distributed Systems 10 (1999), pp. 673-693

Kernighan & Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Systems Technical Journal 49 (1970): pp. 291-307

Fiduccia & Mattheyses, *A linear-time heuristic for improving network partitions*, Proceedings of the 19th IEEE Design Automation Conference (1982), pp. 175-181.

Example software: Mondriaan (Bisseling et al., UU), Zoltan (Boman et al., Sandia), PaToH (Çatalyürek & Aykanat, OSU), and Scotch (Pellegrini, Bordelais).

Communication volume and time: 1D vs. 2D

(Vastenhouw and Bisseling, *SIAM Review* **47** (2005) pp.67–95.)

p	Volume (in data words)			Time (in ms)		
	1D row	1D col	2D	1D row	1D col	2D
1	0	0	0	67.55	67.61	74.15
2	15764	24463	15764	36.65	32.26	32.16
4	42652	54262	30444	14.06	12.22	12.14
8	90919	96038	49120	6.49	6.35	6.62
16	177347	155604	75884	5.22	4.22	4.20
32	297658	227368	106563	4.32	4.08	3.23

Term-by-document matrix `tbdlinux`:
112,757 rows; 20,167 columns; 2,157,675 nonzeros.
Timings obtained on an SGI Origin 3800.

Summary

- We have derived a recursive partitioning algorithm for a sparse matrix. It is **greedy** (minimises splits separately without looking ahead).
- The result is a p-way matrix partitioning $A_0, \ldots, A_{p-1}$.
- We used hypergraphs $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, which generalise the notion of a graph.
- Multilevel methods for hypergraph partitioning find good splits of a sparse matrix in reasonable time.
- The sparse matrix partitioner introduced here optimises communication **volume**. Other possible metrics:
  - h-relations, or
  - number of messages.
- In the book, the vector distribution is used to **balance communication**, i.e., uses the minimised communication volume to get minimised h-relations.

Vector distribution

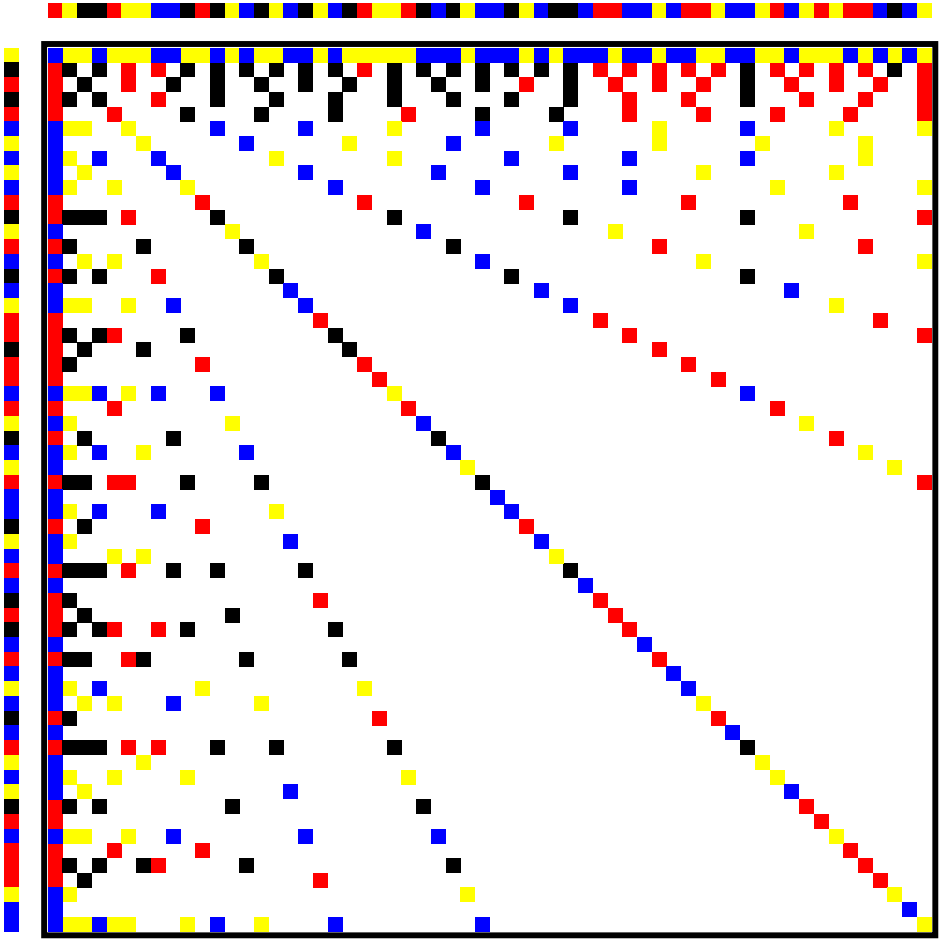
Balance the communication!

- Aim: reduce the BSP cost hg, where

$$h = \max_{0 \le s < p} h(s), \qquad h(s) = \max(h_{\mathrm{s}}(s), h_{\mathrm{r}}(s)).$$

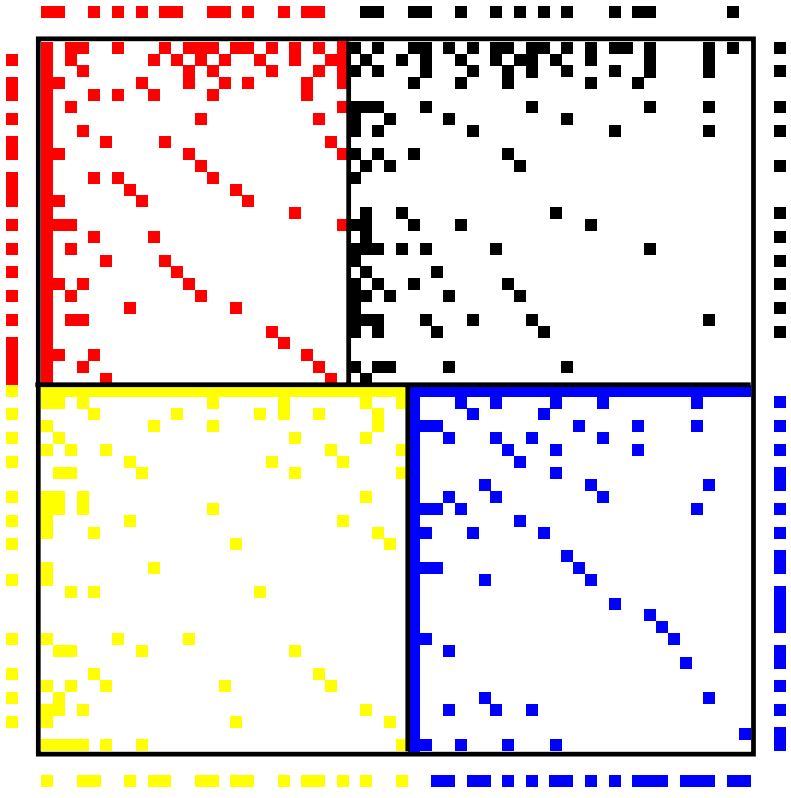
- Thus, given a matrix distribution ϕ, we have to determine a vector distribution $\phi_{\mathbf{v}}$ that minimises h for the fanout and satisfies $j \in J_{\phi_{\mathbf{v}}(j)}$, for $0 \le j < n$.
- Constraint $j \in J_{\phi_{\mathbf{v}}(j)}$ means: processor $P(s) = P(\phi_{\mathbf{v}}(j))$ that owns v_j must own a nonzero in matrix column j, i.e., $j \in J_s$.
- We also have to find a vector distribution $\phi_{\mathbf{u}}$ that minimises the value h for the fanin and that satisfies the constraint $i \in I_{\phi_{\mathbf{u}}(i)}$, for $0 \le i < n$.

Vector partitioning for `prime60`



Global view. Both constraints are satisfied.

Vector partitioning for `prime60`



Local view. The local components of the vector $\mathbf{u}$ are placed to the left of the local submatrix for $P(0)$ and $P(2)$.

The two vector distribution problems are similar

- Nonzero pattern of row i of A equals the nonzero pattern of column i of A^T:
  u_{is} is sent from $P(s)$ to $P(t)$ in the multiplication by A
  $\Leftrightarrow$ v_i is sent from $P(t)$ to $P(s)$ in the multiplication by A^T.
- We can find a good distribution $\phi_{\mathbf{u}}$ given $\phi = \phi_A$ by finding a good distribution $\phi_{\mathbf{v}}$ given $\phi = \phi_{A^T}$.
- Hence, we only solve one problem, namely for $\mathbf{v}$. We can apply this method also for $\mathbf{u}$, with A^T instead of A.

General case: arbitrary q_j values

- Columns with $q_j = 0$ or $q_j = 1$ do not cause communication and are omitted from the problem. Hence, we assume $q_j \geq 2$, for all j.
- For processor $P(s)$:

$$h_{\mathrm{s}}(s) = \sum_{0 \leq j < n,\ \phi_{\mathbf{v}}(j) = s} (q_j - 1),$$

and

$$h_{\mathrm{r}}(s) = |\{j : j \in J_s \wedge \phi_{\mathbf{v}}(j) \neq s\}|.$$

- Aim: for given matrix distribution and hence given communication volume V, minimise

$$h = \max_{0 \leq s < p} \max\left(h_{\mathrm{s}}(s), h_{\mathrm{r}}(s)\right).$$

Egoistic local bound

- An egoistic processor tries to minimise its own $h(s) = \max(h_{\mathrm{r}}(s), h_{\mathrm{s}}(s))$ without consideration for others.
- To minimise $h_{\mathrm{r}}(s)$, it just has to maximise the number of components v_j with $j \in J_s$ that it owns.
- To minimise $h_{\mathrm{s}}(s)$, it has to minimise the total weight of these components, where the weight of v_j is $q_j - 1$.
- A locally optimal strategy is to start with $h_{\mathrm{s}}(s) = 0$ and $h_{\mathrm{r}}(s) = |J_s|$ and grab the components in order of increasing weight, each time adjusting $h_{\mathrm{s}}(s)$ and $h_{\mathrm{r}}(s)$, as long as $h_{\mathrm{s}}(s) \leq h_{\mathrm{r}}(s)$.

Optimal values

- Denote the resulting optimal value of $h_{\mathrm{r}}(s)$ by $\hat{h}_{\mathrm{r}}(s)$, that of $h_{\mathrm{s}}(s)$ by $\hat{h}_{\mathrm{s}}(s)$, and that of $h(s)$ by $\hat{h}(s)$. We have

$$\hat{h}_{\mathrm{s}}(s) \leq \hat{h}_{\mathrm{r}}(s) = \hat{h}(s), \text{ for } 0 \leq s < p.$$

- The value $\hat{h}(s)$ is a local lower bound on the actual value that can be achieved: $\hat{h}(s) \leq h(s)$, for all s.

Algorithm based on local bound

(R. H. Bisseling, W. Meesen, *Electronic Transactions on Numerical Analysis* **21** (2005) pp. 47–65.)

- Define the generalised lower bound $\hat{h}(J, ns_0, nr_0)$ for a given index set $J \subset J_s$ and a given initial number of sends ns_0 and receives nr_0.
- Initial communications are due to columns outside J.
- Bound is computed by the same method, but starting with $h_{\mathrm{s}}(s) = ns_0$ and $h_{\mathrm{r}}(s) = nr_0 + |J|$.
- Note that $\hat{h}(s) = \hat{h}(J_s, 0, 0)$.
- Our algorithm gives preference to the processor that faces the toughest future, i.e., the processor with the highest current value $\hat{h}(s)$.

Initialisation of algorithm

for $s := 0$ **to** $p - 1$ **do**
$L_s := J_s$;
$h_{\mathrm{s}}(s) := 0$;
$h_{\mathrm{r}}(s) := 0$;

- L_s is the index set of components that may still be assigned to $P(s)$.
- The number of sends caused by the assignments done so far is registered as $h_{\mathrm{s}}(s)$; the number of receives as $h_{\mathrm{r}}(s)$.
- The current state of $P(s)$ is represented by the triple $(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s))$.

Termination of algorithm

for $s := 0$ **to** $p - 1$ **do**
  if $h_{\mathrm{s}}(s) < \hat{h}_{\mathrm{s}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s))$ **then**
    $\mathrm{active}(s) := \mathit{true}$;
  else $\mathrm{active}(s) := \mathit{false}$;

- Note that $ns_0 \leq \hat{h}_{\mathrm{s}}(J, ns_0, nr_0)$, so that trivially $h_{\mathrm{s}}(s) \leq \hat{h}_{\mathrm{s}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s))$.
- A processor will not accept more components once it has achieved its optimum, when $h_{\mathrm{s}}(s) = \hat{h}_{\mathrm{s}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s))$.

Main loop of algorithm

while $(\exists s : 0 \leq s < p \wedge \text{active}(s))$ **do**

$s_{\text{max}} := \text{argmax}(\hat{h}_{\text{r}}(L_s, h_{\text{s}}(s), h_{\text{r}}(s)) : 0 \leq s < p \wedge \text{active}(s))$;

$j := \text{min}(L_{s_{\text{max}}})$; $\{j$ has minimal $q_j\}$

$\phi_{\mathbf{v}}(j) := s_{\text{max}}$;

$h_{\text{s}}(s_{\text{max}}) := h_{\text{s}}(s_{\text{max}}) + q_j - 1$;

Main loop of algorithm

while $(\exists s : 0 \leq s < p \wedge \mathrm{active}(s))$ **do**

$s_{\max} := \mathrm{argmax}(\hat{h}_{\mathrm{r}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s)) : 0 \leq s < p \wedge \mathrm{active}(s))$;

$j := \min(L_{s_{\max}})$; $\{j$ has minimal $q_j\}$

$\phi_{\mathbf{v}}(j) := s_{\max}$;

$h_{\mathrm{s}}(s_{\max}) := h_{\mathrm{s}}(s_{\max}) + q_j - 1$;

for all $s : 0 \leq s < p \wedge s \neq s_{\max} \wedge j \in J_s$ **do**

$h_{\mathrm{r}}(s) := h_{\mathrm{r}}(s) + 1$;

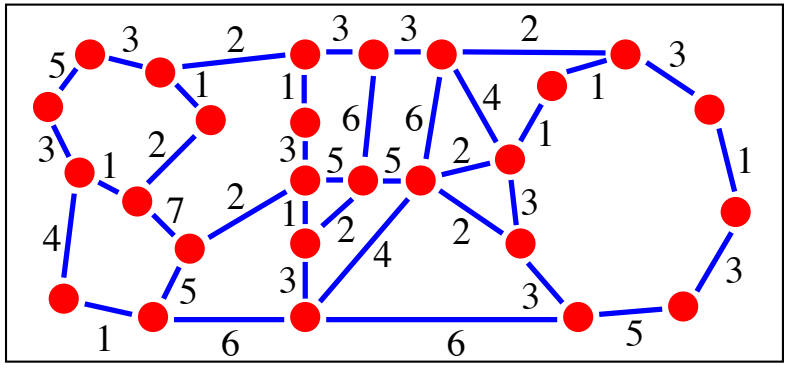
for all $s : 0 \leq s < p \wedge j \in J_s$ **do**

$L_s := L_s \backslash \{j\}$;

if $h_{\mathrm{s}}(s) = \hat{h}_{\mathrm{s}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s))$ **then**

$\mathrm{active}(s) := \mathit{false}$;

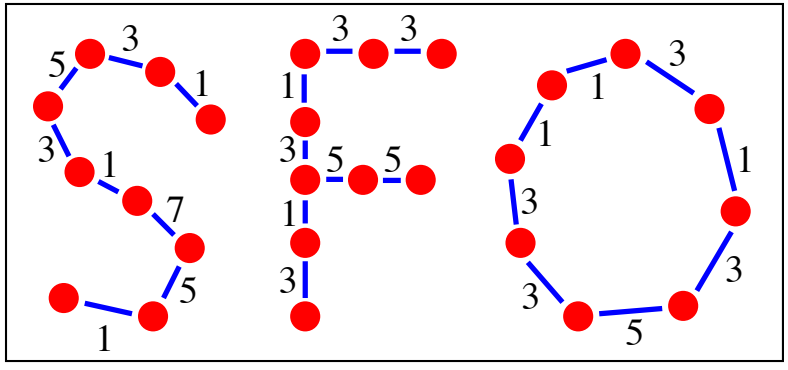
Special case: $q_j \leq 2$



- Vertex s = processor s, $0 \leq s < p$
- Edge (s, t) = processor pair sharing matrix columns
- Edge weight $w(s, t)$ = number of matrix columns shared

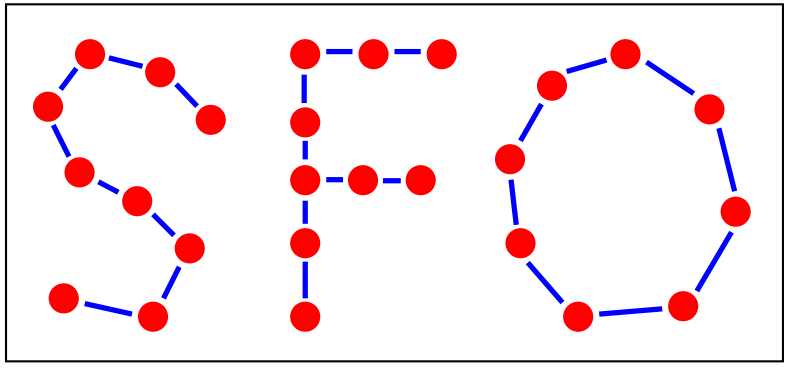
Problem: assign each matrix column/vector component to a processor, balancing the number of data words sent and received

Transform into unweighted undirected graph

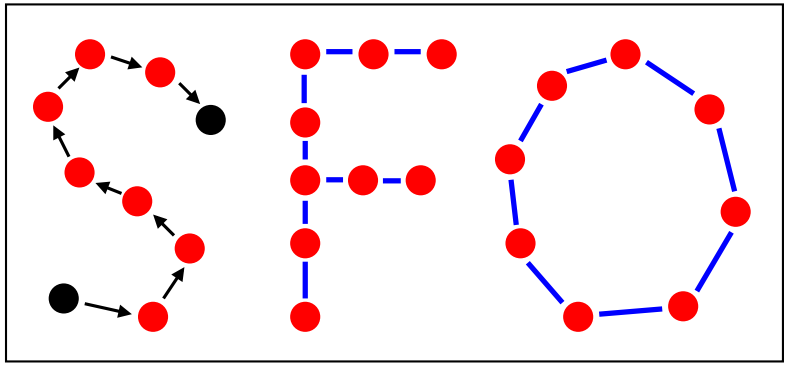


- Assign two shared columns: one to processor s, one to t. $w(s,t) := w(s,t) - 2$.
- Repeat until all edge weights $= 0$ or 1.

Unweighted undirected graph

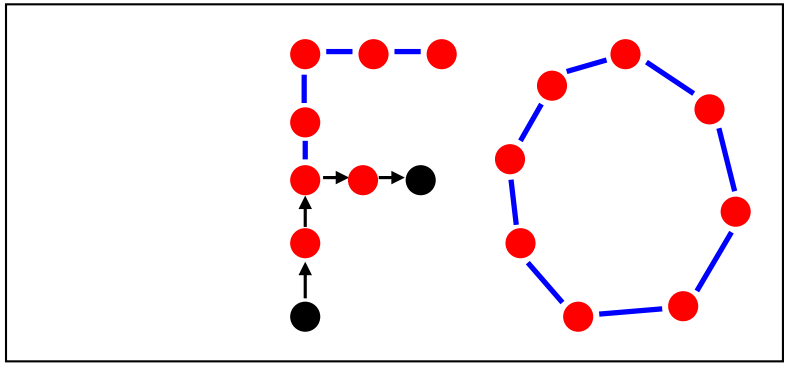


Transform into directed graph

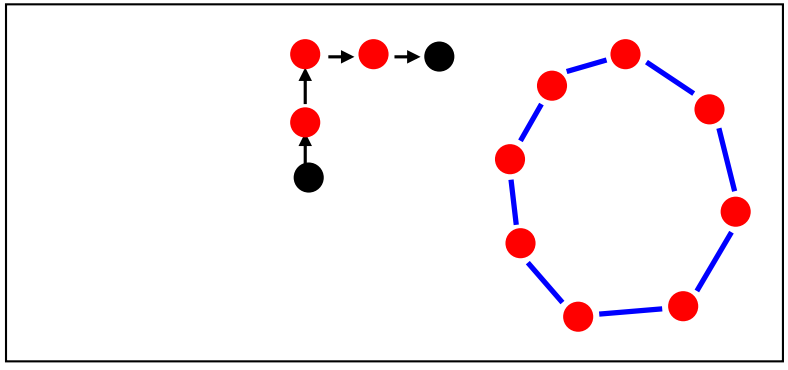


- Walk path starting at odd-degree vertex
- Remove walked edges from undirected graph
- Edge $s \longrightarrow t$: processor s sends, t receives
- Even-degree vertices remain even-degree
- Repeat until all degrees in undirected graph are even.

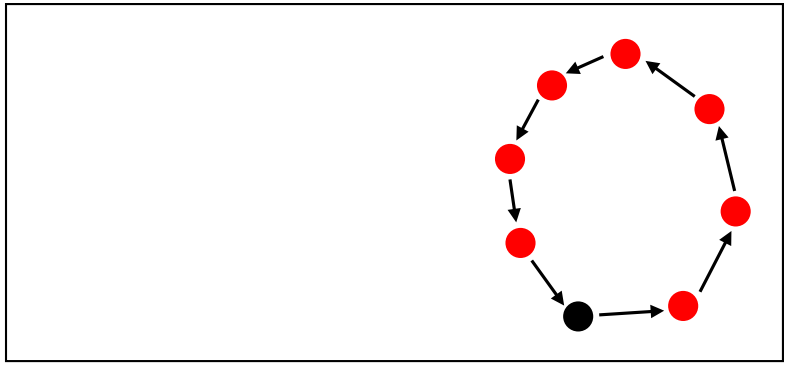
Transform into directed graph



Transform into directed graph



Transform into directed graph



- Walk path starting at even-degree vertex
- Repeat until undirected graph empty
- Solution is provably optimal (see Bisseling & Meesen 2005)

Summary

- BSP cost is a natural metric that encourages communication balancing.
- For the general vector distribution problem, we have developed a heuristic method, which works well in practice.
- The heuristic method is based on assigning vector components to the processor with the toughest future, as predicted by an egoistic local bound.
- For the special case with at most 2 processors per matrix column, we have obtained an optimal method based on walking paths in an associated graph, starting first at odd-degree vertices.