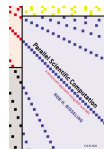
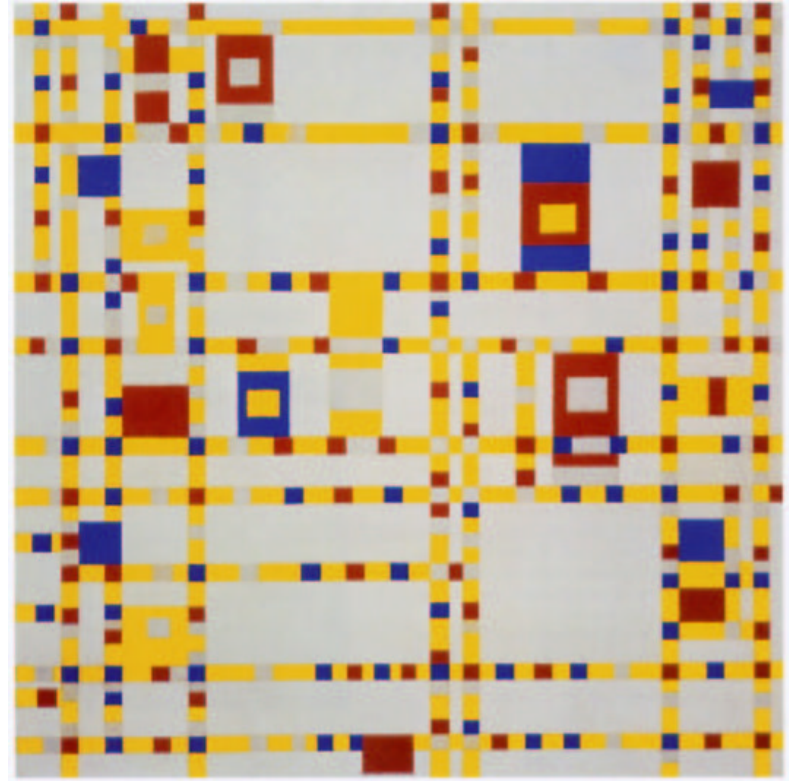
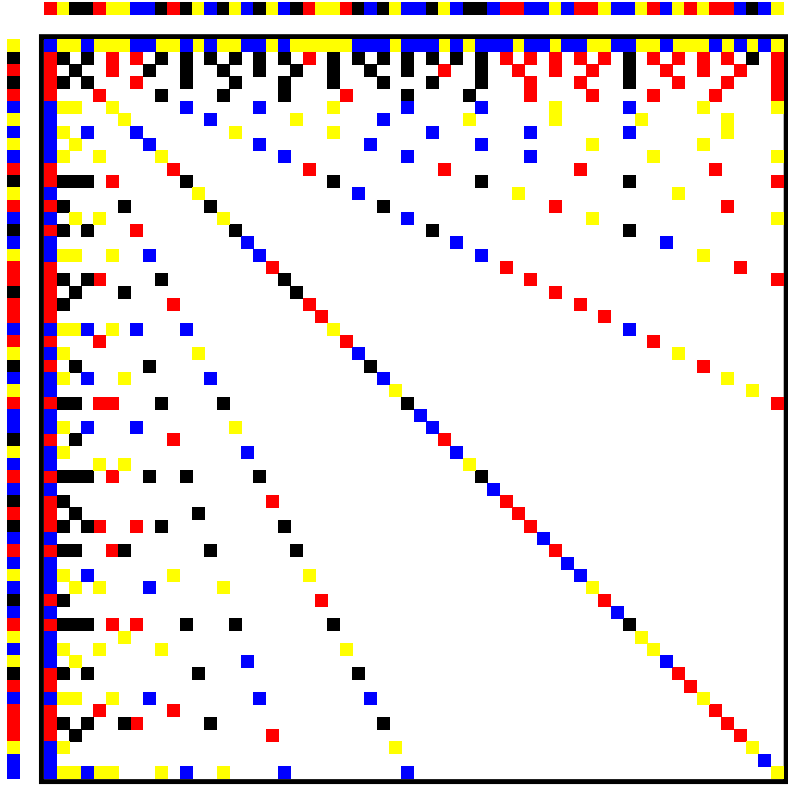


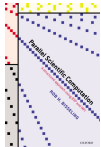
# ***Vector Distribution*** **(PSC §4.6)**



# *Vector partitioning*



Broadway Boogie Woogie  
Piet Mondriaan 1943

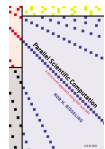


# Balance the communication!

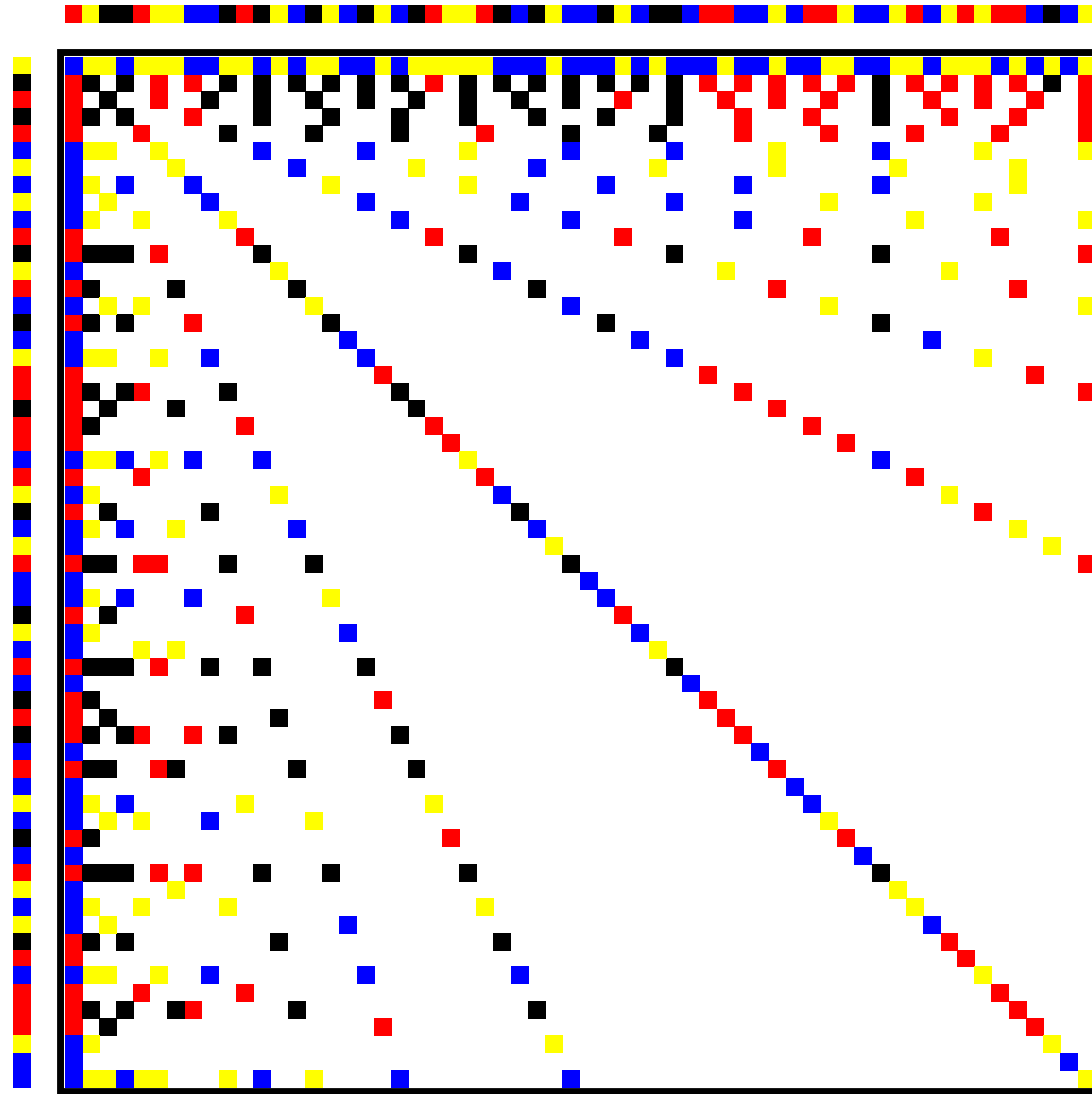
- Aim: reduce the BSP cost  $hg$ , where

$$h = \max_{0 \leq s < p} h(s), \quad h(s) = \max(h_s(s), h_r(s)).$$

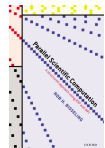
- Thus, given a matrix distribution  $\phi$ , we have to determine a vector distribution  $\phi_v$  that **minimises  $h$**  for the fanout and **satisfies  $j \in J_{\phi_v(j)}$** , for  $0 \leq j < n$ .
- Constraint  $j \in J_{\phi_v(j)}$  means: processor  $P(s) = P(\phi_v(j))$  that owns  $v_j$ 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



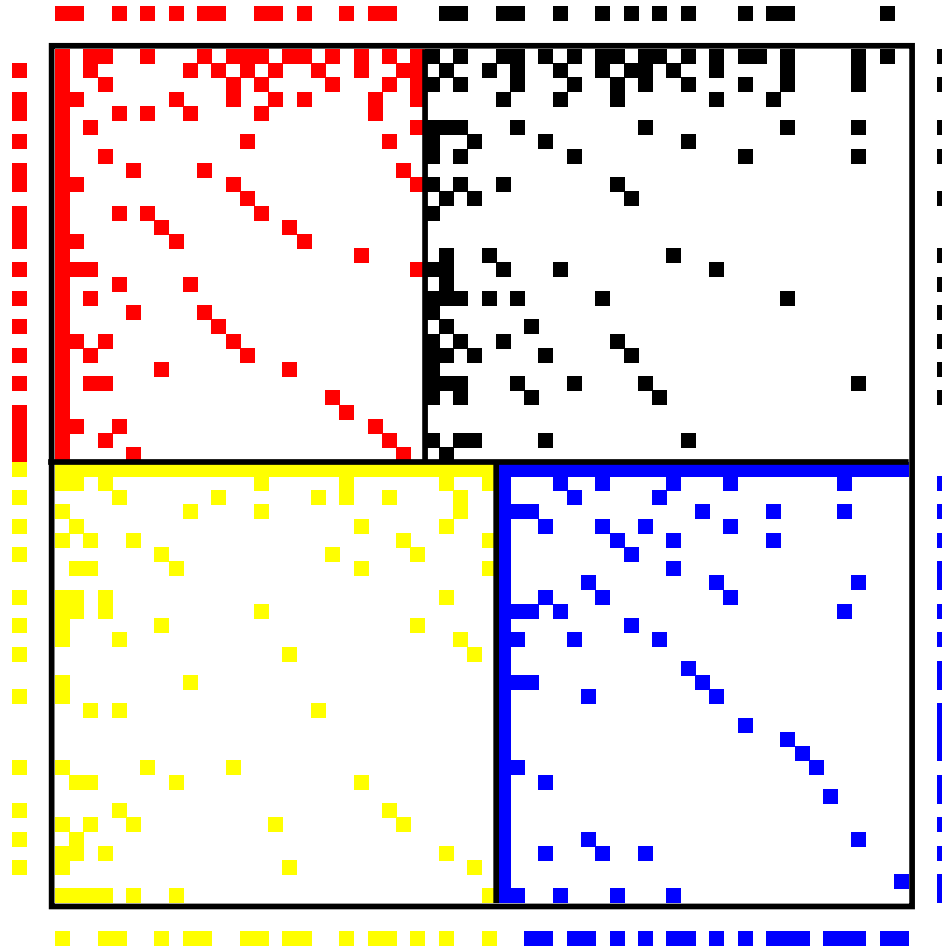
# Vector partitioning for *prime60*



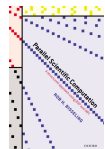
Global view. Both constraints are satisfied.



# Vector partitioning for *prime60*

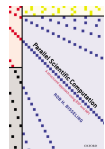


**Local view.** The local components of the vector  $u$  are placed to the left of the local submatrix for  $P(0)$  and  $P(2)$ .



# 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  $\phi_u$  given  $\phi = \phi_A$  by finding a good distribution  $\phi_v$  given  $\phi = \phi_{A^T}$ .
- Hence, we only solve one problem, namely for  $v$ . We can apply this method also for  $u$ , 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  $P(s)$ :

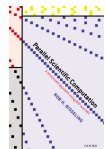
$$h_s(s) = \sum_{0 \leq j < n, \phi_v(j)=s} (q_j - 1),$$

and

$$h_r(s) = |\{j : j \in J_s \wedge \phi_v(j) \neq s\}|.$$

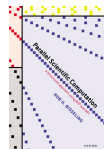
- Aim: for given matrix distribution and hence given communication volume  $V$ , minimise

$$h = \max_{0 \leq s < p} \max (h_s(s), h_r(s)).$$



# Egoistic local bound

- An egoistic processor tries to minimise its own  $h(s) = \max(h_r(s), h_s(s))$  without consideration for others.
- To minimise  $h_r(s)$ , it just has to **maximise the number of components**  $v_j$  with  $j \in J_s$  that it owns.
- To minimise  $h_s(s)$ , 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  $h_s(s) = 0$  and  $h_r(s) = |J_s|$  and grab the components **in order of increasing weight**, each time adjusting  $h_s(s)$  and  $h_r(s)$ , as long as  $h_s(s) \leq h_r(s)$ .



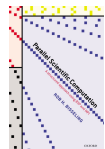


# Optimal values

- Denote the resulting optimal value of  $h_r(s)$  by  $\hat{h}_r(s)$ , that of  $h_s(s)$  by  $\hat{h}_s(s)$ , and that of  $h(s)$  by  $\hat{h}(s)$ . We have

$$\hat{h}_s(s) \leq \hat{h}_r(s) = \hat{h}(s), \text{ for } 0 \leq s < p.$$

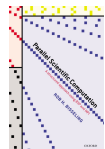
- The value  $\hat{h}(s)$  is a local lower bound on the actual value that can be achieved:  $\hat{h}(s) \leq h(s)$ , for all  $s$ .



## Example vector distribution problem

$s = 0$	1	.	1	.	1	1	1	1
1	1	1	.	1	1	1	1	.
2	.	1	.	.	.	1	1	1
3	.	.	1	1	1	.	.	1
$q_j =$	2	2	2	2	3	3	3	3
$j =$	0	1	2	3	4	5	6	7

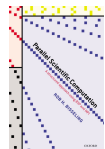
- A 1 in the table denotes that  $P(s)$  owns a nonzero in column  $j$  and hence needs  $v_j$ .
- Columns are ordered by increasing  $q_j$ .
- Processor  $P(0)$  wants  $v_0$  and  $v_2$ , but nothing more, so that  $\hat{h}_s(0) = 2$ ,  $\hat{h}_r(0) = 4$ , and  $\hat{h}(0) = 4$ .
- The fanout will cost at least  $4g$ .



# 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  $h_s(s) = ns_0$  and  $h_r(s) = nr_0 + |J|$ .
- 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

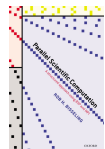
**for**  $s := 0$  **to**  $p - 1$  **do**

$L_s := J_s;$

$h_s(s) := 0;$

$h_r(s) := 0;$

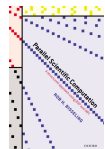
- $L_s$  is the index set of components that may still be assigned to  $P(s)$ .
- The number of sends caused by the assignments done so far is registered as  $h_s(s)$ ; the number of receives as  $h_r(s)$ .
- The current state of  $P(s)$  is represented by the triple  $(L_s, h_s(s), h_r(s))$ .



# Termination of algorithm

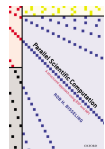
```
for  $s := 0$  to  $p - 1$  do  
    if  $h_s(s) < \hat{h}_s(L_s, h_s(s), h_r(s))$  then  
         $\text{active}(s) := \text{true};$   
    else  $\text{active}(s) := \text{false};$ 
```

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



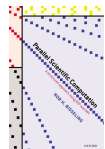
## *Main loop of algorithm*

**while**  $(\exists s : 0 \leq s < p \wedge \text{active}(s))$  **do**  
     $s_{\max} := \operatorname{argmax}(\hat{h}_r(L_s, h_s(s), h_r(s)) : 0 \leq s < p \wedge \text{active}(s));$   
     $j := \min(L_{s_{\max}}); \{j \text{ has minimal } q_j \}$   
     $\phi_{\mathbf{v}}(j) := s_{\max};$   
     $h_s(s_{\max}) := h_s(s_{\max}) + q_j - 1;$

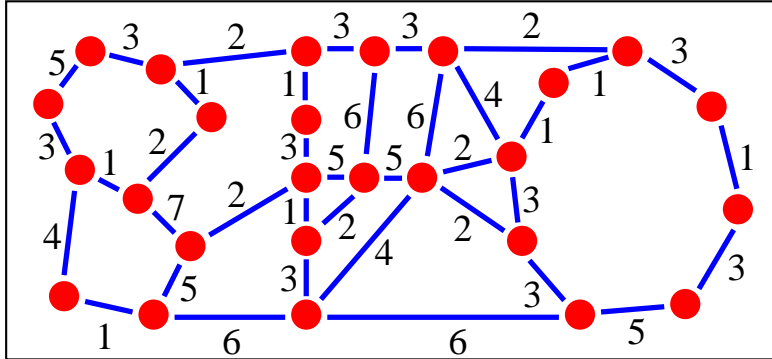


# Main loop of algorithm

**while**  $(\exists s : 0 \leq s < p \wedge \text{active}(s))$  **do**  
     $s_{\max} := \operatorname{argmax}(\hat{h}_r(L_s, h_s(s), h_r(s)) : 0 \leq s < p \wedge \text{active}(s));$   
     $j := \min(L_{s_{\max}}); \{j \text{ has minimal } q_j\}$   
     $\phi_{\mathbf{v}}(j) := s_{\max};$   
     $h_s(s_{\max}) := h_s(s_{\max}) + q_j - 1;$   
  
    **for all**  $s : 0 \leq s < p \wedge s \neq s_{\max} \wedge j \in J_s$  **do**  
         $h_r(s) := h_r(s) + 1;$   
  
    **for all**  $s : 0 \leq s < p \wedge j \in J_s$  **do**  
         $L_s := L_s \setminus \{j\};$   
        **if**  $h_s(s) = \hat{h}_s(L_s, h_s(s), h_r(s))$  **then**  
             $\text{active}(s) := \text{false};$

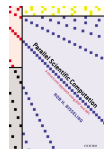


**Special case:**  $q_j \leq 2$



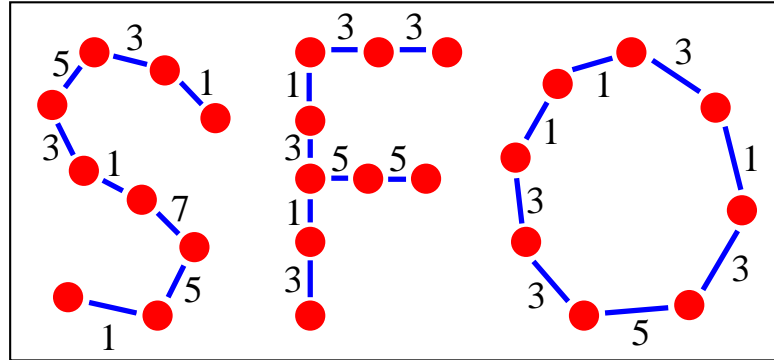
- Vertex  $s$  = processor  $s$ ,  $0 \leq s < p$
- Edge  $(s, t)$  = processor pair sharing matrix columns
- Edge weight  $w(s, t)$  = number of matrix columns shared

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

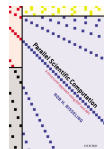




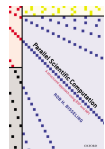
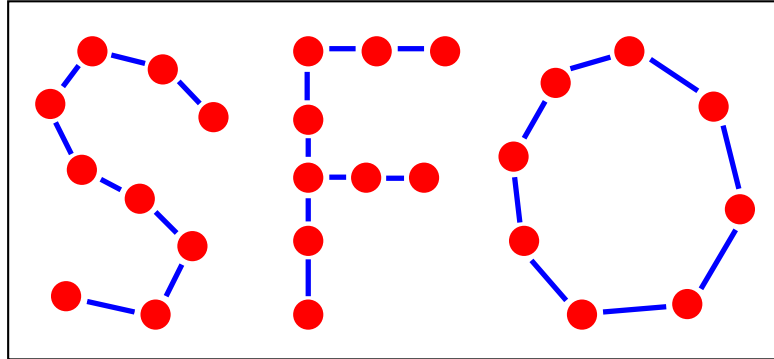
# Transform into unweighted undirected graph



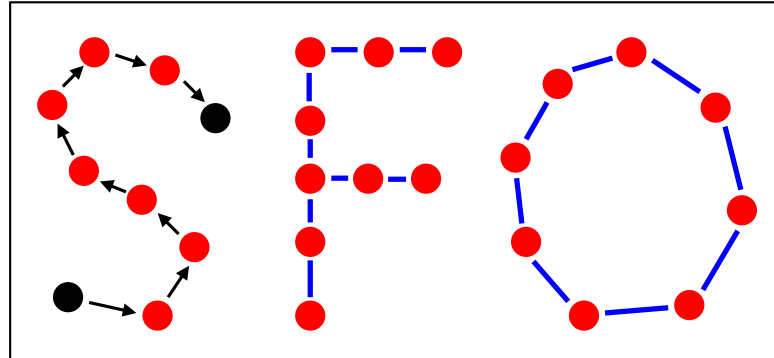
- Assign two shared columns: one to processor  $s$ , one to  $t$ .  
 $w(s, t) := w(s, t) - 2$ .
- Repeat until all edge weights = 0 or 1.



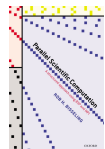
# *Unweighted undirected graph*



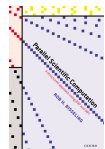
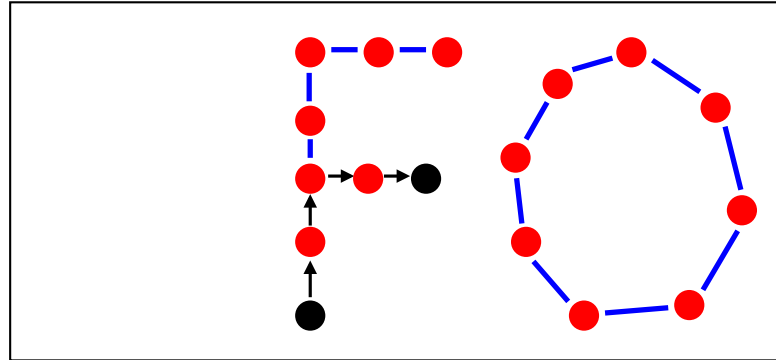
# *Transform into directed graph*



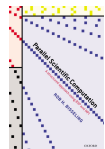
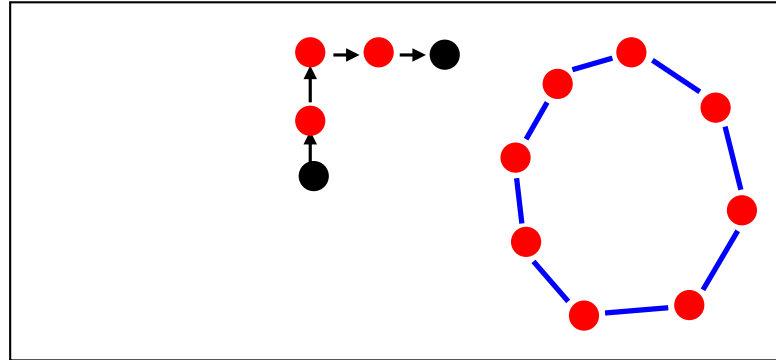
- Walk path starting at odd-degree vertex
- Remove walked edges from undirected graph
- Edge  $s \rightarrow t$ : processor  $s$  sends,  $t$  receives
- Even-degree vertices remain even-degree
- Repeat until all degrees in undirected graph are even.



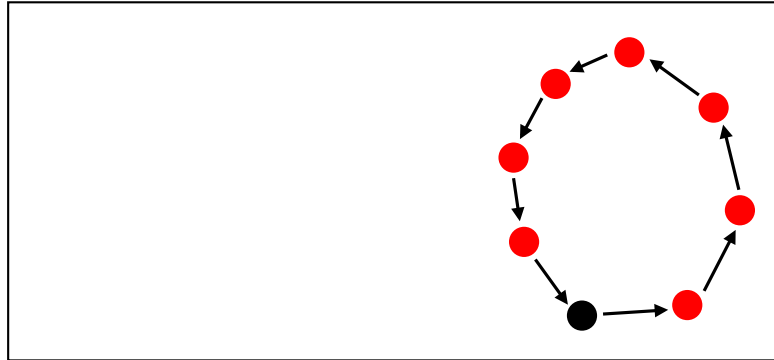
# *Transform into directed graph*



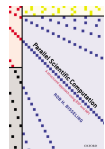
# *Transform into directed graph*



# *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.

