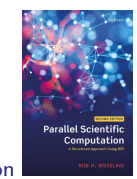
Vector distribution

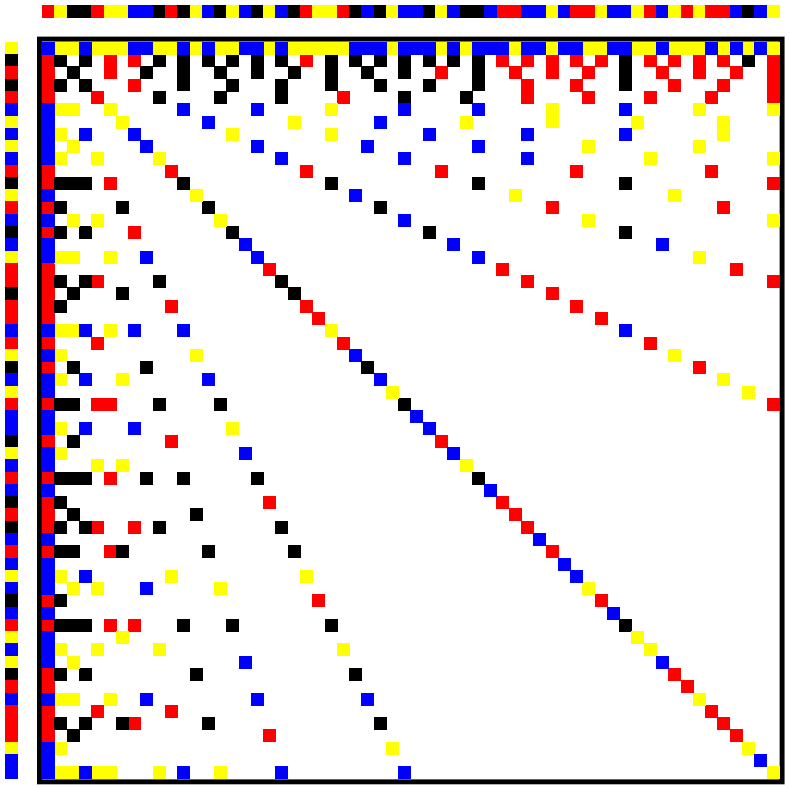
Section 4.7 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

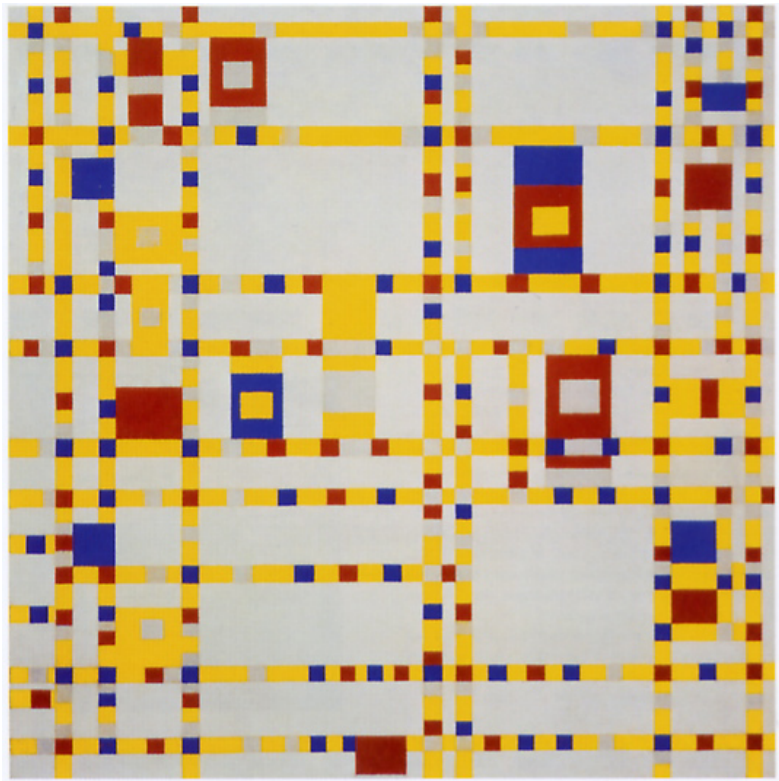
Utrecht University



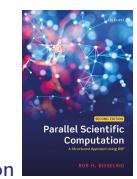
Matrix and vector distribution



Matrix and vector distribution for `prime60`



Broadway Boogie Woogie
Piet Mondriaan 1943



Balancing the communication

- Our aim is to reduce the BSP cost hg, where

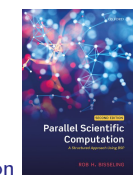
$$h = \max_{0 \leq s < p} h(s), \qquad h(s) = \max(h_{\mathrm{s}}(s), h_{\mathrm{r}}(s)).$$

- Thus, given a matrix distribution ϕ, we have to find a vector distribution $\phi_{\mathbf{v}}$ that
  - minimizes h for the fanout, thus balancing the communication,
  - satisfies the consistency constraint

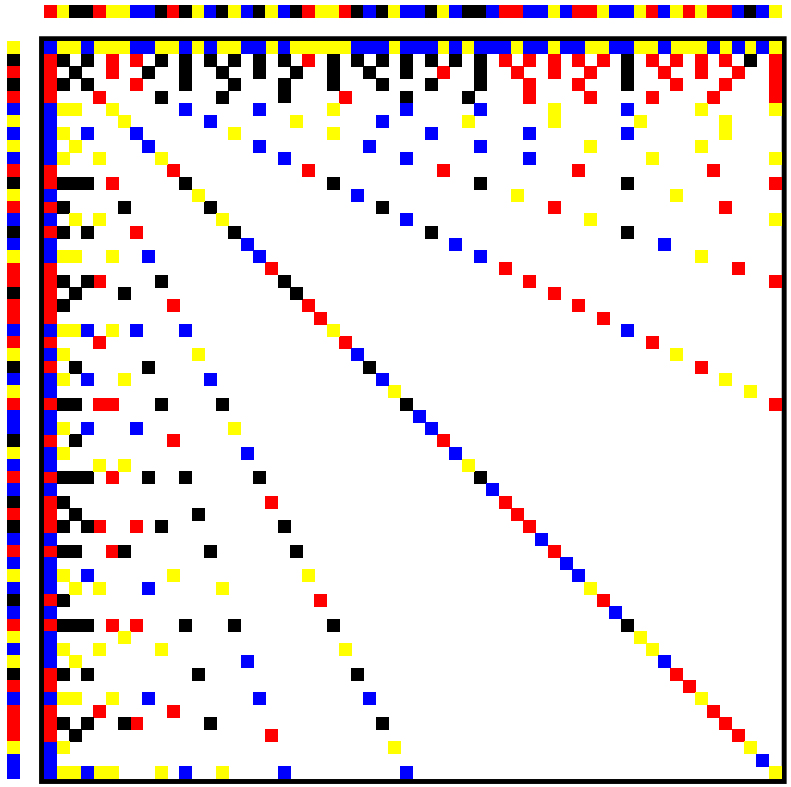
$$j \in J_{\phi_{\mathbf{v}}(j)}, \quad \text{for } 0 \leq j < n.$$

- The constraint means: the 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 minimizes h for the fanin and satisfies the constraint

$$i \in I_{\phi_{\mathbf{u}}(i)}, \quad \text{for } 0 \leq i < n.$$



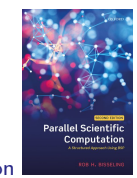
Global view of `prime60` distribution



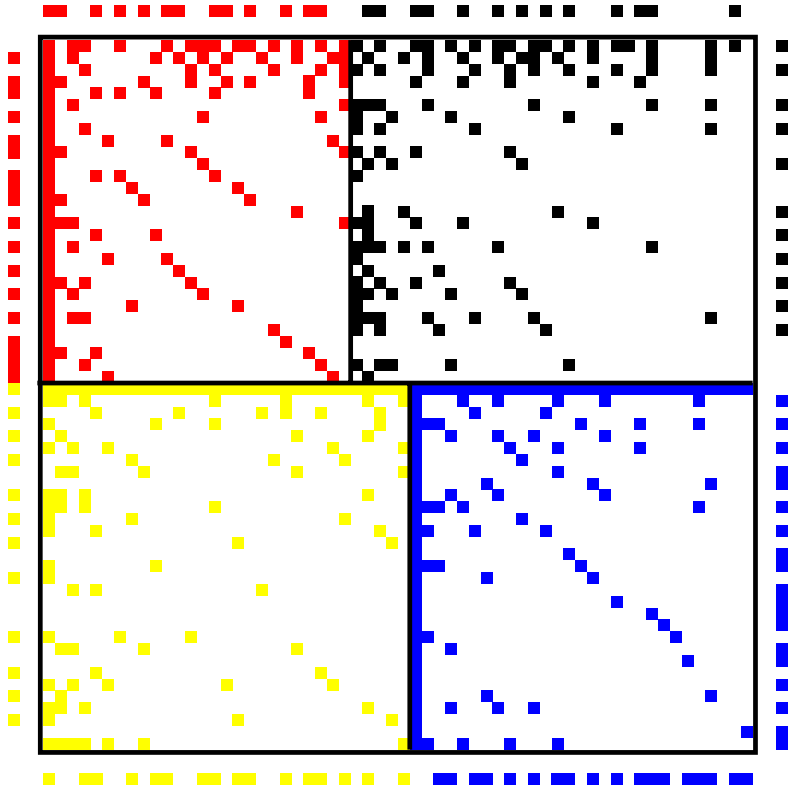
- Matrix `prime60` is defined by

$$a_{ij} \neq 0 \quad \text{if } i \bmod j = 0 \vee j \bmod i = 0, \quad \text{for } 1 \leq i, j < n.$$

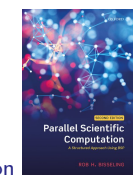
- We start counting at 1, for once!
- Both consistency constraints are satisfied. Check this for row and column 59 (a prime) and 60.



Local view of `prime60` distribution

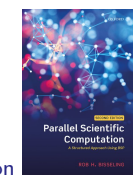


- The locally available components of the vector **u** are placed to the left for $P(0)$ and $P(2)$, and to the right for $P(1)$ and $P(3)$.
- Those of **v** are placed at the top for $P(0)$ and $P(1)$, and at the bottom for $P(2)$ and $P(3)$.



The two vector distribution problems are similar

- The nonzero pattern of row i of A equals the nonzero pattern of column i of A^{T}.
- Therefore, u_{is} is sent from $P(s)$ to $P(t)$ in the multiplication by A if and only if v_i is sent from $P(t)$ to $P(s)$ in the multiplication by A^{T}.
- Sending and receiving is equally costly in the BSP model.
- Therefore, we can find a good distribution $\phi_{\mathbf{u}}$ given $\phi = \phi_A$ by finding a good distribution $\phi_{\mathbf{v}}$ given $\phi = \phi_{A^{\mathrm{T}}}$.
- Hence, we need only one distribution method, namely for distributing $\mathbf{v}$. We can then apply this method for $\mathbf{u}$ with A^{T} instead of A.



General case: arbitrary μ_j values

- $\mu_j = \#$processors that need vector component v_j.
- Columns with $\mu_j = 0$ or $\mu_j = 1$ do not cause communication and can be omitted from the problem.
- Hence, we assume $\mu_j \geq 2$, for all j.
- For processor $P(s)$:

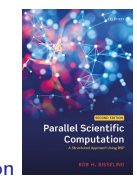
$$h_{\mathrm{s}}(s) = \sum_{\substack{j=0 \\ \phi_{\mathbf{v}}(j)=s}}^{n-1} (\mu_j - 1),$$

and

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

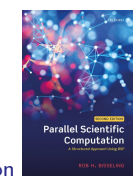
- Our aim: for a given matrix distribution ϕ and hence a given communication volume $V = V_\phi$, minimize

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



Egoistic local bound

- An egoistic processor tries to minimize its own $h(s) = \max(h_{\mathrm{r}}(s), h_{\mathrm{s}}(s))$ without consideration for others.
- To minimize $h_{\mathrm{r}}(s)$, it just has to maximize the number of components v_j with $j \in J_s$ that it owns.
- To minimize $h_{\mathrm{s}}(s)$, it has to minimize the total weight of these components, where the weight of v_j is $\mu_j - 1$.
- 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 $h_{\mathrm{s}}(s) \leq h_{\mathrm{r}}(s)$.

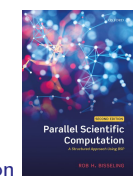


Optimal values

- 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

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

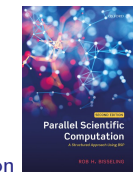
- The value $\hat{h}(s)$ is a local lower bound on the actual value that can be achieved: $\hat{h}(s) \leq h(s)$, for all s.



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
$\mu_j =$	2	2	2	2	3	3	3	3
$j =$	0	1	2	3	4	5	6	7

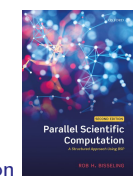
- In the table, a 1 denotes that $P(s)$ owns a nonzero in column j and hence needs v_j.
- Columns are ordered by increasing μ_j.
- Processor $P(0)$ wants v_0 and v_2, but nothing more, so that $\hat{h}_{\mathrm{s}}(0) = 2$, $\hat{h}_{\mathrm{r}}(0) = 4$, and $\hat{h}(0) = 4$.
- Other processors have a lower local bound $\hat{h}(s)$.
- The fanout will cost at least $4g$.



An algorithm based on the local bound

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

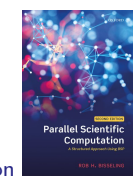
R. H. Bisseling and W. Meesen, *Electronic Transactions on Numerical Analysis* **21** (2005) pp. 47–65.



Initialization of the algorithm

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

- The matrix columns corresponding to J_s are assumed to be ordered by increasing μ_j.
- L_s is the index set of vector 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 the algorithm

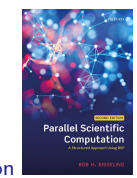
for $s := 0$ **to** $p - 1$ **do**
  if $h_{\mathrm{s}}(s) < \hat{h}_{\mathrm{s}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s))$ **then**
    $\mathrm{active}(s) := \mathit{true}$;
  else
    $\mathrm{active}(s) := \mathit{false}$;

- Note that $ns_0 \leq \hat{h}_{\mathrm{s}}(J, ns_0, nr_0)$, so that by substitution

$$h_{\mathrm{s}}(s) \leq \hat{h}_{\mathrm{s}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s)).$$

- A processor will not accept more components once it has achieved its optimum, i.e., when

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

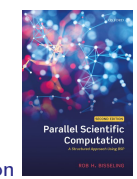


Main loop of the algorithm

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

{ Choose processor with highest local bound }
$s_{\max} := \text{argmax}_{0 \leq s < p}(\hat{h}_{\text{r}}(L_s, h_{\text{s}}(s), h_{\text{r}}(s)) : \text{active}(s))$;
$j := \min(L_{s_{\max}})$;
$\phi_{\mathbf{v}}(j) := s_{\max}$;

{ Update sends and receives }
$h_{\text{s}}(s_{\max}) := h_{\text{s}}(s_{\max}) + \mu_j - 1$;
for all $s : 0 \leq s < p \wedge s \neq s_{\max} \wedge j \in J_s$ **do**
    $h_{\text{r}}(s) := h_{\text{r}}(s) + 1$;



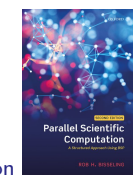
Main loop of the algorithm

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

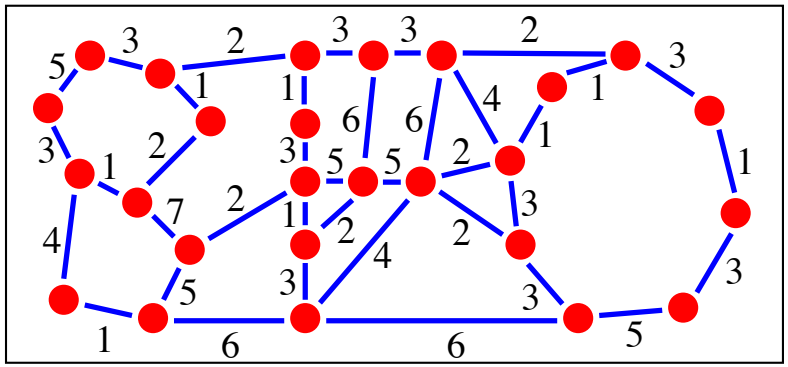
{ Choose processor with highest local bound }
$s_{\max} := \mathrm{argmax}_{0 \leq s < p}(\hat{h}_{\mathrm{r}}(L_s, h_{\mathrm{s}}(s), h_{\mathrm{r}}(s)) : \mathrm{active}(s))$;
$j := \min(L_{s_{\max}})$;
$\phi_{\mathbf{v}}(j) := s_{\max}$;

{ Update sends and receives }
$h_{\mathrm{s}}(s_{\max}) := h_{\mathrm{s}}(s_{\max}) + \mu_j - 1$;
for all $s : 0 \leq s < p \wedge s \neq s_{\max} \wedge j \in J_s$ **do**
    $h_{\mathrm{r}}(s) := h_{\mathrm{r}}(s) + 1$;

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



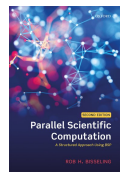
Special case: all $\mu_j \leq 2$



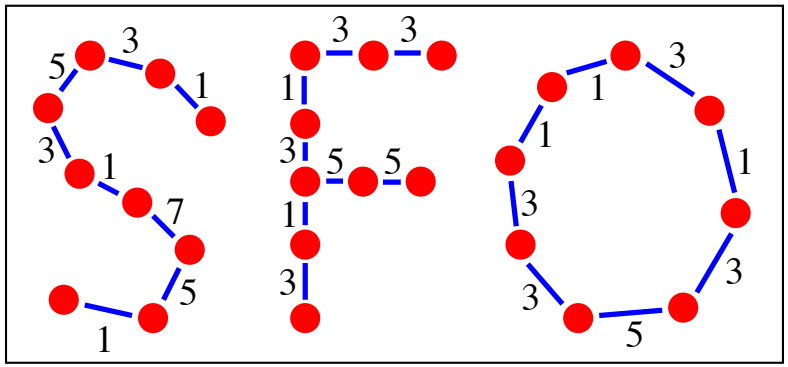
We first create a weighted undirected graph, the communication graph, where:

- vertex s = processor $P(s)$, for $0 \leq s < p$.
- edge (s, t) = processor pair $(P(s), P(t))$ sharing one or more matrix columns.
- edge weight $\omega(s, t)$ = the number of matrix columns shared.

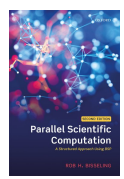
Problem: assign each matrix column (i.e., vector component) to a processor, while balancing the number of data words sent and received.



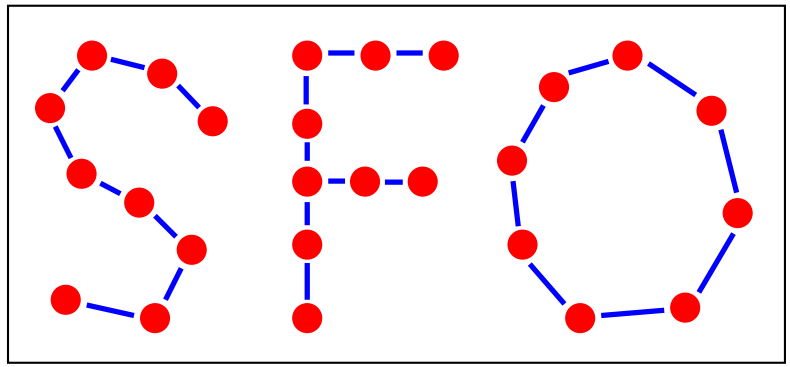
Reduce the weights by pairing matrix columns



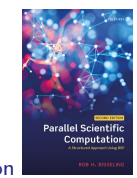
- Assign shared matrix columns in pairs: one column to processor $P(s)$, one to $P(t)$.
- Reduce the weight accordingly, by $\omega(s,t) := \omega(s,t) - 2$.
- Shown here: the result of doing this for edges with even weight.



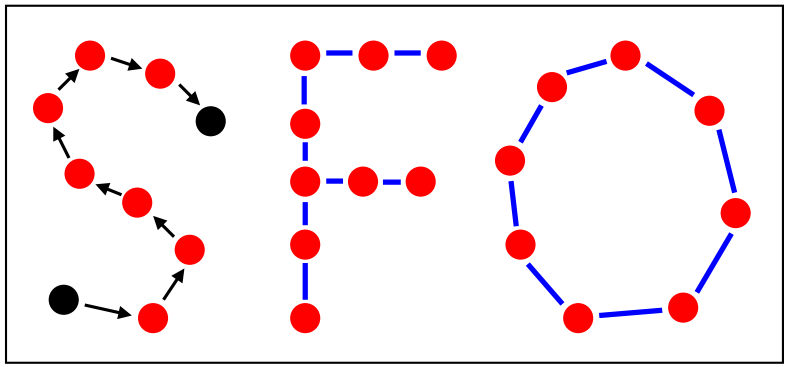
Transform the graph into an unweighted undirected graph



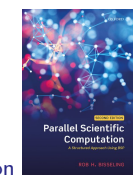
- Repeat the paired assignment until all edge weights are 0 or 1.
- The result can be viewed as an unweighted undirected graph.



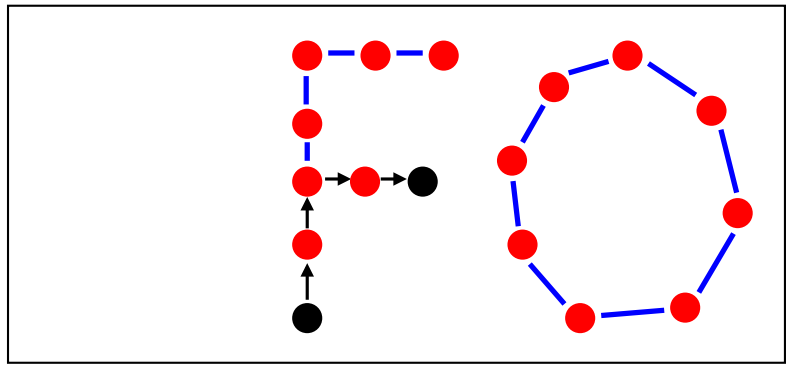
Transform the undirected graph into a directed graph

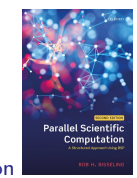


- Walk a path in the undirected graph starting at a vertex with an odd degree (number of incident edges).
- Transform each walked edge (s, t) into a directed edge $s \to t$, which means that processor $P(s)$ sends and $P(t)$ receives.
- Walk until you reach a dead end.
- Note that even-degree vertices remain even-degree.
- Repeat the procedure until all degrees in the undirected graph are even.

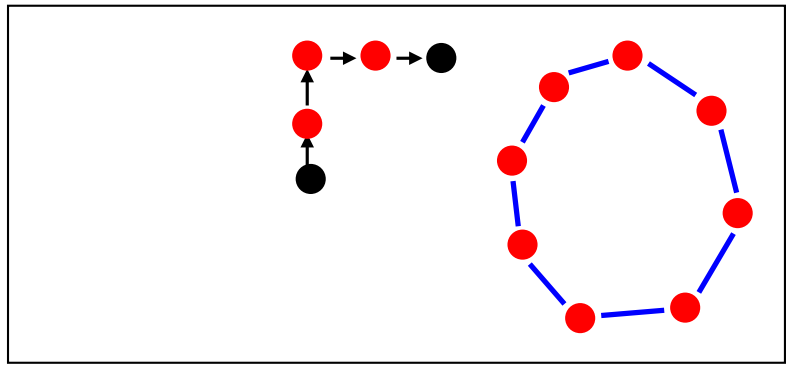


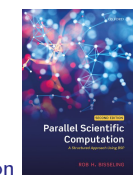
Transform into a directed graph (starting at odd degree)



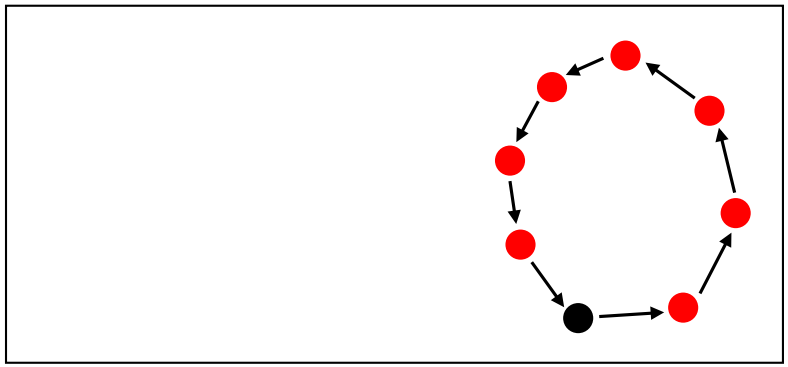


Transform into a directed graph (starting at odd degree)

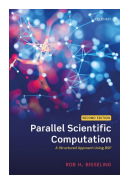




Transform into a directed graph (starting at even degree)



- Final phase (no more odd-degree vertices): walk a path in the undirected graph starting at a vertex with an even degree.
- Repeat the procedure until the undirected graph has no more edges.
- The resulting vector distribution is provably optimal.



Summary

- The BSP cost is a natural metric that encourages communication balancing.
- The general vector distribution problem is NP-complete, as shown by Ali Pinar. We have developed a heuristic method that works well in practice.
- This 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 communication graph.
- Here, optimal does not mean perfect, because some processors may still communicate more than others.