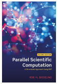
Parallel Graph Matching

Section 5.4 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

Utrecht University



p-way vertex partitioning

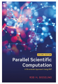
- A p-way vertex partitioning $\mathcal{V}_0, \ldots, \mathcal{V}_{p-1}$ is a set of p nonempty subsets of $\mathcal{V}$ that satisfy

$$\mathcal{V} = \bigcup_{s=0}^{p-1} \mathcal{V}_s,$$

and

$$\mathcal{V}_s \cap \mathcal{V}_t = \emptyset, \quad \text{for all } s \neq t.$$

- $\mathcal{V}_s$ is the local vertex set of processor $P(s)$.
- $\phi(v)$ is the processor number of vertex v.
- The adjacency list Adj_v of a vertex v is stored together with v on processor $P(\phi(v))$.
- Adj_v may contain vertices u from another processor.

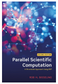


Halo vertices

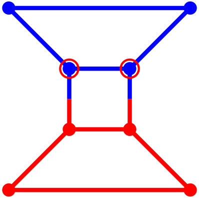
- The halo of a processor is the set of surrounding data that interact directly with the processor, causing communication.
- The halo set of processor $P(s)$ is the vertex set

$$\mathcal{H}_s = \left(\bigcup_{v \in \mathcal{V}_s} Adj_v \right) \setminus \mathcal{V}_s.$$

- For a good partitioning, $|\mathcal{H}_s| \ll |\mathcal{V}_s|$.



Internal and external edges

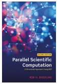


$n = 8,\ m = 10,\ p = 2$

- $P(0)$ owns four red vertices.
- Its halo set $\mathcal{H}_0$ consists of two blue vertices marked by a red circle.
- The edge set of processor $P(s)$ is

$$\mathcal{E}_s = \{(u, v) \in \mathcal{E} : v \in \mathcal{V}_s\}.$$

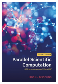
- The edge set of $P(0)$ consists of:
  - 4 internal edges, with both ends in $\mathcal{V}_0$, shown in red;
  - 2 external edges (cut edges), with one end in $\mathcal{V}_0$ and one in $\mathcal{H}_0$, shown as pairs of red/blue edges.



Parallel local domination algorithm for $P(s)$: main loop

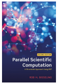
$\mathcal{M}_s := \emptyset;$ ▷ matches
$R_s := \emptyset;$ ▷ received messages
$Q_s := \mathcal{V}_s;$ ▷ queue of vertices

done $:=$ **false**;
while not done do
  done$_s := (R_s = \emptyset \wedge Q_s = \emptyset);$
  put **done**$_s$ in $P(*)$;
  $\mathrm{ProcessReceivedMessages}(R_s, Q_s, \mathcal{M}_s, \mathcal{V}_s, \omega, \ldots);$
  while $Q_s \neq \emptyset$ **do**
    pick a vertex $v \in Q_s$;
    ...
  Sync;
  done $:= \bigwedge_{t=0}^{p-1}$ **done**$_t$;



Detecting termination of the algorithm

- The algorithm terminates when all processors have an empty receive buffer R_s and an empty work queue Q_s.
- Received messages can give rise to new work, hence both R_s and Q_s must be empty when declaring the local work done.
- Termination is expressed in a boolean variable **done**, which is true if all local booleans $\mathbf{done}_s$ are true.
- This can be checked without requiring extra synchronization by broadcasting the local booleans once every superstep.



Parallel local domination algorithm for $P(s)$: inner loop

while $Q_s \neq \emptyset$ **do**

  pick a vertex $v \in Q_s$; $Q_s := Q_s \setminus \{v\}$;

  $\mathrm{FindAlive}(v, Adj_v, \omega, alive, suitor, d)$;

  $r := \mathrm{FindSplitter}(v, Adj_v, \omega, alive, splitter_v, suitor, d)$;

  $(u, v) := \mathrm{FindPref}(Adj_v, \omega, r, d_v - 1)$;

  $d_v := d_v - 1$;

  $pref(v) := u$;

  if $u = suitor(v)$ **then**   ▷ Register a match or propose

    $\mathcal{M}_s := \mathcal{M}_s \cup \{(u, v)\}$;

    $d_v := 0$;

    if $u \in \mathcal{V}_s$ **then**

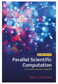
      $d_u := 0$;

    else

      put $\mathrm{accept}(v, u)$ in $P(\phi(u))$;

  else if $u \notin \mathcal{V}_s$ **then**

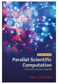
    put $\mathrm{propose}(v, u)$ in $P(\phi(u))$; ...



How to propose

Source: The Guardian, June 1, 2010.
Photo by Getty.

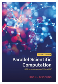
- $\mathrm{propose}(v, u)$ means: v proposes to u



How not to propose

IJsselstein, the Netherlands. Source: ANP, December 13, 2014.

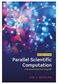
- No one got hurt, she accepted, and they ran off to Paris to celebrate.
- $\mathrm{accept}(v, u)$ means: v accepts u



Mixed superstep

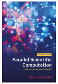
...
put $\mathrm{accept}(v, u)$ in $P(\phi(u))$;
...
put $\mathrm{propose}(v, u)$ in $P(\phi(u))$;
...

- The strongest point of BSP for graph computations: we can freely mix computation and communication and initiate communication from anywhere in the algorithm.
- Still, we achieve a superstep structure by assuming delayed communication executed at the next synchronization (**Sync**).
- This helps us in thinking about algorithms, analysing their time complexity, and proving their correctness.



One-sided communication gives flexibility

- One-sided communication is the basis for the ability to send data from anywhere in a program text, without any worries about corresponding receive operations.
- In contrast, think of the horrors of using two-sided communication: we would have to match send-statements hidden somewhere with receive-statements hidden somewhere else.



Inner loop (cont'd)

while $Q_s \neq \emptyset$ **do**

  …

  $\mathit{pref}(v) := u$;

  …

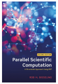
  if $u \in \mathcal{V}_s$ **then** ▷ Replace the previous suitor

    $x := \mathit{suitor}(u)$;

    $\mathit{suitor}(u) := v$;

    RejectSuitor($u, x, Q_s, \mathcal{V}_s, \mathit{alive}, \mathit{pref}$)

  $\mathrm{SplitAdj}(\mathit{Adj}_v, \omega, \mathit{splitter}_v, r, d_v - 1)$; ▷ Split adjacency list

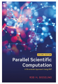


Rejecting a suitor

function $\textsc{RejectSuitor}(v, x, Q_s, \mathcal{V}_s, \mathit{alive}, \mathit{pref})$

  if $x \neq$ **nil** **then**
    if $x \in \mathcal{V}_s$ **then**
      $Q_s := Q_s \cup \{x\}$;
      $\mathit{pref}(x) :=$ **nil**;
    else
      put $\mathrm{reject}(v, x)$ in $P(\phi(x))$;
    $\mathit{alive}(v, x) :=$ **false**;

- $\mathrm{reject}(v, u)$ means: v rejects u



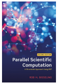
Processing received messages: main loop

function $\textsc{ProcessReceivedMessages}(R_s, Q_s, \ldots)$

while $R_s \neq \emptyset$ **do**
    pick a message $msg \in R_s$;
    $R_s := R_s \setminus \{msg\}$;

    if $msg = \mathrm{propose}(u, v)$ **then**
        { Register a match }
        if $u = pref(v)$ **then**
            $\mathcal{M}_s := \mathcal{M}_s \cup \{(u, v)\}$;
            $d_v := 0$;

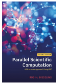
        ...
    else if $msg = \mathrm{accept}(u, v)$ **then**
        ...
    else if $msg = \mathrm{reject}(u, v)$ **then**
        ...



Remember your proposals!

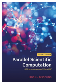
if $msg = \mathrm{propose}(u, v)$ **then**
  { Register a match }
  if $u = pref(v)$ **then**
    $\mathcal{M}_s := \mathcal{M}_s \cup \{(u, v)\}$;
  ...

- If v prefers a remote u and sends a proposal to u, it needs to remember this. Just as in real life.
- In the parallel algorithm, we need to store both $suitor(v)$ and $pref(v)$ for each local vertex v, because suitor information is spread across different processors.



Reasoning with supersteps

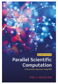
- In case u proposes to v, where v has already proposed to u, this will be detected by the condition $u = pref(v)$.
- The proposal by v to u must have been sent simultaneously with the proposal by u to v in the previous superstep.
- It cannot have been sent earlier, because in that case u would have answered with an accept message instead of sending a proposal.
- Here, our reasoning is based on supersteps that
  - first process received messages;
  - after that, set preferences and send proposals.
- The proposal is then tacitly accepted, without sending an accept message, because both sides know about the match.



Processing a proposal: the complete text

if $msg = \mathrm{propose}(u, v)$ **then**
    { Register a match }
    if $u = pref(v)$ **then**
        $\mathcal{M}_s := \mathcal{M}_s \cup \{(u, v)\}$;
        $d_v := 0$;

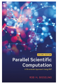
    { Assign new suitor }
    $x := suitor(v)$;
    if $\omega(u, v) > \omega(x, v)$ **then**
        $suitor(v) := u$;
        RejectSuitor$(v, x, Q_s, \mathcal{V}_s, alive, pref)$
    else
        put $\mathrm{reject}(v, u)$ in $P(\phi(u))$;
        $alive(u, v) :=$ **false**;



Processing an accept message

if $msg = \mathrm{accept}(u, v)$ **then**
  $\mathcal{M}_s := \mathcal{M}_s \cup \{(u, v)\};$
  $d_v := 0;$
  $x := suitor(v);$
  $suitor(v) := u;$
  $\mathrm{RejectSuitor}(v, x, Q_s, \mathcal{V}_s, alive, pref)$

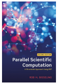
- If u accepts v, the match (u, v) is registered, the degree d_v of v is set to 0, and the previous suitor x is rejected.
- To ward off others, u is still registered as the suitor of v.



Processing a reject message

if $msg = \mathrm{reject}(u, v)$ **then**
    $Q_s := Q_s \cup \{v\}$;
    $pref(v) :=$ **nil**;
    $alive(u, v) :=$ **false**;

- If u rejects v, the vertex v is reinserted into the queue, its preference is reset to **nil**, and the edge (u, v) is declared dead.



Summary

- We parallelized the local domination algorithm by partitioning the vertex set $\mathcal{V}$ into subsets $\mathcal{V}_s$.
- Each processor $P(s)$ obtains a vertex set $\mathcal{V}_s$, a halo set

$$\mathcal{H}_s = \{u \in \mathcal{V} \setminus \mathcal{V}_s : (\exists v \in \mathcal{V}_s \ : \ (u, v) \in \mathcal{E})\},$$

  and an edge set

$$\mathcal{E}_s = \{(u, v) \in \mathcal{E} : v \in \mathcal{V}_s\}.$$

- The parallel algorithm is based on mixed supersteps, where communication can conveniently be initiated from anywhere within the superstep.
- Each superstep starts with processing received messages, of type propose, accept, or reject; then, it repeatedly sets preferences; and finally, it sends out new messages.