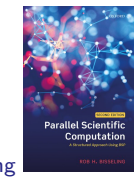
Tie-breaking and Load Balancing

Sections 5.6–5.7 of Parallel Scientific Computation, 2nd edition

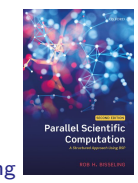
Rob H. Bisseling

Utrecht University

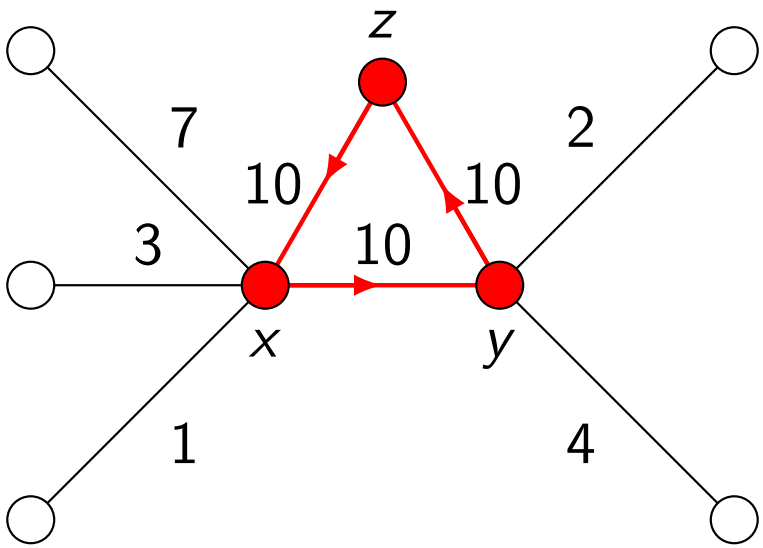


Ties

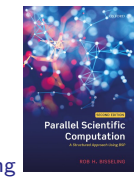
- Ties are edges with equal weight. For ease of explanation, we assumed so far that these do not occur.
- Ties are only a problem if they share a vertex, since then that vertex has no unique preference.
- In practice, equal weights do occur. In the worst case, all weights could be equal, which amounts to cardinality matching.
- Our algorithm should work for such cases as well.



Preference cycle of length 3



- 3 preferences have been set, along 3 dominant edges.
- Still, no matches are made.



Possible solution: break ties by a secondary weight

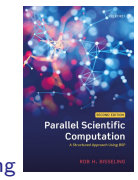
- Define the secondary weight of an edge (u, v) by

$$\omega_2(u, v) = u + v, \text{ for } 0 \leq u, v < n,$$

  where we identify vertices with their index.
- We only need to compare edges that share a vertex v, which simplifies to:

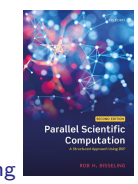
$$\begin{aligned} \omega_2(u, v) > \omega_2(u', v) &\iff u + v > u' + v \\ &\iff u > u'. \end{aligned}$$

- Thus, preferring the highest-indexed partner breaks all ties and prevents cycles.
- This solution works for both the sequential and the parallel algorithm.

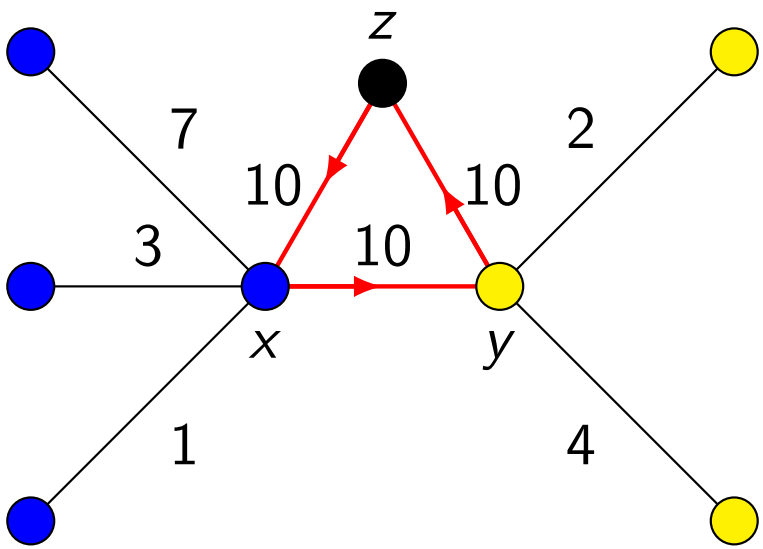


Is it a feature of a bug?

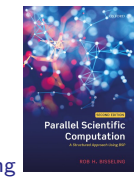
- Encountering equal preferences: is it undesired behaviour for an algorithm or an opportunity?
- In the sequential case, we can exploit equal weights, finding matches earlier than by making weights unique.
- When setting a preference for vertex v:
  - if $u = \mathit{suitor}(v)$ is among the tied preferences with the highest weight, we match v to u;
  - if u is not among them, or there is no suitor yet, we break ties arbitrarily.
- We cannot do this in the parallel case.



Preference cycle on 3 processors



- In one superstep: x proposes to y, y proposes to z, and z proposes to x.
- In the next superstep, all three will have a suitor: $suitor(y) = x$, $suitor(z) = y$, and $suitor(x) = z$.
- No matches are made, and no more preferences will be set by x, y, and z.

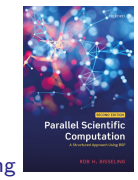


Enhancing locality

- Define another weight of an edge (u, v) by

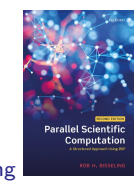
$$\omega_1(u, v) = \begin{cases} 1 & \text{if } \phi(u) = \phi(v) \\ 0 & \text{otherwise.} \end{cases}$$

- Using ω_1 as a secondary criterion leads to setting more local preferences, thereby reducing the communication of proposals and rejects, which are caused by nonlocal preferences.
- This saves communication time and prevents delays because local preferences can be checked immediately, e.g. to see whether they are mutual.

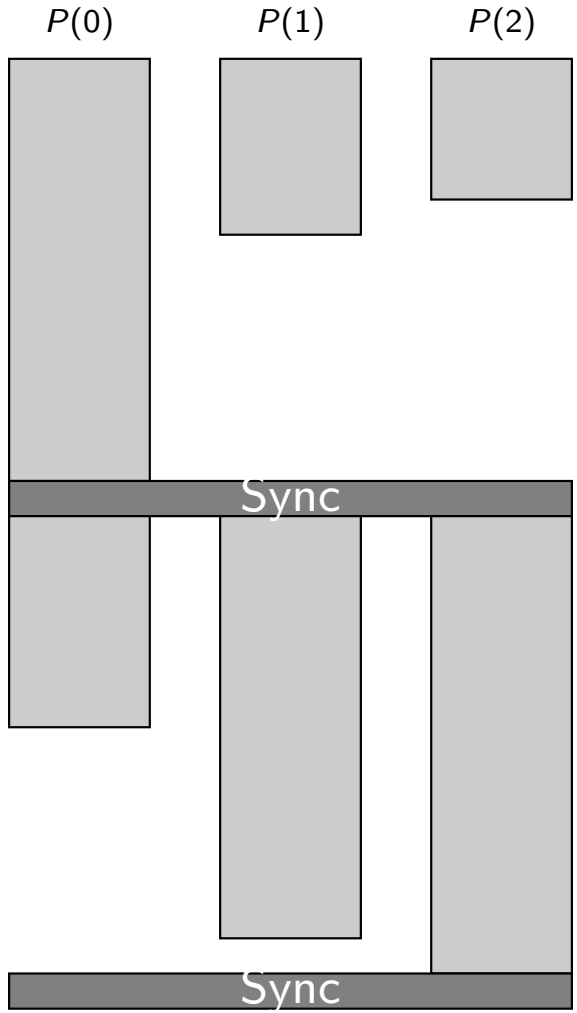


Same quality

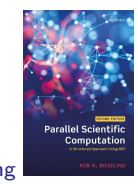
- We use $\omega, \omega_1, \omega_2$ (weight, locality, index) as primary, secondary, and tertiary criterion, respectively.
- Enhancing locality by using ω_1 as a secondary criterion gives the same quality as using ω_2, because all ties have equal weight and are therefore, in principle, equally good.
- This quality would not be guaranteed if we would use ω_1 as a primary criterion: the boy or girl next door would then be preferred even if there were a better match in another village.
- Enhancing locality introduces a new form of nondeterminism: the matching produced may be different for different p.
- Still, the matching guarantees at least half the optimal weight.



Load imbalance



- In a superstep, work queues may differ a lot in size, even when the partitioning is well-balanced.

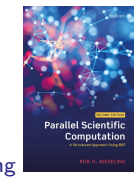


Load-balancing mechanism

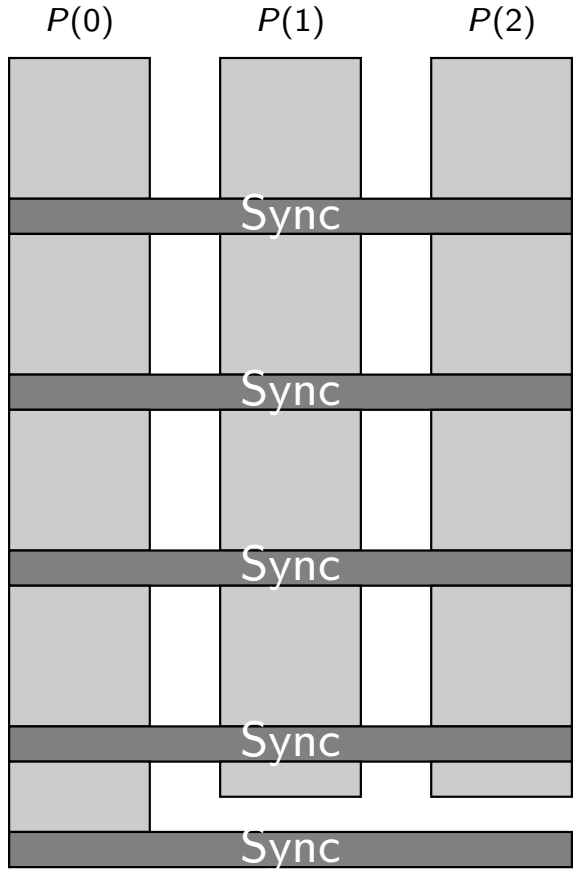
- We need not wait for the work queue to be empty to call for a synchronization!
- Synchronizations refill the work queues.
- Every processor $P(s)$ can keep track of its amount of work W_s carried out so far in the current superstep by inserting operation counters into the algorithm, or by using range sizes.
- We should call for a synchronization when

$$W_s \geq W_{\max} \vee Q_s = \emptyset,$$

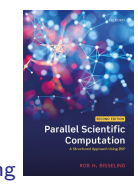
where $W_{\max}$ is the maximum amount of work to be done until the next synchronization.



Inserting synchronizations



- Inserting synchronizations keeps all processors busy doing useful work until near the end of the algorithm.

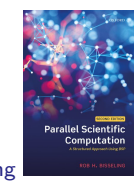


Robust load balancing

- We can use the work counter $W_{\max}$ to determine when to look at the system clock of $P(s)$ to obtain the time t_s elapsed since the start of the superstep.
- We call for synchronization if

$$t_s \geq t_{\max} \vee Q_s = \emptyset,$$

  where $t_{\max}$ is a carefully chosen maximum superstep time.
- Using the work counter prevents us from looking at the clock all the time, which itself consumes precious computation time.



How often should we synchronize?

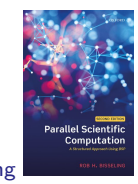
- We should not synchronize too often, to reduce the total synchronization cost.
- We will never spend more than 20% of the total time synchronizing if we choose

$$W_{\max} \geq 4l.$$

- To minimize synchronization overhead, we can communicate the remaining queue sizes at the end of a superstep, and take for the next superstep

$$W_{\max} \geq \min_s |Q_s|,$$

where $W_{\max}$ must then be expressed as a maximum number of preferences to be set.

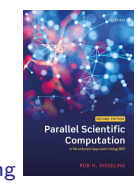


Communication cost

- The communication cost is hard to analyse exactly, because it depends not only on the static partitioning, but also on the dynamic information flow of the algorithm.
- The partitioning determines which edges are cut and hence can give rise to a proposal and an answer, or a mutual proposal. The flow of the algorithm decides which messages are actually sent.
- The edge cut of the vertex partitioning is defined as

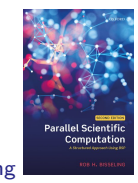
$$EC_\phi = |\{(u, v) \in \mathcal{E} : \phi(u) \neq \phi(v)\}|.$$

- An upper bound on the total communication volume of the algorithm is $2EC_\phi$.



Think like a vertex

- For a good partitioning, the edge cut EC_ϕ will be small, so that most preferences will be local and the number of proposals sent will be limited.
- For a bad partitioning, all preferences may turn out to be nonlocal, and communication will most likely be dominant.
- Systems like Google's Pregel and Apache Giraph, with the motto 'think like a vertex', impose $p = n$, identifying every vertex with a virtual processor,.
- They accept the fact that all edges between vertices cause communication.
- If it thinks like a vertex, it talks like a vertex!



Summary

- Ties in the matching can best be broken by preferring local vertices.
- Any remaining ties can be broken by choosing the vertex with the highest vertex number.
- The load balance can be improved by synchronizing more often, which refills the work queues and keeps all processors busy doing useful work.
- The optimal synchronization frequency is a trade-off between the cost of the synchronization itself and the imbalance caused by waiting for a synchronization.
- The edge cut of a vertex partitioning ϕ equals

$$EC_\phi = |\{(u, v) \in \mathcal{E} : \phi(u) \neq \phi(v)\}|.$$

- An upper bound on the total communication volume of the parallel matching algorithm is $2EC_\phi$.