Algorithm: Parallel BFS (Push mode, frontier-based)

Input: Graph $G = (V, E)$, source node s
Output: Distance $\boldsymbol{dist}$ for each node from the source node s

foreach $u \in [0, |V| - 1]$ **do**
    $cur\_ftr[u] = 0$;
    $nxt\_ftr[u] = 0$;
    $dist[u] = \infty$;
$dist[s] = 0$; $cur\_ftr[s] = 1$;
$level = 0$;
while $\exists\, u \in V$ *s.t.* $cur\_ftr[u] == 1$ **do**
    foreach u *s.t.* $cur\_ftr[u] == 1$ **do**
        foreach $v \in Nbr(u)$ **do**
            if $dist[v] == \infty$ **then**
                $dist[v] == level + 1$;
                $nxt\_ftr[v] = 1$;
    $level = level + 1$;
    Clear(cur_ftr);
    Swap(cut_ftr, nxt_ftr);

Algorithm: Parallel SSSP (Bellman-Ford)

Input: Graph $G = (V, E, W)$[W: integral edge weights], source node s
Output: Distance $\boldsymbol{dist}$ for each node from the source node s

foreach $u \in [0, |V| - 1]$ **do**
    $dist[u] = \infty$;
$dist[s] = 0$;
do
    $exist = false$;
    foreach $u \in [0, |V| - 1]$ **do**
        foreach $v \in Nbr(u)$, *edge* (u, v) *with* $w = W(u, v)$ **do**
            if $dist[u] + w < dist[v]$ **then**
                $exist = true$;
                $dist[v] = dist[u] + w$;
while $exist = true$;

Algorithm: Parallel Closeness Centrality

Input: Undirected Connected Graph $G = (V, E)$, source node s
Output: Closeness Centrality(CC) of the source node s

foreach $u \in [0, |V| - 1]$ **do**
    $cur\_ftr[u] = 0$;
    $nxt\_ftr[u] = 0$;
    $dist[u] = \infty$;
$dist[s] = 0$; $cur\_ftr[s] = 1$;
$level = 0$;
while $\exists\, u \in V$ *s.t.* $cur\_ftr[u] == 1$ **do**
    foreach u *s.t.* $cur\_ftr[u] == 1$ **do**
        foreach $v \in Nbr(u)$ **do**
            if $dist[v] == \infty$ **then**
                $dist[v] == level + 1$;
                $nxt\_ftr[v] = 1$;
    $level = level + 1$;
    Clear(cur_ftr);
    Swap(cut_ftr, nxt_ftr);
$dist\_sum = 0$;
foreach $u \in [0, |V| - 1]$ **do**
    $dist\_sum += dist[u]$;
$dist\_avg = dist\_sum/(|V| - 1)$;
$CC(s) = 1/dist\_avg$;

Algorithm: Parallel WCC

Input: Graph $G = (V, E)$
Output: Component id $\boldsymbol{wcc}$ for each node

foreach $u \in [0, |V| - 1]$ **do**
    $wcc[u] = u$;
do
    $exist = false$;
    foreach $u \in [0, |V| - 1]$ **do**
        foreach $v \in Nbr(u)$ **do**
            if $wcc[u] < wcc[v]$ **then**
                $exist = true$;
                $wcc[v] = wcc[u]$;
while $exist = true$;

Algorithm: Parallel SpMV (Tiling-based processing)

Input: Graph $G = (V, E, W)$[W: matrix value assigned to edges], input vector $VectorIn$

Output: Output vector $VectorOut$

foreach $u \in [0, |V| - 1]$ **do**
    $sum = 0;$
    foreach $v \in Nbr(u)$, *edge* (u, v) *with* $w = W(u, v)$ **do**
        $sum = sum + w * VectorIn[v];$
        $VectorOut[u] = sum;$

Algorithm: Parallel PageRank (Tiling-based processing)

Input: Graph $G = (V, E)$, damping factor d, #iterations N

Output: Rank vector $\boldsymbol{R[; N]}$ after N iterations

$iter = 0;$
foreach $u \in [0, |V| - 1]$ **do**
    $R[u; 0] = 1/|V|;$
for i ***from 1 to*** N **do**
    $rank = (1 - d)/|V|;$
    foreach $v \in Nbr(u)$ **do**
        $rank = rank + d/deg(v) * R[v; iter - 1];$
    $R[u; iter] = rank;$