Eric Aubanel

Elements of Parallel Computing[1] *Errata*

CHAPTER 1

Errata

1.1 JULY 23, 1017

Note: corrected algorithms appear on the following pages.

1. Bug in Algorithm 3.3: redundantly initializes and updates k.
2. Algorithm 4.18: the last 2 lines are wrong, as *group* doesn't include source. Need to do two broadcasts: one among columns 1 to $\sqrt{p}-1$ and then one among first column. See implementation in `matVec2DMPI.c` at `https://github.com/eaubanel/ElementsOfParallelComputing.git`. Also need to slightly modify discussion on p. 120, since 2 broadcasts needed (doesn't change complexity).
3. Algorithm 4.19: didn't check to avoid communication within process, on lines 22, 26, 36.
4. Algorithm 5.1: line before scan(b) should be $b[id] \leftarrow a[(id+1) * n/p - 1]$.
5. Algorithm 6.5: $F_N[i]$ should be $F_N[i] + j$ on lines 8 and 9.

Algorithm: 3.3 (corrected)

Input: array $a[lower..upper-1]$, with each subarray $i \in [lower..mid)$ and $i \in [mid..upper)$ sorted in ascending order.
Output: sorted array b
Procedure `merge`(a, $lower$, mid, $upper$, b)
    $i \leftarrow lower$
    $j \leftarrow mid$
    $k \leftarrow 0$
    for $k \leftarrow lower$ *to* $upper-1$ **do**
        if $i < mid \wedge (j \geq upper \vee a[i] \leq a[j])$ **then**
            $b[k] \leftarrow a[i]$
            $i \leftarrow i+1$
        else
            $b[k] \leftarrow a[j]$
            $j \leftarrow j+1$
        end
    end
end

Algorithm: 4.18 (corrected)

`// Assume` $p = q^2$`, matrix is square,` $n \bmod q = 0$
$q \leftarrow \sqrt{p}$
$nb \leftarrow n/q$
for $i \leftarrow 0$ *to* $n/q-1$ **do**
    $c[i] \leftarrow 0$
    for $j \leftarrow 0$ *to* $n/q-1$ **do**
        $c[i] \leftarrow c[i] + a[i,j] * b[j]$
    end
end
$rowID \leftarrow \lfloor id/q \rfloor$
$colID \leftarrow id \bmod q$
$destID \leftarrow rowID * q$
$group \leftarrow [destID..destID+q-1]$
`reduce`($destID$, c, nb, sum, c, $group$)
$sourceID \leftarrow colID * q$
$group \leftarrow [sourceID, colID, colID+q, .., coldID+(q-1)*q]$
`broadcast`($sourceID$, c, nb, c, $group$)
$group \leftarrow [0, q, .., (q-1)*q]$
`broadcast`(0, c, nb, c, $group$)

Algorithm: 4.19 (corrected)

Input: Array $s[1..n]$ of n positive integers, target sum S
Output: Completed array F
Data: Array $F[1..n, 0..nb-1]$ initialized to 0 (nb is number of columns owned by task), array $L[0..\lceil S/p \rceil - 1]$ to store received messages

```
1  broadcast(0,s,n,s)
2  myFirst ← ⌊id * S/p⌋ + 1
3  myLast ← ⌊(id + 1) * S/p⌋
4  if id = 0 then //first block has one extra value
5      myFirst ← 0
6      F[1, 0] ← 1
7  end
8  nb ← myLast − myFirst + 1
9  nL ← ⌈S/p⌉
10 if id = findID(s[1] − 1, p, S) then  F[1, s[1] − myFirst] ← 1
11 for i ← 2 to n do
12     id1 ← findID(myFirst + s[i] − 1, p, S)
13     id2 ← findID(myLast + s[i] − 1, p, S)
14     if id1 < p then
15         if id1 = id2 then
16             myLocalBegin ← max(0, ⌊id1 * S/p⌋ + 1 − s[i] − myFirst)
17             send F[i − 1, myLocalBegin..nb − 1] to id1
18         else
19             destBegin ← myFirst + s[i]
20             destLast ← ⌊(id1 + 1) * S/p⌋
21             nb1 ← destLast − destBegin + 1 // # of elements to send to id1
22             if id1 > id then send F[i − 1, 0..nb1 − 1] to id1
23             if id2 < p then send F[i − 1, nb1..nb − 1] to id2
24         end
25     end
26     if id > 0 ∧ myLast − s[i] ≥ 0 then
27         id1 ← findID(myFirst − s[i] − 1, p, S)
28         if myFirst − s[i] < 0 then
29             myLocalBegin ← s[i] − myFirst
30         else
31             myLocalBegin ← 0
32         end
33         receive from id1 into L[myLocalBegin..nL − 1]
34         nS ← size of message received
35         id2 ← findID(myLast − s[i] − 1, p, S)
36         if id1 ≠ id2 ∧ id2 < id then receive from id2 into L[nS..nL − 1]
37     end
38     solveRow(F, L, s, nb, myFirst, i)
39 end
   // Trace F to return subset, or have task p − 1 print F[n, myLast] to
      return yes/no
```

Algorithm: 5.1 (corrected)

```
// assumes n divisible by p
shared a, b
start ← id * n/p
sum ← 0
for j ← 0 to n/p − 1 do
    sum ← sum + a[start + j]
    a[start + j] ← sum
end
b[id] ← a[(id + 1) * n/p − 1] // sum of all values in sub-array
scan(b)
if id > 0 then
    for j ← 0 to n/p − 1 do
        a[start + j] ← a[start + j] + b[id − 1]
    end
end
```

Algorithm: 6.5 (corrected)

Input: Graph with n vertices and edges with nonnegative weights W, source vertex s. Vertices and edges stored in the CSR representation with offset array E_O and edge array E.

Output: D: distance between s and all other vertices.

1 $\{D[v] \leftarrow \infty : i \in [0, n)\}$
2 $\{D[s] \leftarrow 0\}$
3 $\{F_1[0] \leftarrow s\}$
4 **while** $F_1 \neq \emptyset$ **do**
5 $\quad\{F_N[i] \leftarrow E_O[F_1[i]] : i \in [0, |F_1|)\}$
6 $\quad\{S[i] \leftarrow E_O[F_1[i] + 1] - E_O[F_1[i]] : i \in [0, |F_1|)\}$
7 $\quad S_O \leftarrow$ `scan(`exclusive, sum, S`)`
8 $\quad\{F_O[S_O[i] + j] \leftarrow E[F_N[i] + j] : i \in [0, |F_1|), j \in [0, S[i])\}$
9 $\quad\{W_O[S_O[i] + j] \leftarrow W[F_N[i] + j] : i \in [0, |F_1|), j \in [0, S[i])\}$
10 $\quad\{F_I[S_O[i] + j] \leftarrow F_1[i] : i \in [0, |F_1|), j \in [0, S[i])\}$
11 $\quad\{B \leftarrow$ `atomicMin(`$D[F_O[i]]$, $D[F_I[i]] + W_O[i]$`)` $i \in [0, |F_O|)\}$
12 $\quad\{T[i] \leftarrow 0 : i \in [0, n)\}$
13 $\quad\{B \leftarrow$ `CAS(`$T[F_O[i]]$, 0, 1`)` $: i \in [0, |F_O|) \mid B[i] = 1\}$
14 $\quad B_S \leftarrow$ `scan(`exclusive, sum, B`)`
15 $\quad\{F_2[B_S[i]] \leftarrow F_O[i] : i \in [0, |F_O|) \mid B[i] = 1\}$
16 $\quad$Swap references to F_1 and F_2
17 **end**