Mastermath midterm examination
Parallel Algorithms. Solution

Teacher: Rob H. Bisseling, Utrecht University

October 21, 2020

1. (a) The algorithm for 2-superstep matrix multiplication is given as Algorithm 1. The main idea is that the matrix element c_{ij} is computed by

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj},$$

where every processor computes the elements of c_{ij} for its own block in superstep (1). To do this, all elements from row i of A and column j of B are needed, and these are sent beforehand in superstep (0).

Algorithm 1 Matrix multiplication for processor $P(s,t)$

input: A, B: $n \times n$ matrices, distr(A) = distr(B) = square block.
output: C: $n \times n$ matrix, $C = AB$, distr(C) = square block.

```
b = n/M;                                          ▷ Superstep (0)
for i := sb to (s+1)b − 1 do
    for j := tb to (t+1)b − 1 do
        put a_ij in P(s, ∗);
        put b_ij in P(∗, t);
                                                  ▷ Superstep (1)
for i := sb to (s+1)b − 1 do
    for j := tb to (t+1)b − 1 do
        c_ij := 0;
        for k := 0 to n − 1 do
            c_ij := c_ij + a_ik b_kj;
```

(b) The BSP cost of the algorithm is

$$T_{\text{matmat}} = \frac{2n^3}{p} + \frac{2n^2}{p}(\sqrt{p}-1)g + 2l,$$

because every one of the $\frac{n^2}{p}$local elements is sent to $2M-2$ other processors in superstep (0), where all processors send and receive the same number of data words, so $h = \frac{2n^2}{p}(M-1)$. For every local element, $2n$ flops are performed in superstep (1).

(c) Every processor needs to store its own block of b^2 elements of the output matrix C but also b complete rows of A and b complete columns of B. Together this is $2bn + b^2 = \frac{2n^2}{\sqrt{p}} + \frac{n^2}{p}$ data words, which is an increase of about a factor of $\sqrt{p}$ compared to the input size. If not enough memory is available, more supersteps should be created by splitting the first superstep into sending parts of rows/columns (such as a $b \times b$ block of A and a $b \times b$ block of B) and by using these parts immediately in an update of a $b \times b$ block of C.

2. (a) The algorithm for vector reduction is given as Algorithm 2. The main idea is that processor $P(s)$ computes a block $x[sb : (s+1)b-1]$ of $\mathbf{x}$, where $b = k/p$ is the block size. To do this, it first needs the block $x_t[sb : (s+1)b-1]$ from all processors $P(t)$, which is sent in superstep (0). The computed block is broadcast to all processors in superstep (2). Note that this approach is similar to two-phase broadcasting: the responsibility for computing $\mathbf{x}$ is spread evenly over all the processors.

(b) The BSP cost of superstep (0) is $\frac{k(p-1)}{p}g + l$, because every processor sends k/p values to $p-1$ others, and it receives the same number of values. The cost of superstep (1) is $k + l$, because every processor adds k/p values received from each of the p processors. The BSP cost of superstep (2) is $\frac{k(p-1)}{p}g + l$, the same as for superstep (0). The total BSP cost of the algorithm is

$$T_{\text{reduction}} = k + \frac{2k(p-1)}{p}g + 3l \approx k + 2kg + 3l.$$

3. (a) A possible solution algorithm is the following. Every processor has a block of size $b = n/p$ of the vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$. To compute *digits* z_i, $sb \leq i < (s+1)b$, processor $P(s)$ first adds $z_i = x_i + y_i + c_i$, where c_i is the *carry* for index i. If this leads to $z_i \geq r$, we subtract r

Algorithm 2 Vector reduction for processor $P(s)$

input: $P(s)$ has vector $\mathbf{x}_s$ of length k.
output: $P(s)$ has vector $\mathbf{x}$ of length k, with $x[i] = \sum_{t=0}^{p-1} x_t[i]$, for $0 \leq i < k$.

```
b = k/p;                                          ▷ Superstep (0)
for t = 0 to p − 1 do
    for i := tb to (t + 1)b − 1 do
        put x_s[i] in P(t);
                                                  ▷ Superstep (1)
for i := sb to (s + 1)b − 1 do
    x[i] := 0;
    for t = 0 to p − 1 do
        x[i] = x[i] + x_t[i];
                                                  ▷ Superstep (2)
for t = 0 to p − 1 do
    for i := sb to (s + 1)b − 1 do
        put x[i] in P(t);
```

and define $c_{i+1} = 1$. Otherwise, z_i is already in the right format and we set $c_{i+1} = 0$. In the first instance, the processor disregards carries from other processors. Note that each carry is either 0 or 1, which can easily be proven by induction. This process is the initial computation superstep.

After that, the carry produced by the highest-indexed local digit $z_{(s+1)b-1}$ becomes the carry $c_{(s+1)b}$ to start with in $P(s+1)$, for $s < p-1$. This carry is sent from $P(s)$ to $P(s+1)$. Then the process of addition is performed again, but now just computing $z_i := z_i + c_i$, and only until a carry 0 is encountered. If for all local elements $z_i = r - 1$ holds (which for $r = 10$ is the *all-9s* case), this will result in another carry being sent. The communication superstep of sending these carries concludes round 0.

If any carries were sent, a new round is performed, consisting of first a computation superstep that propagates a carry locally, and then if needed a communication superstep that sends a carry to the next processor.

At the end of round 0, processor $P(0)$ is done, since it does not receive a carry itself. This also means that $P(1)$ will be done at the end of round 1. At the end of round k, processors $P(0), \ldots, P(k)$ will be done. Thus, in the worst case at the end of round $p-1$ all

processors will be done and the algorithm terminates.

(b) This worst case can indeed happen: an example with $r = 10$ is the very large integer $x = 4444\cdots445$ and $y = 4555\cdots555$ (in decimal notation with $x_0 = 5$), which first leads to $z = 8999\cdots9990$, and a carry $c_1 = 1$; the single carry then propagates through all blocks in p rounds. The cost for one round is $b + g + 2l$ since b additions are performed and at most one carry is sent or received per processor. The first round has an extra b flops, for adding $allx_i + y_i$. The cost of the worst case is therefore

$$T_{\text{worst}} = p(b + g + 2l) + b = n + n/p + pg + 2pl.$$

The worst case is in fact completely sequential, with only one processor working at the same time, in all rounds except round 0.

In the random case, there will hardly ever be more than two rounds. At the end of round 0, about half the processors will have a carry, depending on their highest digit $z_{(s+1)b-1}$. In round 1, the carries will propagate, but only in the all-9s case a new carry must be communicated, which is highly unlikely. So the cost of round 1 will probably be only a few additions. The expected cost will thus be about $2n/p + g + 4l$ for the expected two rounds.

A subtle point is that we need a termination mechanism to detect that all processors are done. We could incorporate this by letting each processor broadcast whether it is *active*, meaning that it sends a carry at the end of a round. If no processor is active, we are done. This mechanism costs an extra $(p-1)g$ per round. It is most likely invoked only once, in round 1. The total expected cost for the random case is therefore

$$T_{\text{random}} = 2n/p + (2p-1)g + 4l.$$

This will also be the expected cost in the average case.

Note that other solution algorithms may also be possible, with different worst-case and random-case behaviour. For instance, a solution with only three supersteps can be obtained based on a broadcast to all processors of a pair of data, $active_s$ and $propagate_s$, where $propagate_s$ means that processor $P(s)$ will propagate a carry all the way (meaning all its components are $r-1$). This information enables every processor to complete the computation in the computation superstep of round 1.