Solutions Exercise 1.2 from PSC2

(a) *Minimum finding: determine the index j of the component with the minimum value and subtract this value from every component: $y_i = x_i - x_j$, for all i.*

The basic idea is to find the local minimum first, send it to all processors, determine the global minimum redundantly, and then subtract it from all local values.

---

input: $\mathbf{x}$ vector of length n, $\mathrm{distr}(\mathbf{x}) = $ block.
output: $\mathbf{y}$ vector of length n, $\mathrm{distr}(\mathbf{y}) = $ block,
$j = \mathrm{argmin}\{x_i : 0 \le i < n\}$, $y_i = x_i - x_j$, for all i.

```
b = ⌈n/p⌉;                                        ▷ Superstep (0)
minval_s := ∞;
for i := sb to min((s+1)b, n) − 1 do
    if x_i < minval_s then
        j_s := i;
        minval_s := x_i;

for t := 0 to p − 1 do                            ▷ Superstep (1)
    put j_s, minval_s in P(t);

minval := ∞;                                      ▷ Superstep (2)
for t = 0 to p − 1 do
    if minval_t < minval then
        j := j_t;
        minval := minval_t;
for i := sb to min((s+1)b, n) − 1 do
    y_i := x_i − minval;
```

The BSP cost of the algorithm is

$$2\lceil n/p \rceil + p + 2(p-1)g + 3l,$$

where we counted 1 flop for a comparison, and no flops for an assignment.

(b) *Rotating to the right: assign $y_{(i+k) \bmod n} = x_i$.*

The basic idea is to put the data into the correct destination processor in a single communication superstep. We may assume without loss of generality that $k \le n/2$, because otherwise we have a rotation of $n-k$ to the left, which is similar.

We compute the BSP cost as follows. For $k < n/p$, every processor $P(s)$ puts its k rightmost data into processor $P((s+1)) \bmod p)$, at a cost of $kg + l$. The remaining $n/p - k$ local data are shifted to the right in a local memory copy. For $k \ge n/p$, the cost is $(n/p)g + l$.

(c) *Smoothing: replace each component by a moving average* $y_i = 1/(k+1) \sum_{j=i-k/2}^{i+k/2} x_j$*, where* k *is even. Assume here that* $x_j = 0$ *for* $j < 0$ *or* $j \geq n$.

Assume for simplicity that $k/2 \leq n/p$ (a commonly used value these days is $k = 6$). The remaining case is similar, but a bit more elaborate. The rightmost vector component of a processor $P(s)$ has to obtain at most $k/2$ values from $P(s+1)$. The leftmost vector component has to obtain at most $k/2$ values from $P(s-1)$. With this information, all local components y_i can be computed.

The BSP cost of the algorithm is

$$3\lceil n/p \rceil + k + kg + 2l.$$

Here, we first obtain all locally needed data in kg time. After that, we need k flops to add the first k values, and then we produce a moving average by subtracting the left value and adding the right value of the moving window in $2\lceil n/p \rceil$ time. Finally, we divide all local data by $k+1$ in $\lceil n/p \rceil$ time.

Note that if you perform a cost analysis, this forces you to be precise about what happens in the algorithm. Still, you do not always have to give the full details as in (a).

(d) *Partial summing: compute* $y_i = \sum_{j=0}^{i} x_j$*, for all* i.

The basic idea is that every processor $P(s)$ computes its local partial sums, without regard for the others. $P(0)$ will then already have the correct result, but the others will have to add a correction, which consists of the total sums for all processors $P(t)$ with $t < s$. This can be done by processor $P(t)$ sending its rightmost value to all higher-numbered processors. The received sums are then added locally. In a final pass through the local data, the correction is added.

The BSP cost of the algorithm is (please check)

$$2\lceil n/p \rceil + p - 1 + (p-1)g + 3l.$$

In an implementation, it would be useful to write a separate sequential function to compute partial sums, and use this function in the parallel program. Reuse of code!