E1 Reduction I

The summation of $N$ numbers can be computed with the following loop.

\[
\text{for } i \leftarrow 0 \text{ to } N-1 \text{ do} \\
\quad s \leftarrow s + a(i) \\
\text{od}
\]

where $s$ denotes a scalar value and $a(\cdot)$ an array of size $N$. As the scalar value $s$ is needed in each step of the iteration, the loop cannot be vectorised (i.e. instruction pipelining) in that way. How might the problem of the so-called reduction loop be solved in order to exploit the performance of the underlying hardware?

E2 Reduction II

Implement (in pseudo code) the above reduction problem, i.e. the summation of all elements of a vector, using the PRAM model and think about any access conflicts concerning your shared registers. What is the parallel time complexity of your algorithm?

E3 Speedup and Parallel Efficiency

Given is a matrix $A$ of size $N \times N$. Some iterative smoothing algorithm (1) calculates the mean value of each coefficient $a_{ij}$ and its four (direct) neighbours before the new value $\hat{a}_{ij}$ is stored at position $(i, j)$ again – such a smoothing step takes $T_S$ time. For a parallel approach with $p$ processing elements, the matrix $A$ shall be divided into $p$ parts, thus, each part consists of $N/p$ columns and $N$ rows. For calculating the mean at the border of each part one processing element has to exchange its border values with its direct neighbours – a single exchange of one value $a_{ij}$ takes $T_E = 2 \cdot T_S$ time.

Calculate the maximum number $p$ of processing elements, thus, the parallel efficiency $E$ is at least 50%.

\[
\hat{a}_{ij} = \frac{1}{5} \cdot (a_{ij} + a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1}) \quad (1)
\]