FDA125 Advanced Parallel Programming March 2007

# Exercises for Topic IV (Parallel Algorithmic Paradigms)

**Exercise IV.1**

Write a pipelined sum program in Fork that uses a linear chain of $p$ processors as pipeline graph, as shown in the Figure below. Each step of the pipeline starts the processing of a slice of $p$ subsequent input elements $a_0, ..., a_{p-1}$. Processor 0 adds $a_0$ and $a_1$ and stores the result in its `data` field. Each processor $j$, $0 < j < p-1$ adds an input value $a_{j+1}$ and the `data` value computed by its predecessor in the previous step and writes the sum to its `data` entry. The last processor accumulates the global sum in its `data` entry by repeatedly adding the `data` value computed by its predecessor in the previous step. The result is available in the `data` field of the last processor after $\lceil n/p \rceil + p$ steps. Note that the structure of sum computation implied by this algorithm does not follow the order of the elements in the input sequence, that is, it assumes commutativity of addition.

**Exercise IV.2**

Consider the parallel `treeforce` implementation of the lecture for the case $p > N$, in particular $p \approx 8N$. For the case $\theta \approx 1$, is the synchronous exploitation of nested parallelism really a good idea? Develop an alternative way of exploiting nested parallelism that may lead to better load balancing.

(Hint: use asynchronous mode and one parallel task queue per particle, that is, per processor subgroup created by the topmost `fork` statement. Create tasks in `treeforce` as in the asynchronous N-Queens example, see PPP 7.6.)

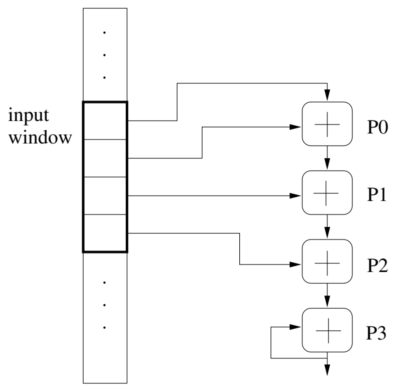


Figure 1: A linear sum pipeline for 4 processors