Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Prof. L. Thiele

Hardware/Software Codesign - HS 15

Exercise Sheet 4

Issue Date: 14 October 2015
Discussion Date: 21 October 2015

4.1 Closeness Function

The nodes of the data flow graph shown in Fig. 1 represent arithmetic operations. The edges are labeled with the bit widths of the required data types. The following closeness function is given:

$$\begin{aligned} Closeness(o_i, o_j) &= \left(\frac{Conn(o_i, o_j)}{TotalConn(o_i, o_j)}\right) + \\ &+ \left(\frac{FU_{cost}(o_i) + FU_{cost}(o_j) - FUGROUP_{cost}(o_i, o_j)}{FUGROUP_{cost}(o_i, o_j)}\right) \end{aligned}$$

The different terms in this function have the following meanings:

- $Conn(o_i, o_j)$: the number of common wires between objects o_i and o_j.
- $TotalConn(o_i, o_j)$: the sum of all wires that connect to objects o_i and o_j. Common wires are counted twice - once for each object.
- $FU_{cost}(o_i)$: the cost of the functional unit that implements o_i.
- $FUGROUP_{cost}(o_i, o_j)$: the cost of the minimal number of functional units that are required to execute objects o_i and o_j. (For example, should both o_i and o_j be additions, they would be assigned to one functional unit of type adder.)

4.1.a) Physical interpretation

Find a physical interpretation of the closeness function.

4.1.b) Hierarchical clustering

- Compute the closeness values for all pairs of objects in Fig. 1 according to the given closeness function. The cost of an adder and the cost of a subtractor are 1.
- Define a suitable closeness function for clustering hierarchical objects based on the function given above. Cluster the graph until you receive a bi-partitioning.

4.2 HW/SW partitioning

Fig. 2 shows the pseudo code of a greedy algorithm for HW/SW partitioning. The algorithm starts with a partition where all objects are realized in hardware. Then, objects are migrated to software as long as the performance requirement is satisfied (function SatisfiesPerformance) and the cost of the new partitioning is lower (function f). If an object is migrated, the algorithm also tries to migrate all successor nodes (function Successors).

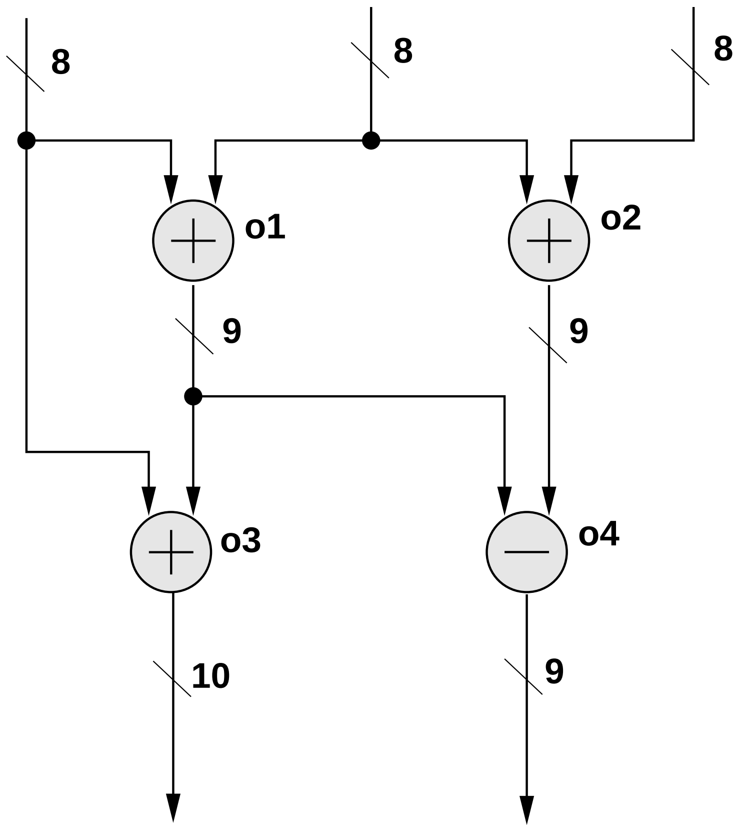

Figure 1: Data flow graph

- Apply the algorithm to the sequence graph shown in Fig. 3. The function SatisfiesPerformance(P) should return `TRUE` if P satisfies the latency bound $\bar{L} \leq 7$. To determine the latency of a partitioning, you have to construct a valid schedule. The execution times of start- and end nodes of the sequence graph are 0, all other node execution times are given in Fig. 3, split into HW (d_{HW}) and SW (d_{SW}). For a communication between HW and SW, a delay of 0.5 per edge has to be accounted for. For HW nodes there are no resource constraints, i.e., all ready nodes can be executed in parallel. The SW nodes have to share one processor. The function f determines the cost. For a SW node the cost is 0, and for a HW node the cost is 1.

Note again that the solution is given when the main procedure processes elements in increasing order of their indexes. However, if we do not stick to this assumption, other nodes may also be added to the software partition. For example, node 10 can also be moved to software implementation.

```
$P = \{\{\}, O\}$; /* all in HW */

PROCEDURE PARTITIONING
    REPEAT
        $P_{old}$=$P$;
        FORALL $o_i \in HW$
            AttemptMove($P, o_i$);
        ENDFOR
    UNTIL ($P == P_{old}$)
END PROCEDURE

PROCEDURE AttemptMove($P, o_x$)
    IF SatifiesPerformance(Move($P, o_x$)) AND
      (f(Move($P, o_x$)) < f(P))
            $P$ = Move($P, o_x$);
            FORALL ($o_y \in$ Sucessors($o_x$))
                AttemptMove($P, o_y$) ;
            ENDFOR
    ENDIF
END PROCEDURE
```

Figure 2: Pseudo code for a greedy HW/SW partitioner

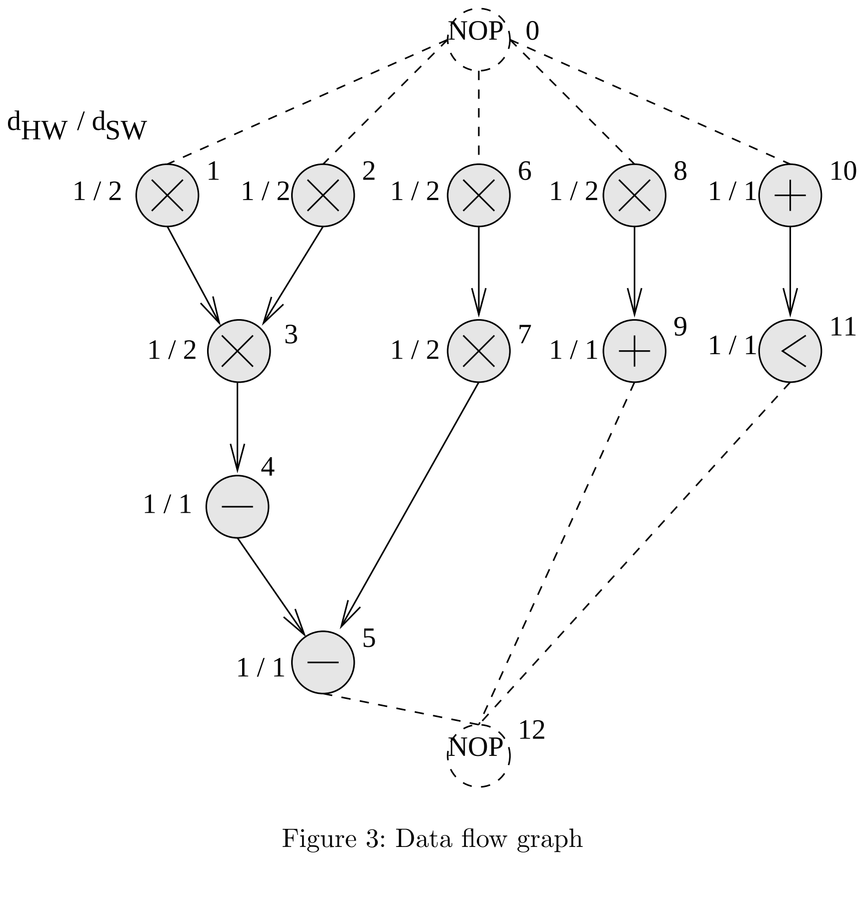

Figure 3: Data flow graph