SODS HLS CONTEST 2020-2021

GROUP 06:
Luca Dalmasso 281316
Gianmarco Dogliani s290146
Alessandro Landra s284939

Description of the algorithm:
The starting point of the overall algorithm is a revisited version of the list scheduling (resource-constrained) algorithm. The original list algorithm picks up the nodes to be scheduled through the *get_sorted_nodes* command, which only ensures a topological order among the nodes. Our revisited version instead, has an additional feature: it prioritizes the scheduling of the nodes belonging to the critical path. In this way, always respecting the topology of the graph, we are able to put more effort into the scheduling of the nodes which are more distant from the sink.
The critical path has been identified as the longest path, in terms of number of edges, among all paths starting from the sink nodes and ending at one of the possible root nodes. In order to carry out this computation, the graph has been divided into levels, and the path whose root node is characterized by the highest level identifies the critical path.

Once this scheduling operation has been performed, a first scheduled DFG is available. Notice that parallelism has not been considered yet; in other words, only one resource for each type, the smallest, is available.

In order to decide which resources to schedule in parallel, the scheduled DFG has been considered again to be sliced into levels. Two nodes belong to the same level if they have the same distance, in terms of number of edges, from the sink node. According to us, two resources of the same type belonging to the same level are more likely to be executed in parallel.
For every level, the number of resources of the same type belonging to that level has been annotated. Then, for every resource, the overall maximum has been taken. In this way, we could have an idea of the maximum amount of parallelism possible.
Given this, the schedule has been updated adding, for each resource, one resource at a time, up to the maximum amount of them that could have been executed in parallel.
For this purpose, the available resources have been ordered in decreasing-latency order and the slowest resource has been optimized for first.
Specifically, this operation iterates on the resources and on their various implementations, starting from the slowest up to the fastest version.

Clearly, the procedure works as long as the area constraint is met.

The following page reports a series of puts extracted from the execution of the second phase of the algorithm, it might help the understanding of the parallelization procedure.

```
params: {L6 1 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
FIRST RUN (WORST CASE LATENCY): 145, AREA(the minimum one): 70
params: {L6 1 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 145, AREA: 70
params: {L6 2 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 80, AREA: 110
params: {L6 3 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 80, AREA: 150
params: {L6 4 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 80, AREA: 190
params: {L5 1 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 75, AREA: 100
params: {L5 2 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 75, AREA: 170
params: {L5 3 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 75, AREA: 240
params: {L5 4 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 75, AREA: 310
params: {L5 4 MUL} {L2 1 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 75, AREA: 310
params: {L5 4 MUL} {L2 2 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 45, AREA: 320
params: {L5 4 MUL} {L2 3 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 35, AREA: 330
params: {L5 4 MUL} {L2 4 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 35, AREA: 340
params: {L5 4 MUL} {L2 4 ADD} {L12 1 LOD} {L15 1 STR}
Latency: 35, AREA: 340
params: {L5 4 MUL} {L2 4 ADD} {L12 2 LOD} {L15 1 STR}
Latency: 35, AREA: 350
params: {L5 4 MUL} {L2 4 ADD} {L12 2 LOD} {L15 1 STR}
Latency: 35, AREA: 350
params: {L5 4 MUL} {L2 4 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 30, AREA: 360
params: {L4 1 MUL} {L2 4 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 36, AREA: 180
params: {L4 2 MUL} {L2 4 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 27, AREA: 280
params: {L4 3 MUL} {L2 4 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 27, AREA: 380
params: {L4 4 MUL} {L2 4 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 27, AREA: 480
params: {L4 4 MUL} {L1 1 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 32, AREA: 460
params: {L4 4 MUL} {L1 2 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 22, AREA: 480
params: {L4 4 MUL} {L1 3 ADD} {L12 2 LOD} {L15 2 STR}
Latency: 20, AREA: 500
params: {L4 4 MUL} {L1 3 ADD} {L11 1 LOD} {L15 2 STR}
Latency: 17, AREA: 500
params: {L4 4 MUL} {L1 3 ADD} {L11 1 LOD} {L14 1 STR}
Latency: 14, AREA: 500
params: {L4 4 MUL} {L0 1 ADD} {L11 1 LOD} {L14 1 STR}
Latency: 17, AREA: 480
params: {L4 4 MUL} {L0 1 ADD} {L10 1 LOD} {L14 1 STR}
Latency: 17, AREA: 500
```

Figure: series of puts taken from the execution of the parallelization steps