IL2212 EMBEDDED SOFTWARE

Practical Homework 3

VERSION 1.0

The total amount of points in this homework is 8 points + 2 bonus points. To pass the practical homework, 17 points out of 24 points in all three homework examinations are required.

1. (1 POINT) A succesful implementation of the C-program of Task 2 in the theoretical homework will give 1 point for the practical homework.

2. (1 POINT + 1 BONUS POINT) A succesful implementation of the C-program of Task 3 in the theoretical homework will give 1 point and 1 bonus point for the practical homework.

3. (3 POINTS)

   Introduction ForSyDe supports the modelling of data parallelism by providing the data type `Vector` and higher-order functions, called *skeletons*, which operate on the `Vector` data type. Skeletons work in the same way as *process constructors*, but they model data parallelism, while process constructors are used to model processes belonging to a concrete model of computation.

   The combination of using both process constructors and skeletons becomes very powerful, since it allows to model parallelism. Furthermore, it enables design transformation, which allows to convert a process that includes potential data parallelism into a parallel process network with the same function.

   An example can illustrate this. The process `p_1` operates on vectors, which provide potential data-parallelism.

```haskell
p_1 :: Signal (Vector Integer) -> Signal (Vector Integer)
p_1 = mapSY (mapV f)
```

   The following code shall illustrate the usage of signals and vectors.

```
ghci> v1 = vector [0..3]
ghci> v2 = vector [4..7]
ghci> sv = signal [v1, v2]
ghci> f x = x * 2
ghci> mapV f v1
<0,2,4,6>
ghci> p_1 = mapSY (mapV f)
```

```
ghci> sv
{<0,1,2,3>,<4,5,6,7>}
ghci> p_1 sv
{<0,2,4,6>,<8,10,12,14>}
```

However, in the present form , the process `p_1` will be implemented as sequential code running on one processor, since the ForSyDe compilation rules treat a process as a single entity during synthesis and will not divide a process further. The process `p_1` is illustrated in Figure 1.

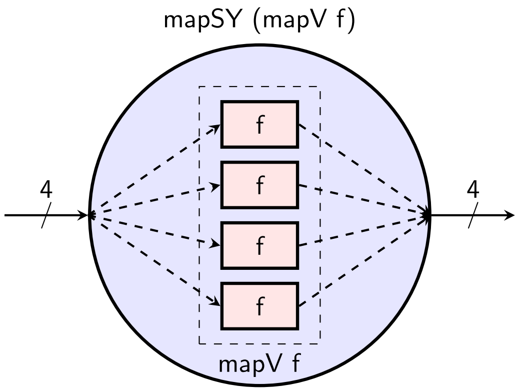

Figure 1: Process network for process `p_1` using vector size 4

Instead, the identical functionality can also be expressed using a vector of parallel processes. The process `p_1'` is functional identical to `p_1`, but moves the parallelism to the process level.

```
p_1' :: Signal (Vector Integer) -> Signal (Vector Integer)
p_1' = zipxSY . mapV (mapSY f) . unzipxSY
```

Here, the function `unzipxSY` converts a signal of vectors to a vector of signals. Thus, it splits a signal into several signals. Then the skeleton `mapV` maps the process `mapSY f` on each element of the vector, i.e. on each individual signal. Then the function `zipxSY` converts a vector of signals into a signal of vectors, which merges several signals into a single signal. The operator `(.)` is the *function composition* operator that takes two functions and creates a new function. It has the type `(.) :: (b -> c) -> (a -> b) -> a -> c`.

The process `p_1'` is illustrated in Figure 2.

Using the example function `f x = x * 2`, we can see that `p_1` and `p_1'` has the same functionality as `p_1`. The difference is that `p_1'` can be implemented on several processors, since it uses a vector of processes `mapSY f`.

```
ghci> p_1 sv_1 == p_1' sv_1
True
```

NOTE: There is an example program on the course web page, which should be used as a starting point to get a better understanding of how to use skeletons to model systems with exlicit parallelism. This example program also introduces `zipWithV`, `reduceV` and `groupV`. Modify the code to get a better understanding of the skeletons. More information can be found in the online-documentation of **`ForSyDe.Shallow`** on Hackage.

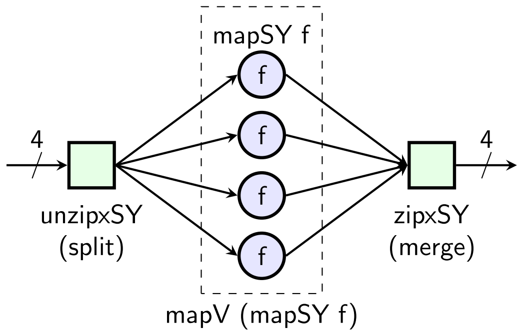

Figure 2: Process network for process `p_1'` using vector size 4

https://hackage.haskell.org/package/forsyde-shallow-3.4.0.1/docs/ForSyDe-Shallow-Core-Vector.html

Exercise Task Given are the following two process networks:

```
p_2 :: Signal (Vector Integer) -> Signal Integer
p_2 = mapSY (reduceV g . mapV f)

p_2' :: Signal (Vector Integer) -> Signal Integer
p_2' = mapSY (reduceV g) . zipxSY . mapV (mapSY (reduceV g))
        . mapV (mapSY (mapV f)) . unzipxSY . mapSY (groupV 4)
```

(a) Illustrate the process network `p_2` using the notation of Figure 1 and Figure 2. Give a short explanation.

(b) Illustrate the process network `p_2'` using the notation of Figure 1 and Figure 2. Give a short explanation.

(c) Assume you have a multiprocessor with four processors. Is it always beneficial to exploit all four processors according to the program `p_2'`? Motivate!

4. (3 Points + 1 Bonus Point) This exercise is related to the system model of an image processing application, which is available on

https://gits-15.sys.kth.se/ugeorge/il2212-lab/tree/master/model/src/IL2212.

(a) Illustrate the functions in the module **`Utilities`** graphically and give a short explanation about their functionality and their potential for parallelisation.

i. `mapMatrix`

ii. `reduceMatrix`

(b) Illustrate the functions in the the model **`ImageProcessing`** graphically and give a short explanation about their functionality and their potential for parallelisation. You can use the functions `mapMatrix` and `reduceMatrix` as black boxes in this task.

i. `grayscale`

ii. `resize`