Programming computers with SkIml

Quentin Carbonneaux

INRIA

November 23, 2017

Table of contents

Table of contents

Introduction

Sklml simple skeletons

Data parallel skeletons

Instruction parallel skeletons

Control skeletons

Full example

Compiling and running programs

Compiling

Running

Several kinds of parallelism

In several cases, parallelism can be expressed with high level schemes:

- several data, one action (data parallelism);
- several actions, several data (instruction parallelism).

The kind of data and actions on these data can also be a source of parallelism:

- product parallelism: data are (α, β) pairs, actions are pairs of functions (f_α, f_β);
- sum parallelism: data are of two kinds α or β, and actions are specific f_α or f_β.

SkIml goals

The SkIml framework addresses the following problems:

- provide a uniform framework to express common forms of parallelism;
- provide a toolkit to compile and run programs in a fast and simple fashion;
- let the programmer quickly prototype the program to identify the bottlenecks;
- stay functional friendly (type safe, compositional);
- behave the same in parallel and sequential modes.

Sklml skeleton kinds

As described above, Sklml provides three kinds of skeletons to express parallelism:

- data parallel skeletons;
- instruction parallel skeletons;
- control skeletons.

Complex skeletons are built by composing skeletons. A basic domain decomposition skeleton and a `if-then-else` skeleton has been written using this technique. Those composite skeletons are available in the library `sklml_extra`

The farm skeleton

The `farm` skeleton performs the same action on a set of data.

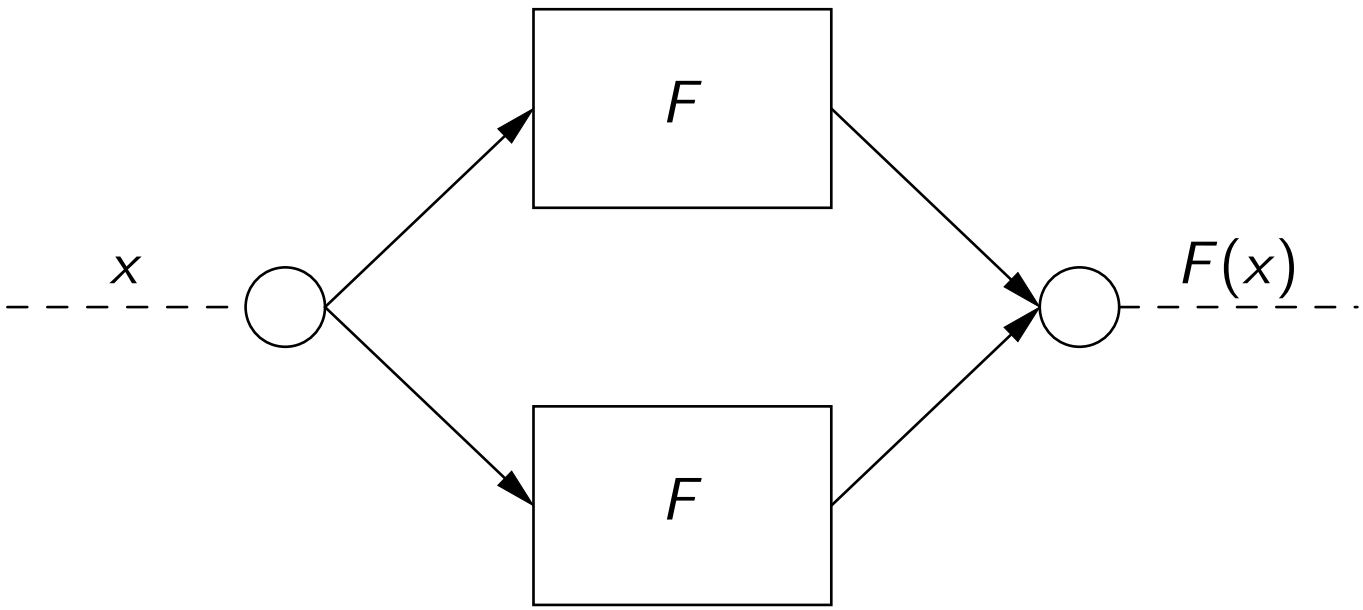

Figure: `farm` F skeleton graph

The product skeleton

The ***, or product, skeleton applies a pair of functions to a pair of values in parallel.

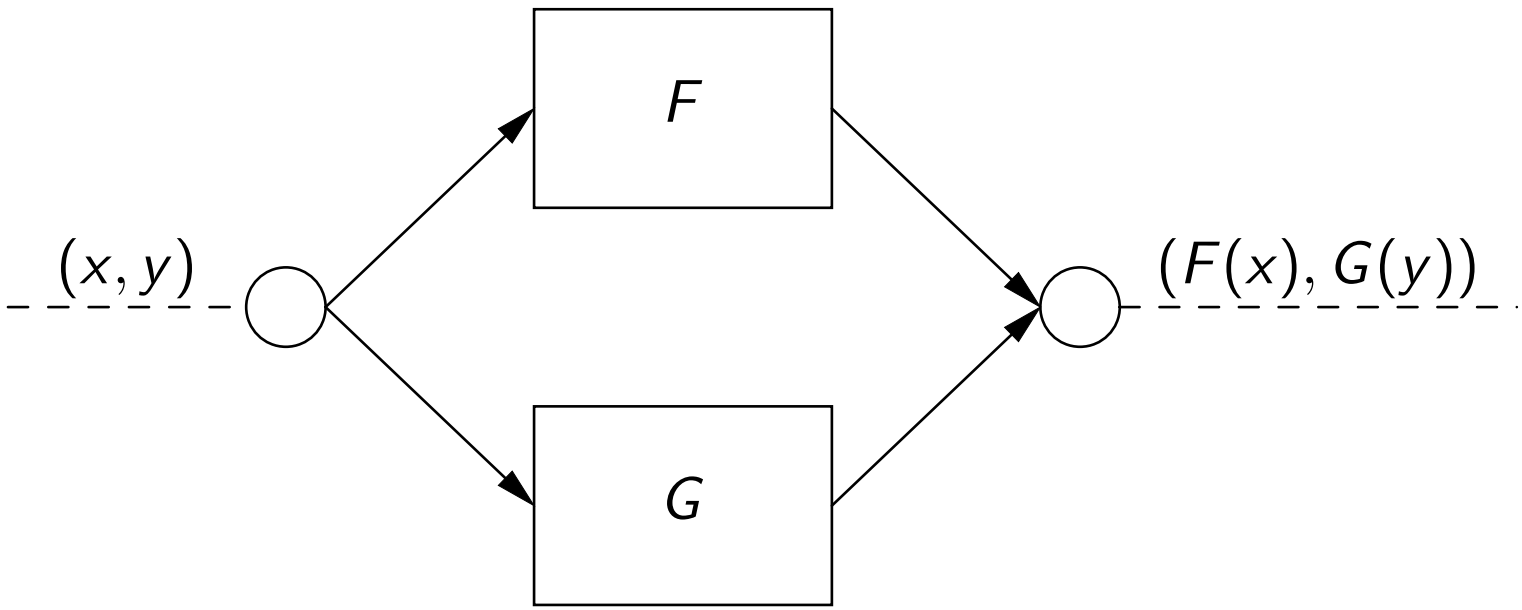

Figure: F *** G skeleton graph

The pipe skeleton

The |||, or pipe, skeleton is the simplest instruction parallel skeleton: it implements the parallel composition of functions.

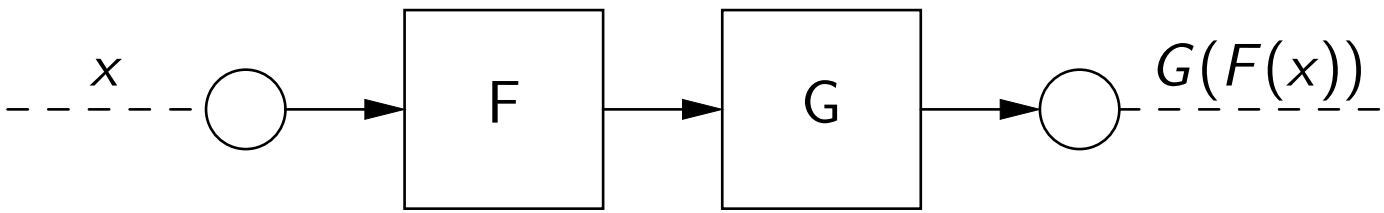

Figure: The F ||| G skeleton graph

The loop skeleton

The `loop` skeleton is a control skeleton: it computes the fixpoint of a function (composing a function until some predicate becomes `false`).

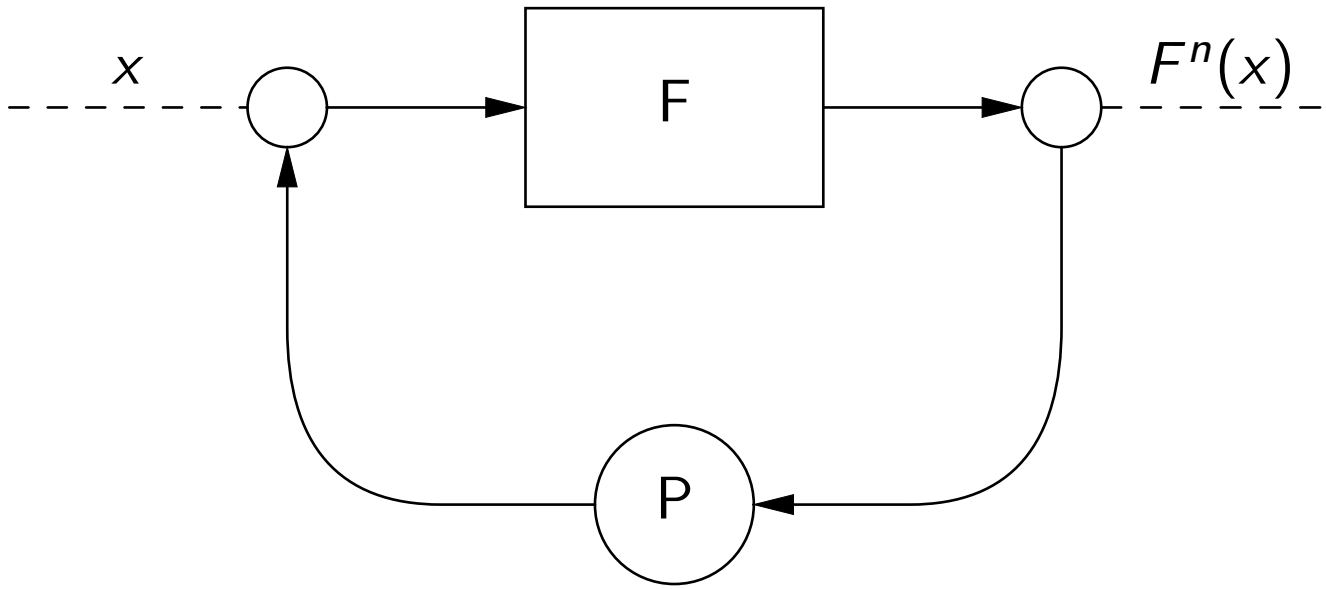

Figure: `loop` (P, F) skeleton graph

Simple skeleton example

Let's define the skeleton computing the function $(x, y) \mapsto H(F(x), G(y))$.

Simple skeleton example

Let's define the skeleton computing the function $(x, y) \mapsto H(F(x), G(y))$.

```
let sk = (F *** G) ||| H;;
```

Compiling with `sklmlc`

Sklml provides a compiler to compile programs either in parallel or sequential mode. This compiler wraps the relevant options to ultimately call the OCaml compiler.
The `-mode` option specifies the desired compiling mode.

```
$ sklmlc -mode seq -o hello.out hello.ml
```

Running Sklml programs

Running sequential programs is as simple as running any program:

```
$ ./hello.out
```

Sklml provides the `sklmlrun` helper to run parallel programs:

```
$ sklmlrun ./hello.out
```