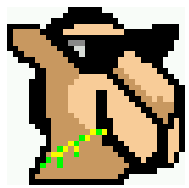
Parallel programming with Skiml

Quentin Carbonneaux François Clément Pierre Weis

MaGiX@LiX - September 22nd, 2011



Industry standards

The problems which led to the design of Sklml

OpenMP

- It is often used to parallelize sequentially thought code;
- it was designed for shared memory architectures.

MPI

- It is the new parallelism assembly, lets you do everything, possibly dangerous but fast;
- the code is a mixture of sequential instructions and parallel functions.

What Sklml is

Sklml is:

- high level and type safe;
- not bound to shared memory systems (works on clusters);
- written in OCaml;
- a complete toolkit (compiler + library + runtime system);
- a way to separate parallel and sequential logics.

What Sklml is not

Sklml is not:

- a way to encode every parallel scheme;
- the fastest parallel toolkit;
- ...

Sklml skeletons

What is a skeleton

A skeleton value is an OCaml value with type `(`α`, `β`) skel`, its input is of type α, its output of type β.

The Sklml library provides skeletal combinators which might either

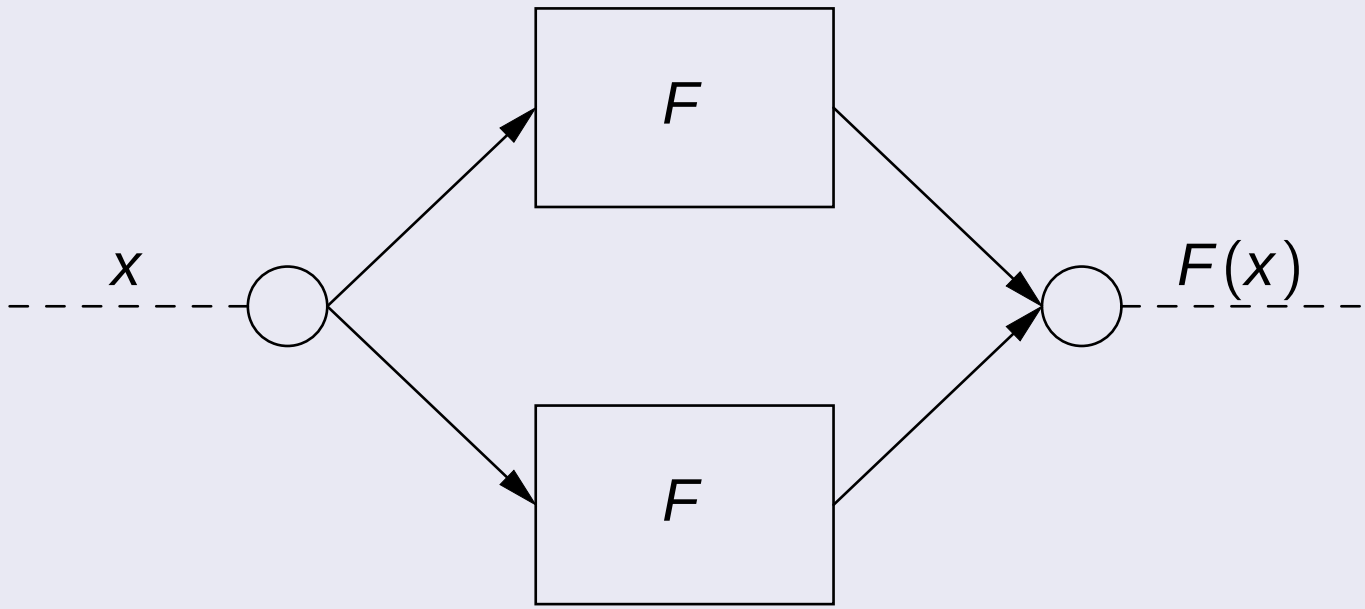
- encode some kind of parallelism (data parallelism, program parallelism);
- encode some kind of control structure (`if-then-else`, `do-while`,...).

Sklml skeletons

The farm skelton combinator

The farm skeleton combinator is used when one treatment must be applied in parallel to a flow of data.

`val farm : (`α`, `β`) skel * int` $\rightarrow$ `(`α`, `β`) skel;;`



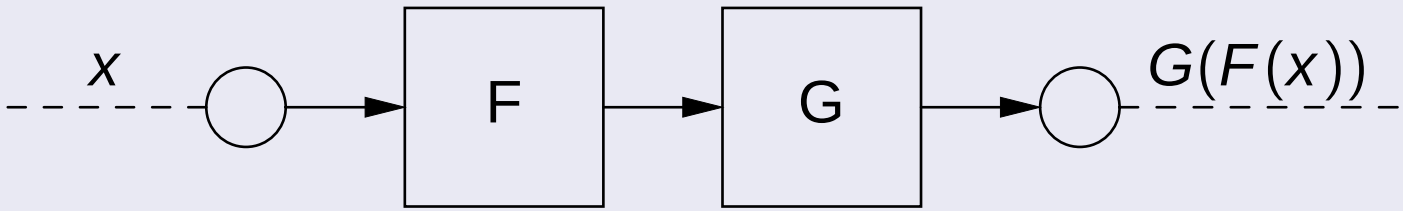
`farm` $(F, 2)$ skeleton graph

Sklml skeletons

The pipeline skeleton combinator

The pipeline skeleton combinator is used when a composition of two functions must be applied to a flow of data.

```
val ( ||| ) :
  (α, β) skel → (β, γ) skel → (α, γ) skel;;
```



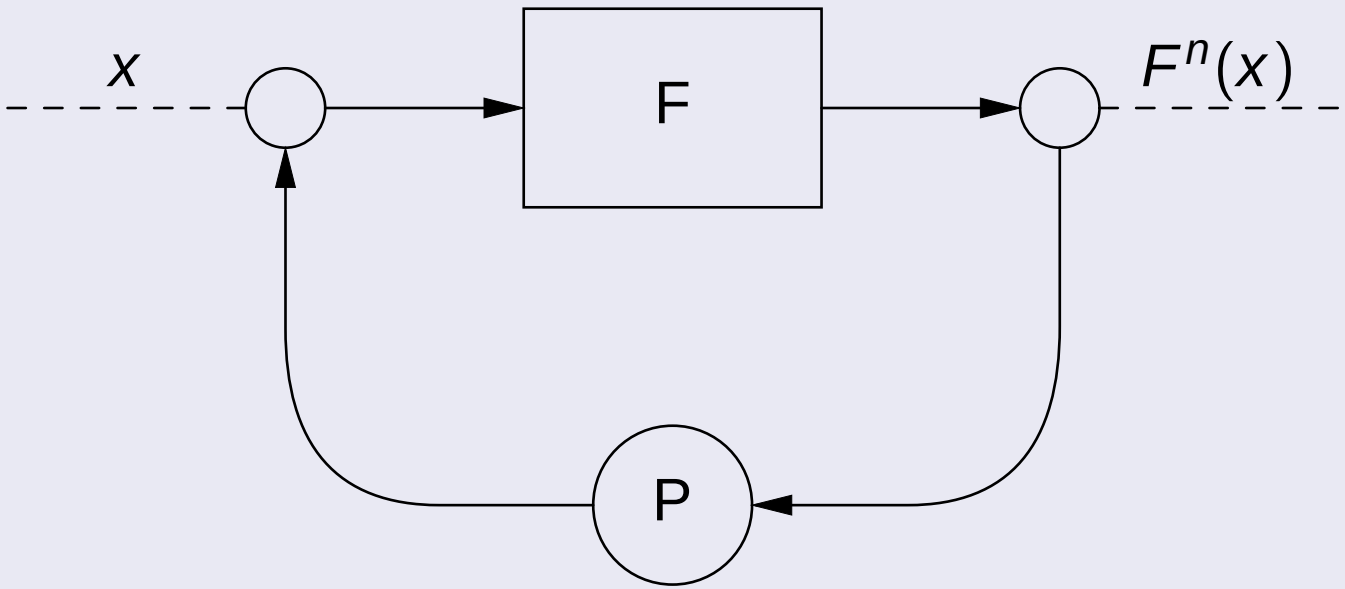
G ||| *F* skeleton graph

Sklml skeletons

The loop skeleton combinator

The loop skeleton combinator is a control combinator, it iterates a skeleton on a data until a given predicate turns to be false.

```
val loop :
  (α, bool) skel * (α, α) skel → (α, α) skel;;
```



loop (P, F) skeleton graph

Simple example
Introducing the example

Problem

Find the first element which does not satisify a given property P.
Testing this property is expensive and must be done in parallel.
Two functions are provided:

- `next_elm` which gives the "successor" of its input;
- `test_elm` a predicate function which test if an element satisfies the property P.

This problem is taken from the test `PrimeGen`, a generator of primes satisfying strong cryptographic properties. This test can be found in the standard Sklml distribution.

Simple example
The actual Sklml code

In Sklml:

```
let find_skl nw =
  loop ( farm (test_elm, nw) ||| fold_or,
         next_elms ) in
  ...
```

Simple example
The actual Sklml code

In Sklml:

```
let find_skl nw =
  loop ( farm (test_elm, nw) ||| fold_or,
         next_elms ) in
  ...
```

In C:

```
do {
    elm = next_elm(elm);
} while (test_elm(elm) == True);
```

Sklml and the other languages

Sequential parts of Sklml programs can be written:

- in pure OCaml;
- in C, the standard OCaml extension system can be used;
- with any other language, use external worker communicating through a IO library (c.f. PIO).

Old code can be reused with Sklml!

Sklml distribution

Sklml is a set of 4 components written both in OCaml and Sklml:

- a compiler (`sklmlc`);
- a core library;
- an extra library;
- a parallel process manager (`sklrun`).

Sklml's key feature (1)

Fact

Skeletal combinators have simple sequential semantics.

As a consequence, two compilation modes are proposed, a sequential interpretation of skeletal combinators and a parallel one.

Sklml's key feature (1)

Fact

Skeletal combinators have simple sequential semantics.

As a consequence, two compilation modes are proposed, a sequential interpretation of skeletal combinators and a parallel one.

Two semantics in practice

Compile either in parallel mode:

```
sklmlc -mode par code.ml
```

Or in sequential mode:

```
sklmlc -mode seq code.ml
```

Sklml's key feature (2)

Moreover, the Sklml system guaranty that:

- parallel and sequential programs will give exactly the same results;
- if your code run in sequential mode, it is guaranteed to work in parallel mode.

Thus:

1. develop and debug using the sequential semantics and compilation;
2. start the heavy thing by simply changing a flag in the makefile.

Sklml and OCaml 3.12

Sklml, because of its high abstraction, uses not so regular features of the OCaml language.

- First class modules to emulate GADTs (3.12).
- Lazy evaluation to represent possibly infinite computations.
- Second rank polymorphism to provide a really polymorphic API.
- Polymorphic recursion to handle uniformly skeletons (3.12).

Things to do

Sklml is already usable but it can be improved:

- improve the load balancing system;
- handle failures;
- improve error messages;
- add skeletons (?);
- tell people they must use it!

That's all folks!

- Any questions?
- Want to see some code?