PySke: Algorthmic Skeletons for Python

Jolan Philippe

Northern Arizona University
IMT Atlantique

September 11, 2019

PySke

PySke = Python + Skeletons
Aims ta easing parallelism

PySke: Python

Why Python? Because Python is cool! (But pythons are not).

- langage orienté objet simple à utiliser et à mettre en oeuvre
- Lambda calcul easy to use
- Typage dynamique, pas de declaration de variable (Smiley interrogatif) (typage possible depuis 3.7 mais pour cb de temps?)

Very used language in the programmer community

PySke: Python

Community graph 1 (Stack overflow researches)

PySke: Python

Community graph 2 (Google trend)

PySke: Skeleton

Definition (Murray Cole) SPMD vs Global view Inspiration Fonctionnelle
Pattern de calcul. PySke : Sur des structures de donnees Object (notation pointee) + functional (return of a new structure) Style

MURRAY COLE. Algorithmic Skeletons: Structured Management of Parallel Com- putation. In *MIT Press* 1989.

PySke: Skeleton

Title : PySke 1.0
Papiers: These + HPCS 2019

Jolan Philippe and Frédéric Loulergue. PySke: Algorithmic skeletons for Python. In *International Conference on High Performance Computing and Simulation (HPCS)*. IEEE, 2019.

Jolan Philippe. Systematic Development of Efficient Programs on Parallel Data Structures. Master Thesis.

PySke: Skeleton

Etat de l'art (tableau) TARGET OF THE LANGUAGE: Lists + Trees

PySke: Functionnal inspiration

Definition of lists (functional inspiration and Bird-Meertens Formalism).

$$\begin{array}{rl} \text{List A}: & \mid \textbf{nil}: \text{List A} \\ & \mid \textbf{cons}: \text{A} \to \text{List A} \to \text{List A} \end{array}$$

Richard S. Bird. An introduction to the theory of lists. In *Logic of Programming and Calculi of Discrete Design. NATO ASI Series F*

PySke: Inspiration

Definition of lists (functional inspiration and Bird-Meertens Formalism).

$$\begin{aligned} \text{List A} : \quad & | \ \textbf{nil}\text{: List A} \\ & | \ \textbf{cons}\text{: A} \to \text{List A} \to \text{List A} \end{aligned}$$

Notations:

- $[\,] = \textbf{nil}$
- $h :: t = \textbf{cons}\ h\ t$
- $a :: b :: c :: [\,] = [a; b; c]$

Richard S. Bird. An introduction to the theory of lists. In *Logic of Programming and Calculi of Discrete Design. NATO ASI Series F*

PySke: Inspiration

Map: applies a function to every element of the structure.

$$map : (A \to B) \to List\ A \to List\ B$$
$$map\ f\ [x_1; ...; x_n] = [f\ x_1; ...; f\ x_n]$$

PySke: Inspiration

Map: applies a function to every element of the structure.

$$map : (A \to B) \to List\ A \to List\ B$$
$$map\ f\ [x_1; ...; x_n] = [f\ x_1; ...; f\ x_n]$$

Formally:

$$\begin{cases} map\ f\ [\,] & = & [\,] \\ map\ f\ (x :: xs) & = & (f\ x) :: (map\ f\ xs) \end{cases}$$

PySke: Inspiration

Reduce: collapse all the elements of the structure.

$$reduce : (A \to A \to A) \to List\ A \to A$$
$$reduce\ (\oplus)\ [x_1; ...; x_n] = x_1 \oplus ... \oplus x_n$$

PySke: Inspiration

Reduce: collapse all the elements of the structure.

$$reduce : (A \to A \to A) \to List\ A \to A$$
$$reduce\ (\oplus)\ [x_1; ...; x_n] = x_1 \oplus ... \oplus x_n$$

Formally:

$$\begin{cases} reduce\ \oplus\ [\,] & = & \iota_\oplus \\ reduce\ \oplus\ (x :: xs) & = & x \oplus (reduce\ \oplus\ xs) \end{cases}$$

PySke: Inspiration

Scan: Accumulate values through the list

$$scan : (A \to A \to A) \to List\ A \to List\ A$$

$$scan\ (\oplus)\ [x_1; x_2; ...; x_n] = [x_1; x_1 \oplus x_2; ...; x_1 \oplus x_2 \oplus ... \oplus x_n]$$

PySke: Inspiration

Scan: Accumulate values through the list

$$scan : (A \to A \to A) \to List\ A \to List\ A$$

$$scan\ (\oplus)\ [x_1; x_2; ...; x_n] = [x_1; x_1 \oplus x_2; ...; x_1 \oplus x_2 \oplus ... \oplus x_n]$$

Formally? Boring definition.

PySke: Inspiration

Scan: Accumulate values through the list

$$scan : (A \to A \to A) \to List\ A \to List\ A$$
$$scan\ (\oplus)\ [x_1; x_2; ...; x_n] = [x_1; x_1 \oplus x_2; ...; x_1 \oplus x_2 \oplus ... \oplus x_n]$$

Formally? Boring definition.
But not that much if $\oplus$ is symmetric:
$scan\ (\oplus)\ l = reverse \circ (scan'\ (\oplus)\ l) \circ reverse$ with

$$\begin{cases} scan'\ \oplus\ [\,] & = & [\,] \\ scan'\ \oplus\ (x :: xs) & = & (reduce\ \oplus\ (x :: xs)) :: (scan'\ \oplus\ xs) \end{cases}$$