

INVERSE PROBLEM, SENSITIVITY ANALYSIS AND FUNCTIONAL PROGRAMMING WITH OCAML

F. Clément (Inria-Rocquencourt, Estime)

P. Weis (Inria-Rocquencourt, Cristal)

H. Ben Ameer (ENIT/LAMSIN, Tunis)

I. Berre (U. of Bergen, Norway)

E. Marchand (Inria-Rocquencourt, Estime)

OUTLINE

- Introduction
- OCaml, a **safe** functional programming language
- OCamlP3I, a **structured approach** to parallel programming
- Refinement indicators
- Sensitivity analysis
- Conclusions

INTRODUCTION

Motivation: implementation of **complex algorithms**

Code coupling (**domain decomposition**)

Inversion (**refinement indicators**, sensitivity analysis)

Generic implementation: \rightsquigarrow coupling platform, inversion platform

Abstraction of functionalities (eg, communications for parallelism)

\Rightarrow Need for **high-level** programming capabilities

Applications: underground **nuclear waste disposal** (GDR MOMAS, ANDRA)

3D **flow** + **transport** simulations, **chemical** phenomena

Estimation of hydraulic parameters, computation of **uncertainties**

HIGH-LEVEL PROGRAMMING TOOLS

Functional programming languages (**safety** is one of the main issue)

OCaml, Haskell, Lisp/Scheme, . . .

Skeletal parallel programming

Tools to abstract communications (eg, management of processes, data flow):

OCamIP3I (OCaml), GpH (Haskell), Eden (Haskell),
eSkel (C/MPI), Muesli (C++/MPI), Muskel (Java), Assist, HOC. . .

Bleeding edge tools \Rightarrow close **interaction with the developers** is invaluable

Automatic differentiation

Source transformation (Tapenade, Adifor), operator overloading (**AdoIC**)

OCAML

Distributed by INRIA since 1985 (<http://caml.inria.fr/>)

⇒ Unison, MLdonkey, Coq, FFTW, cdttools, ...

High-level language based on clear theoretical foundation and formal semantics

Strong typing: static and automatic ⇒ safety

Segmentation fault or bus error never happens

A program that compiles is close to be correct

Functional language ⇒ close to the mathematics

```
# let compose f g = function x -> g (f x);;
```

```
compose : ( $\alpha \rightarrow \beta$ )  $\rightarrow$  ( $\beta \rightarrow \gamma$ )  $\rightarrow$   $\alpha \rightarrow \gamma$ 
```

Polymorphism, modules, objects ⇒ complex structures/algorithms

- It requires a different way of thinking

OCAMLP3L

Developed at INRIA, Paris 7 and Pisa (<http://www.ocamlp3l.org/>)

P3I skeleton programming within OCaml

P3I provides generic programming patterns (= skeletons)

OCaml allows high-level implementation of algorithms

OCamlP3I provides determinism

Template based compilation

⇒ three interpretations of skeletons: graphical, sequential, and parallel

Theorem: Sequential and parallel modes always produce the same result

⇒ design and debug in sequential mode (on a local PC/Linux),
exploitation in parallel mode (on a cluster)

Available skeletons

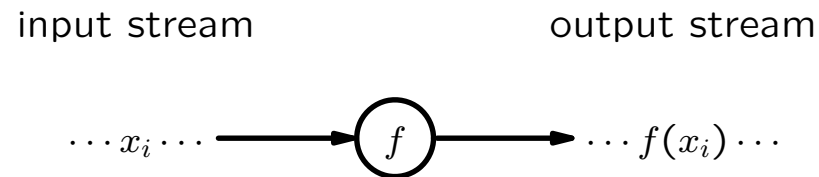
pipe, farm, mapvector, reducevector, seq, parfun, loop, pardo

OCAML3L SKELETONS: STREAM PROCESSING FUNCTIONS

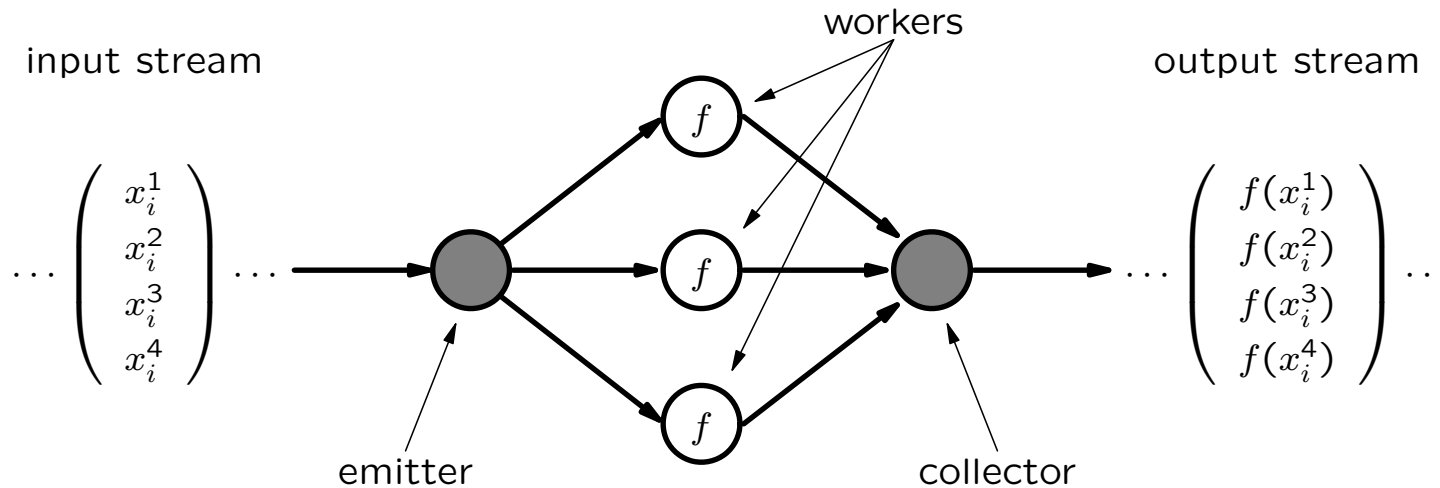
Stream: (potentially) infinite sequence of items

An example: `mapvector (seq f, n)`

`seq f`



`mapvector (seq f, 3)`



PARADIGM FOR THE IMPLEMENTATION OF COMPLEX ALGORITHMS

- Heavy computations by **external programs** written in **any language**
- **Driver** written in **OCaml**
- **Parallelism** is handled by **OCamlP3I**
- **Communications** with external programs follow a specific **protocol** using pipes on standard I/O

≅ Alternative to MPI + Corba

Communication protocol: safe, simple (**OCaml**, **C++**, Fortran, Java)

The driver sends a **Task** communication (= function name + its arguments) and after computation, the worker sends back

- either a **Result** communication (= values)
- or an **Error** communication (= exception value)

APPLICATIONS

- Studies for a deep underground nuclear waste disposal

Code coupling (domain decomposition)

Additive Schwarz algorithm (2D Poisson's equation)

Robin-to-Robin interface operator (3D flow equation)

⇒ Neumann-Neumann preconditioning + balancing (3D flow + transport)

Inversion, sensitivity analysis

⇒ Refinement indicators (1D/2D diffusion equation)

⇒ Uncertainty computation using SVD (3D flow + transport + precipitation)

REFINEMENT INDICATORS

Forward model $F : p \mapsto c$

Inverse problem: $F^{-1}d?$

$$\min_p J = \frac{1}{2} \|d - F(p)\|^2 \quad \nabla_p J(p_0) = -F'(p_0)^T (d - F(p_0))$$

Parameterization $P : m \mapsto p$ with $\dim m \ll \dim p$

$$\min_m (J \circ P) = \frac{1}{2} \|d - F(P(m))\|^2 \quad \nabla_m (J \circ P)(m_0) = P'(m_0)^T \nabla_p J(P(m_0))$$

eg, zonation (or clustering)

Refinement indicators: iterative process giving an “optimal” P^n and m_{opt}^n

Remark: $F = \text{identity} \Rightarrow$ image segmentation

Implementation

The OCaml driver runs the refinement indicators algorithm

An external program minimizes wrt m , and computes the gradient wrt p

DEFINITIONS

Mesh: $T_h = \cup_{i \in \mathcal{I}} K_i$

Zonation: $T_h = \cup_{j \in \mathcal{J}^n} Z_j^n$ with $Z_j^n = \cup_{i \in \mathcal{I}_j^n} K_i$ ($\mathcal{I} = \sqcup_{j \in \mathcal{J}^n} \mathcal{I}_j^n$)

P^n : $m^n = (m_j^n)_{j \in \mathcal{J}^n} \mapsto p = (p_i)_{i \in \mathcal{I}}$ with $K_i \subset Z_j^n \Rightarrow p_i = m_j^n$

Projection

\tilde{P}^n : $p = (p_i)_{i \in \mathcal{I}} \mapsto m^n = (m_j^n)_{j \in \mathcal{J}^n}$ with $m_j^n = \frac{1}{\sum_{i \in \mathcal{I}_j^n} |K_i|} \sum_{i \in \mathcal{I}_j^n} |K_i| p_i$

Cut: $Z_j^n = Z_{j_1}^{n+1} \cup Z_{j_2}^{n+1}$ with $\mathcal{I}_j^n = \mathcal{I}_{j_1}^{n+1} \sqcup \mathcal{I}_{j_2}^{n+1} \Rightarrow P^{n+1}$

ALGORITHM

Initialization: $P^1 = (1 \dots 1)^T$ (ie, $Z_1^1 = T_h$)

given m_{init}^1 , compute $m_{\text{opt}}^1 = \arg \min_{m^1} J \circ P^1(m^1)$ and $J_{\text{opt}}^1 = J \circ P^1(m_{\text{opt}}^1)$

Iterations: for $n \geq 1$,

if $J_{\text{opt}}^n < \varepsilon$, then stop, else compute $g_{\text{opt}}^n = \nabla_p J(P^n(m_{\text{opt}}^n))$

for $k \in \mathcal{K}^{n+1}$ (cuts), define $P^{n+1,k}$ and compute $I^{n+1,k} = \left| \left(P_{j_1}^{n+1,k} \right)^T g_{\text{opt}}^n \right|$

define $\mathcal{K}_{\text{eff}}^{n+1} \subset \mathcal{K}^{n+1}$ for highest indicators

for $k \in \mathcal{K}_{\text{eff}}^{n+1}$, set $m_{\text{init}}^{n+1,k} = \tilde{P}^{n+1,k} P^n m_{\text{opt}}^n$ and

compute $m_{\text{opt}}^{n+1,k} = \arg \min_{m^{n+1,k}} J \circ P^{n+1,k}(m^{n+1,k})$, $J_{\text{opt}}^{n+1,k} = J \circ P^{n+1,k}(m_{\text{opt}}^{n+1,k})$

define $k_0 = \arg \min_{k \in \mathcal{K}_{\text{eff}}^{n+1}} J_{\text{opt}}^{n+1,k}$

and set $P^{n+1} = P^{n+1,k_0}$, $m_{\text{opt}}^{n+1} = m_{\text{opt}}^{n+1,k_0}$, $J_{\text{opt}}^{n+1} = J_{\text{opt}}^{n+1,k_0}$

UNCERTAINTY COMPUTATION

Forward model $F : p \mapsto c$

First order approximation: $\Delta c \simeq F'(p_0)\Delta p$

Singular Value Decomposition: $F'(p_0) = USV^T$

U, V = unitary matrices, $S = \text{diag}(s_k)$ with $s_{k+1} \geq s_k \geq 0$

\Rightarrow (local) principal directions

Implementation

The OCaml driver assembles the Jacobian matrix by lines or by columns (possibly in parallel) and displays singular values and singular vectors

A C++ external program performs automatic differentiation using AdolC for a parallel 3D flow+transport simulator to compute a line or a column of $F'(p_0)$

A C (or Fortran) external program computes the SVD using Lapack

CONCLUSIONS

- Functional languages are **safe** (theorems on codes can be proven)
- **Paradigm** for the implementation of complex numerical algorithms:
 - **generic** OCaml driver
 - driven external codes can be written in any language (**reusability**)
 - inter-language communications use a **safe and simple protocol**
 - “**automatic**” parallelism through OCamlP3I library
(sequential and parallel modes **always** produce the same result)
- \rightsquigarrow Inversion **platform** (refinement indicators, SVD)
- Link with image segmentation techniques (**level set** methods)