Efficiently Coding With Julia

μHackademy 2023

A. A. Tamburini

Introduzione

Perché Julia?

- FOSS
- Velocità di esecuzione
- Velocità di sviluppo
- Curva di apprendimento
- Sintassi
- Non Object Oriented
- Flessibilità
- General Purpose

Quando non Julia? (anche se...)

- Processore non x86 o ARM (e pochissimi altri)
- No Sistema Operativo
- Pochissima Memoria Libera (< 512 MB)
- Principalmente I/O
- Avvio Veloce
- Tempo di Esecuzione Deterministico

Applicazioni Principali

- Calcolo Numerico
- Simulazione
- Supercomputing
- Programmazione Matematica (Ottimizzazione)
- Statistica
- Machine Learning

Caratteristiche Tecniche

- Broadcasting
- Tipizzazione Dinamica
- Multiple Dispatch
- Shell Interattiva
- Compilazione Just-Ahead-Of-Time (JAOT) (LLVM back end)
- Possibilitá di compilare binari nativi (*PackageCompiler.jl*)
- Garbage Collector
- Macro
- Chiamate a funzioni C e Fortram senza wrapper
- Facile parallelizzare

Broadcasting

Senza

```
myf = x -> sqrt(sum(x))
var = [[1, 2, 3], [5, 10]]
res = []
for inner in var
        push!(res, myf(inner))
end
```

Con

```
myf = x -> sqrt(sum(x))
var = [[1, 2, 3], [5, 10]]
res = myf.(var)
```

Multiple Dispatch

Polimorfismo

```
void add(Foo o) { ... }
void add(Bar o) { ... }
Foo o = new Bar();
add(o); // calls add(Foo)
```

Multiple Dispatch

```
void add(Foo o) { ... }
void add(Bar o) { ... }
Foo o = new Bar();
add(o); // calls add(Bar)
```

Macro

Descrizione

- Scrivere codice che scrive codice durante la compilazione
- allo scopo di migliorare la leggibilità
- e creare Domain Specific Languages

Macro

Esempio

```
using JuMP, HiGHS
model = Model(HiGHS.Optimizer)
@variable(model, x[1:3] .>= 0)
@objective(model, Min, sum(x))
optimize!(model)
```

Parallelizzazione

Esempio

```
a = randn(1000)
@distributed (+) for i = 1:100000
      sum(a[rand(1:end)])
end
```

Confronto

Un semplice ciclo for

C

```
#include <stdio.h>
void main(){
	long res = 0;
	for(long i=0; i<=10000000000;i++) res += i;
}
```

Julia

```
sum(1:10000000000)
```

se moltiplico per 10000000?

Confronto

Un semplice ciclo for

python-1

```
k = 0
for i in range(0, 10000000000):
        k += i
```

python-2

```
import numpy as np
np.ones(10000000000).sum()
```

python-3

```
sum(range(1,10000000000))
```

Quindi Easy...

Uguali?

F2

```
function F2(a, b)
    return a+b
end
```

F1

```
function F1(a::Int64, b::Int64)::Int64
    return a+b
end
```

```
map(j -> Fx(i, j), 1:10000)
```

Quindi Easy...

Uguali?

F3

```
function F3(a::Int64, b::Int64)::Int64
    global i += a
    return a+b
end
```

F1

```
function F1(a::Int64, b::Int64)::Int64
    return a+b
end
```

```
map(j -> Fx(i, j), 1:10000)
```

Quindi Easy...

Uguali?

F?

```
sum(1 for _ in 1:100_000)
```

F?

```
sum([1 for _ in 1:100_000])
```

Quindi Easy...

Uguali?

F?

```
b = zeros(10_000, 10_000);
b[1:100, 2:3] .= 1;
b.^2
```

F?

```
a = spzeros(10_000, 10_000);
a[1:100, 2:3] .= 1;
a.^2
```

```
b[1:10_000, 1:7000] .= 1;
b.^2
```

```
a[1:10_000, 1:7000] .= 1;
a.^2
```

Quindi Easy...

Uguali?

F?

```
function copycols(in::Vector{Float64})
out = zeros(length(in), length(in))
for i in 1:length(in)
        out[:, i] = in
end
return out
end
```

F?

```
function copyrows(in::Vector{Float64})
out = zeros(length(in), length(in))
for i in 1:length(in)
        out[i, :] = in
end
return out
end
```

Quindi Easy...

Uguali?

F?

```
function xinc(x)
    return [x, x+1, x+2]
end;
function loopinc()
    y = 0
    for i = 1:10^7
        ret = xinc(i)
        y += ret[2]
    end
end;
```

F?

```
function xinc!(ret::Vector{Int64},
x::Int64)
    ret[1] = x; ret[2] = x+1
    ret[3] = x+2; nothing
end;
function loopinc_pre()
    ret = Vector{Int64}(undef, 3)
    y = 0
    for i = 1:10^7
        xinc!(ret, i)
        y += ret[2]
    end
end;
```

Quindi Easy...

Views

F?

```
fcopy(x) = sum(x[2:end-1]);
```

F?

```
@views fview(x) = sum(x[2:end-1]);
```

Quindi Easy...

...Non troppo

Tenere d'occhio

- Cosa il Compilatore sa
- Allocazioni di memoria
- Come la memoria é allocata (e la cache)
- Semantica della sintassi a basso livello
- Strutture dati a basso livello
- Garbage Collector!
- Il processore (SIMD?)

Programmazione Matematica (Ottimizzazione)

Libreria JuMP

- FOSS
- Facile da integrare con altri programmi
- Sintassi simile alle espressioni matematiche
- Indipendente dal solver
- Velocitá di esecuzione
- Possibilitá di accedere a feature specifiche dei solver (perdendo generalitá)
- Oltre alla documentazione, esiste un eccellente libro che guida passo a passo l'apprendimento

Programmazione Matematica (Ottimizzazione)

Vehicular Routing Problem

- Una flotta di veicoli
- Un grafo pesato completo $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{C})$
- Un insieme di clienti $v \in \mathcal{V} \setminus \{o\}$ verso cui trasportare merce
- Un deposito $o \in \mathcal{V}$
- Variabili: Tutte le possibili rotte ammissibili $\mathcal{R}$. In totale $(|\mathcal{V}| - 1)!e$ rotte
- Obiettivo: Minimizzare il costo di trasporto

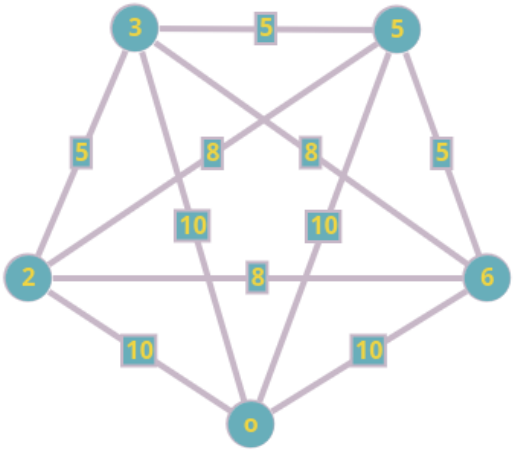
Programmazione Matematica (Ottimizzazione)

$$\min \quad \sum_{r \in \mathcal{R}} p_r \lambda_r \tag{1a}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} a_{ir} \lambda_r = 1 \qquad \forall i \in \mathcal{V}, \tag{1b}$$

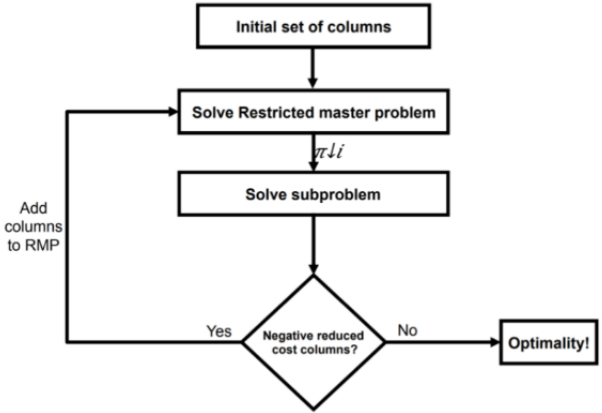
$$\lambda_r \in \{0, 1\} \qquad \forall r \in \mathcal{R} \tag{1c}$$

Applicazioni



Applicazioni

Algoritmo: Column Generation



Applicazioni

Algoritmo: Column Generation

- Euristica iniziale: ogni veicolo visita un cliente e torna al deposito
- Generiamo le rotte promettenti usando i duali***
- Se non ci sono piú rotte promettenti siamo all'ottimo**

** Per il rilassato

Se non é una soluzione intera => branching/cutting

*** Cosa sono i duali?

Costo marginale (ovvero relativo alla soluzione corrente) ottenuto dal visitare un determinato cliente