FIRST LINE OF TITLE
SECOND LINE OF TITLE

THIS IS A TEMPORARY TITLE PAGE
It will be replaced for the final print by a version
provided by the service academique.

Thèse n. 1234 2011
présenté le 12 Mars 2011
à la Faculté des Sciences de Base
laboratoire SuperScience
programme doctoral en SuperScience
École Polytechnique Fédérale de Lausanne

pour l'obtention du grade de Docteur ès Sciences
par

Paolino Paperino

acceptée sur proposition du jury:

Prof Name Surname, président du jury
Prof Name Surname, directeur de thèse
Prof Name Surname, rapporteur
Prof Name Surname, rapporteur
Prof Name Surname, rapporteur

Lausanne, EPFL, 2011

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Contents

List of figures iii

List of tables v

1 Introduction 1

I Torsion 5

2 Geometry of smooth and discret curves 7

2.1 Introduction 7
2.2 Parametric Curves 7
2.3 Frenet's Trihedron 9
2.4 Curvature 10
2.5 Torsion 11
2.6 Curve Framing 12
2.7 Discrete Curvature 18

3 Elastic rod : variational approch 25

3.1 Introduction 25
3.2 Kirchhoff rod 26
3.3 Curve-angle representation 28
3.4 Strains 29
3.5 Elastic energy 29
3.6 Quasistatic assumption 30
3.7 Energy gradient with respect to θ : moment of torsion 30
3.8 Energy gradient with respect to x : internal forces 33
3.9 Numerical Model 42
3.10 Discretization 42
3.11 Connection 42

4 Elastic rod : a novel element from Kirchhoff equations 45

4.1 Introduction 45
4.2 Kirchhoff's law 45

4.3 Constitutive equations 47
4.4 Internal forces and moments 48
4.5 Equations for the dynamic of rods 48
4.6 Main hypothesis 48

II Connection **51**

5 Calculus of variations **53**
5.1 Introduction 53
5.2 Spaces 53
5.3 Derivative 55
5.4 Gradient vector 58
5.5 Jacobian matrix 58
5.6 Hessian 59
5.7 Functional 59

6 Bench for HPC **61**
6.1 Introduction 61
6.2 Languages 61
6.3 From syntax to processor 61
6.4 Benchmark 61
6.5 Memory allocation and garbage collection 63

List of Figures

2.1 Different osculating circles for a spiral. 11
2.2 Differential definition of Frenet's trihedron at given point P_0. 12
2.3 Geometric torsion and rotation of the osculating plane 13
2.4 Adapted moving frame $F(s) = \{\boldsymbol{e}_3(s), \boldsymbol{e}_1(s), \boldsymbol{e}_2(s)\}$ where $\boldsymbol{e}_3(s) = \boldsymbol{t}(s)$. . . . 14
2.5 Geometric interpretation of the Darboux vector of a moving frame. 15
2.7 Variation of the vertex-based discrete curvature. 19
2.8 Variation of the edge-based discrete curvature. 19
2.9 Definition of the osculating circle for discrete curves. 20
2.10 Another definition of the osculating circle for arc-length parametrized curves. 20
2.11 Discrete curvature comparison for $\alpha \in [0.5, 2]$ 21
2.12 Another definition of the osculating circle for arc-length parametrized curves. 22
2.13 Discrete curvature comparison for $\alpha \in [0.5, 2]$ 23
2.14 Another definition of the osculating circle for arc-length parametrized curves. 23
2.15 Another definition of the osculating circle for arc-length parametrized curves. 24

3.1 Repères de Frenet attachés à γ. 34
3.2 $\tilde{F}$ is obtained by rotating $\tilde{F}_\epsilon$ around $\boldsymbol{t}$ of an angle Ψ_ϵ. 35
3.3 $\tilde{F}_\epsilon$ is obtained by parallel tranpsporting F_ϵ from $\boldsymbol{t}_\epsilon$ to $\boldsymbol{t}$. This operation could be seen as a rotation around $\boldsymbol{t}_\epsilon \times \boldsymbol{t}$ of an angle α_ϵ. 35

6.1 Each operator is evaluated on a vector of Float64 of size $n = 10^6$ for about 10s. Results are given relatively to MKL performance (MKL = 1). 62
6.2 Nonlinear cost of memory allocation for arrays (Float64). 66

List of Tables

6.1 Memory allocations for various methods computing sqrt(a) for $n = 10^4$ . . . 64

Part I
Torsion

Connection Part II

6 Bench for HPC

6.1 Introduction

In this section aims at providing basic but reliable guidlines to produce fast and mannagable code for our algorithms

Most compilers with which you are probably familiar are standalone programs which take as input some source code text and compile it into machine code (or some other target representation).

[AMR02]

6.2 Languages

- Csharp - Julia - C++ - Intel MKL - OpenBLAS

6.3 From syntax to processor

A short story about how a code is translated to get machin instructions

6.4 Benchmark

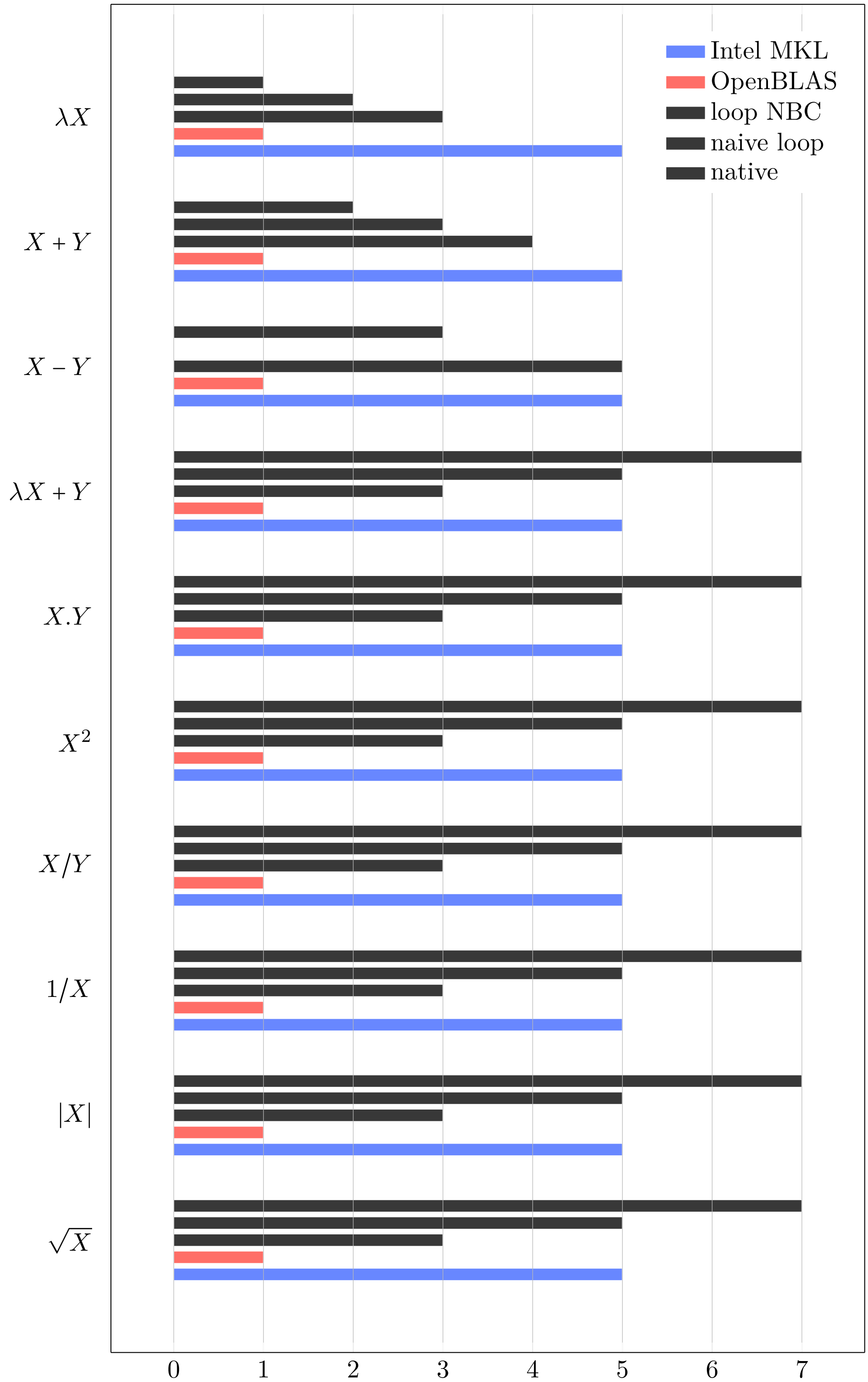

Figure 6.1 – Each operator is evaluated on a vector of Float64 of size $n = 10^6$ for about 10s. Results are given relatively to MKL performance (MKL $= 1$).

6.5 Memory allocation and garbage collection

```julia
using VML

# wrap functions to avoid global scoping while testing
function sqrt_jvectorized{T<:Number}(a::Vector{T})
    sqrt(a)
end
function sqrt_jbroadcast{T<:Number}(a::Vector{T})
    broadcast(sqrt,a)
end
function sqrt_jbroadcast!{T<:Number}(dest::Vector{T}, a::Vector{T})
    broadcast!(sqrt,dest,a)
end
function sqrt_jmap{T<:Number}(a::Vector{T})
    map!(sqrt,a)
end
function sqrt_jmap!{T<:Number}(dest::Vector{T}, a::Vector{T})
    map!(sqrt,dest,a)
end
function sqrt_jloop{T<:Number}(dest::Vector{T},a::Vector{T})
    @inbounds for i in eachindex(a) dest[i]=sqrt(a[i]) end
end

function sqrt_bench()
    # define vector size and floating precision
    n = 1_000
    T = Float64
    # allocate vectors
    dest  = ones(T,n)
    @time a = rand(T,n)
    # bench
    gc()
    gc_enable(false)
    @time sqrt_jvectorized(a)
    @time sqrt_jbroadcast(a)
    @time sqrt_jbroadcast!(dest,a)
    @time sqrt_jmap(a)
    @time sqrt_jmap!(dest,a)
    @time VML.sqrt(a)
    @time VML.sqrt!(dest,a)
    gc_enable(true)
end

sqrt_bench()
```

sqrt | Float64	CPU (ns/el)	Allocation (Bytes)
Allocation	5	8,112
Julia vectorized	12	8,080
Julia broadcast	24	8,352
Julia broadcast!	7	16
Julia map	164	48,000
Julia map!	122	48,000
VML (allocation)	7	8,080
VML (in-place)	5	0

Table 6.1 – Memory allocations for various methods computing sqrt(a) for $n = 10^4$

Toutes les syntaxes ne sont pas égales en termes de gestion de la mémoire. Le problème, c'est le passage de la GC qui est couteux en temps. Donc il faut essayer de minimiser l'utilisation mémoire. Idéalement le problème peu rester dans le cache du processeur (mémoire d'accès bcp plus rapide que la RAM). Donc la meilleur stratégie consiste à pré-allouer les tableaux et à faire des opération "in-place" au maximum, c'est à dire d'écraser les donner au fur et à mesure du calcul.

On remarque que l'allocation mémoire est très différente d'une fonction à l'autre. Il est important de privilégier des opération "in-place" pour contenir l'allocation mémoire, sinon on risque de déclencher la GC qui est couteuse.

Les temps CPU sont indicatifs car le bench est fait sur une durée caractéristique trop courte

```
using DataFrames, VML

function sqrt_bench()

    # vector size
    N = [1,2,3,4,5,6,7,8,9,
         10,20,30,40,50,60,70,80,90,100,200,300,400,500,750,
         1_000,2_500,5_000,7_500,10_000,100_000,1_000_000]

    # dataframe for results
    df = DataFrame(N=[],ALLOC=Float64[],JULIA=Float64[],MKL=Float64[])

    @inbounds for i in 1:length(N)
        T = Float64
        n = N[i]
        a = rand(T,n)
        dest = zeros(T,n)

        # evaluate sqrt and allocation
        # for small n @elapsed applies to a bunch of evaualations
        nrep = 1000
        ncycle = 10_000 ÷ n + 1

        # trigger garbage collection
        gc()
        talloc = 0.0 ; tcpu1 = 0.0 ; tcpu2 = 0.0
        for j in 1:nrep
            talloc += @elapsed for k in 1:ncycle Vector{T}(n) end
            tcpu1 += @elapsed for k in 1:ncycle sqrt(a) end
        end

        # scale results (ns/element)
        talloc = talloc / nrep / ncycle / n * 1e9
        tcpu1  = tcpu1 / nrep / ncycle / n * 1e9

        # write results
        push!(df,[n,talloc, tcpu1])
    end
    df
end

sqrt_bench()
```

Ici on met en évidence la non linéarité du coût d'allocation par élément d'un tableau de taille n. On remarque que la différence entre le coût de sqrt et le coût de l'allocation est constante : c'est le coût de sqrt hors allocation. Attention, cette notion est "language dependent" car les allocations sont gérées par la GC.

Remarque, on trouverait sans doute la même chose pour MKL, à cause du marshalling : le coût d'appel à une fonction C est supérieur à celui d'une fonction managée (cf HPC .Net)

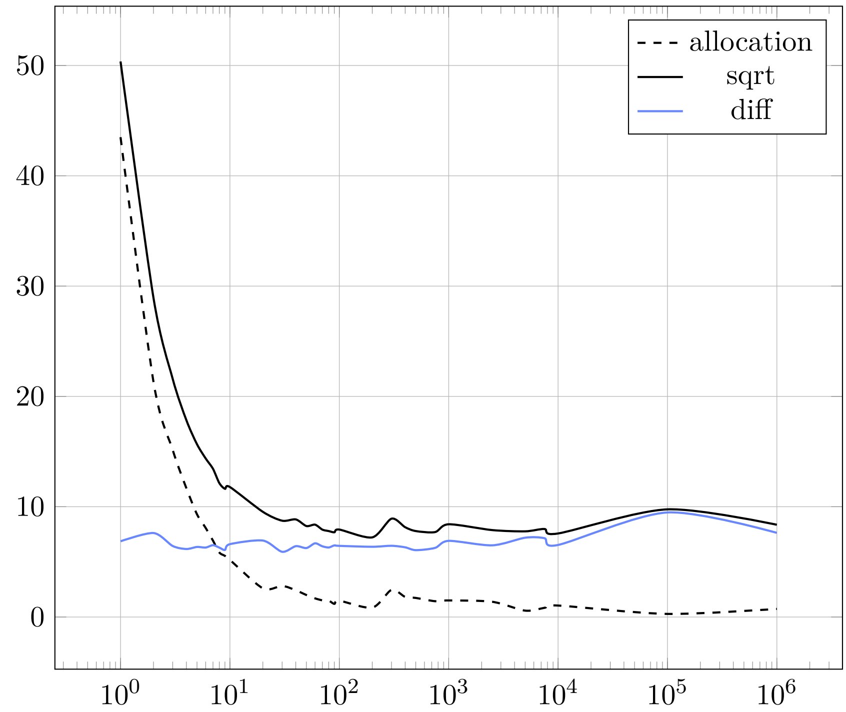

Figure 6.2 – Nonlinear cost of memory allocation for arrays (Float64).

https://msdn.microsoft.com/en-us/library/ms973852.aspx

Bibliography

[AMR02] Ralph Abraham, Jerrold E. Marsde, and Tudor Ratiu. *Manifolds, Tensor Analysis, and Applications (Ralph Abraham, Jerrold E. Marsden and Tudor Ratiu)*. 2002.