New Scientific Programming Language

Taha BEN SALAH (NOCCS, ENISo, Sousse University)
Taoufik AGUILI (Sys’Com, ENIT, UTM University)
Agenda

- Introduction
- Existing Scientific Programming Languages
- Presenting the new Scientific Programming Language
- Comparisons and results
- Conclusion
What is a scientific language

A scientific language is a programming language optimized for the use of mathematical formulas and matrices.
Existing Scientific Languages:
DSL : Domain Specific Languages
MATLAB, Maple, FORTRAN, ALGOL, APL,J, Julia, Wolfram Language/Mathematica, and R.

Non Scientific Languages used by scientists
GPL: General Purpose Languages

C/C++, Python, Scala, Java
Why a new Programming Language

- Limits of the existing Programming Languages
- Simpler syntax or More Powerful syntax
- Better portability
- Better integration
- Better performance
# Existing Languages

<table>
<thead>
<tr>
<th>Type</th>
<th>Python</th>
<th>Java/Scala/Kotlin</th>
<th>Julia</th>
<th>Matlab</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm</td>
<td>GPL</td>
<td>GPL</td>
<td>DSL</td>
<td>DSL</td>
<td>DSL</td>
</tr>
<tr>
<td>Price</td>
<td>Free &amp; Open Source</td>
<td>Free &amp; Open Source</td>
<td>Free &amp; Open Source</td>
<td>Commercial</td>
<td>Free &amp; Open Source</td>
</tr>
<tr>
<td>Advantages</td>
<td>Community, Simplicity, Libraries</td>
<td>Community, Tools, Libraries</td>
<td>Simplicity, Performance</td>
<td>Simplicity, Toolboxes (Simulink)</td>
<td>Toolboxes</td>
</tr>
<tr>
<td>Limitations</td>
<td>Performance, tools, compatibility</td>
<td>Steep learning curve</td>
<td>Small popularity, few libraries</td>
<td>Commercial, not appropriate for big projects, performance</td>
<td>not appropriate for complex projects</td>
</tr>
<tr>
<td></td>
<td>Python</td>
<td>Java/Scala/Kotlin</td>
<td>Julia</td>
<td>Matlab</td>
<td>R</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------</td>
<td>-----------------------</td>
<td>-----------------------------</td>
<td>------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Dyn Typed</td>
<td>Static Typed</td>
<td>Dyn Typed</td>
<td>Dyn Typed</td>
<td>Dyn Typed</td>
</tr>
<tr>
<td><strong>Paradigm</strong></td>
<td>OO / Proc</td>
<td>OO / Func</td>
<td>Proc, loosely OO/Func</td>
<td>Proc - Loosely OO</td>
<td>Proc - Loosely OO</td>
</tr>
<tr>
<td><strong>Compiled?</strong></td>
<td>Interpreted + compiled</td>
<td>Compiled+JIT</td>
<td>compiled+JIT</td>
<td>Interpreted + compiled</td>
<td>Interpreted</td>
</tr>
<tr>
<td><strong>Toolchain</strong></td>
<td>REPL, Interpreter + IDE</td>
<td>Compiler+Externa Build Tools</td>
<td>Compiler</td>
<td>Studio</td>
<td>Studio</td>
</tr>
<tr>
<td><strong>Dependencies &amp; Versioning</strong></td>
<td>Strong support</td>
<td>Strong support</td>
<td>Weak Support</td>
<td>Weak Support</td>
<td>Weak Support</td>
</tr>
</tbody>
</table>
## Existing Languages

<table>
<thead>
<tr>
<th>Feature</th>
<th>Python</th>
<th>Java/Scala/Kotlin</th>
<th>Julia</th>
<th>Matlab</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel / Concurrent / Distributed</td>
<td>Supported</td>
<td>Strongly supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>support CPU/GPU</td>
<td>Via Libraries</td>
<td>JOCL, JogAmp, and JavaCL</td>
<td>Via Libraries</td>
<td>Via Libraries</td>
<td>Via Libraries</td>
</tr>
<tr>
<td>Libraries and Ecosystem</td>
<td>Large community</td>
<td>Extremely Large community</td>
<td>Small community</td>
<td>Large community</td>
<td>Large community</td>
</tr>
<tr>
<td>Tools support</td>
<td>Good support by Commercial IDE</td>
<td>Best IDEs</td>
<td>Little support</td>
<td>Product Studio</td>
<td>Product Studio</td>
</tr>
<tr>
<td>Performance</td>
<td>Bad performance</td>
<td>Good Performance</td>
<td>Better performance</td>
<td>Bad performance</td>
<td>Bad performance</td>
</tr>
</tbody>
</table>
# Existing Languages

<table>
<thead>
<tr>
<th></th>
<th>Python</th>
<th>Java/Scala/Kotlin</th>
<th>Julia</th>
<th>Matlab</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability</td>
<td>Bad Portability</td>
<td>Better Portability</td>
<td>Supported Win/Linux</td>
<td>Supported Win/Linux</td>
<td>Supported Win/Linux</td>
</tr>
<tr>
<td>Numeric</td>
<td>Libraries, bad integration</td>
<td>Libraries, bad integration</td>
<td>Complex / Matrices</td>
<td>Complex / Matrices</td>
<td>Complex / Matrices</td>
</tr>
<tr>
<td>Scientific</td>
<td></td>
<td></td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Calculation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbolic</td>
<td>Libraries, bad integration</td>
<td>Libraries, bad integration</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Scientific</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculation</td>
<td>bad</td>
<td>worst</td>
<td>good</td>
<td>good</td>
<td>average</td>
</tr>
<tr>
<td>Scientific</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Readability</td>
<td></td>
<td></td>
<td>good</td>
<td></td>
<td>average</td>
</tr>
<tr>
<td>Native</td>
<td>average</td>
<td>average</td>
<td>good</td>
<td>average</td>
<td>minimal</td>
</tr>
<tr>
<td>Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Journey to a new Programming Language

- Used **Matlab/Scilab/Octave** in simulations
  - As soon as the number of files becomes important (100) this is no more manageable
  - Serious performance issues
- Moved to **C/C++** (blas, ...)
  - Very hard to maintain too
- Moved to **Java**
  - Rewritten all of the code in Java
  - Written microwaves library
  - Good performance
  - Not accepted complexity by other research colleagues

- Moved to **Scala**
  - Refactored the code into
    - Hadruplots
    - Hadrumaths
    - Hadruwaves
  - Written ports to the libraries in Scala
  - Much better readability
  - Still
    - quite difficult to start a new “code” from scratch
    - Scala is difficult for non initiated to programming researchers
    - Compiling errors difficult to “understand”
    - Limitations inherited from the Java Language
    - Inconsistencies between Scala Collections & Java Collections
Hadra Language

- New programming Language that obviously learns from predecessors
- Focuses on Complex numbers, Vectors, and Matrices
- Base on the Java VM (compiles to Java Byte Code)
- Statically Typed
- Makes advantage of Unicode support

- Concise
- Readable
- Single file project
- Modular & Extensible
- Functional and OO
  - All constructs are functions (for, while, switch...)
  - All functions are Objects

- Introduces Elastic Calculation Concept: Numeric and Symbolic at the same time
- GPL as DSL
file: hello.hl
println(matrix(3,(i,j)->i+j));

shell:
> hl hello.hl

result:
[
  0 1 2
  1 2 3
  3 4 5
]

file: plot.hl
import net.thevpc.scholar.hadrumaths;
Plot.title("sinus function").asCurve.plot(sin(X)*II(0..2π));

shell:
> hl plot.hl
**Hadra: Literals**

```plaintext
int twelveDecimal = 12;
short sixteenBinary = 0b10000s;
bigint twelveHexBigInt = 0xCI;
bigdecimal tens = 10.2E23D;
long C = 300_000_000 GHz;
var μ₀ = 4π*10⁻⁷ H/m;
localdate d = t"2020-02-01";
Complex c = i+1;
var c2 = i+1;
var msg="the day is $d";
var json={a:1, b:'two'};
```

**Arrays**

```plaintext
int[5] tab ( i -> 2*i ) ; // [0, 2, 4, 6, 8]
tab[0..2] = [15, 20, 30]; // [15, 20, 30, 6, 8]
tab[2..4] = tab[4..2]; // [15, 20, 8, 6, 30]
int[5] tab2(1) ; // [1, 1, 1, 1, 1]
int[5] tab3(Math::random) ; // [0.1, 0.5, 0.2, 0.7, 0.1]
int[5] tab4=[1, 2, 3] ; // [1, 2, 3]
int[] tab5 = tab1 :+ tab2 :+ tab3; // concat
```
fun int sqsum(int ...a) { // sqsum(1,2)=5
    switch(a.length)
    case 0: 0;
    case 1: a[0];
    default: a[0]²+sqsum(a[1..]);}

fun boolean palindrome(int[] a) {
    a[..$/$2]==a[$/$2..];}

Extension function
fun double Complex::norm(this a) {a.abs();}
var c = i+1;
var v1=norm(c); var v2=c.norm();

class Complex(double r, double i){
    fun double abs(){sqrt(r²+i²);}
    fun Complex +(Complex o) {
        Complex(r+o.r,i+o.i);
    }
    fun Complex +(double o) {
        Complex(r+o,i);
    }
    fun Complex (double o)+ {
        Complex(r+o,i);
    }
}
Hadra : Matrices

**Matrix<int>** m1 =
```
[ 0 , 0 , 0 ; 0 , 1 , 2 ; 0 , 2 , 4 ];
```

**Matrix<int>** m2 = [ 1 , 2 , 3 ;
```
3 , 2 , 1 ];
```

**var** m3 =matrix(3,(i,j)->i*j); // = m1

**var** m3 =matrix(3,(i,j)->i*i+j); // complex matrix

**var** m3 =symMatrix(3,(i,j)->i*i+j);

**var** v1 =vector(3,(i)->i*i); // vector of complexes

**Sums**

**double** v=sum(1..1000, x -> sin(x²)) ;

**Scalar Products**

**int[]** tab1 = [ a , b , c ];

**int[]** tab2 = [ A , B , C ];

**int** v = tab1 ** tab2 ; // = aA+bB+cC

**int[][]** v2 = tab1 :** tab2 ; // = [ a**A , a**B , a**C b**A , b**B , b**C c**A , c**B , c**C ]
Hadra : Symbolic Programming

// function declared on [0..π], zero elsewhere
var f = \sin(X) \cdot \cos(Y) \cdot \Pi(0..\pi);

// symbolic derive
var g = derive(f, X);

// symbolic integration
var g = integrate(f, X, 0..\pi/2);

**Param m();**
**Param n();**

var fₙₙ = \sin(m \cdot X) \cdot \cos(n \cdot Y) \cdot \Pi(0..\pi);

var f₀₁ = fₙₙ(n -> 0, m -> 1);

var all_f = seq(fₙₙ, n -> 0..3, m -> 0..3);

Using Latex

// ## This is a markdown comment Title

// Here is an example of using latex expressions

var θ = X;

var f₁ = \cos² \theta - \sin² \theta;

var f₂ = \cos(θ)² \sin(θ)²;

Plot.plot(f₁, f₂);
Hadra: Elastic Calculation

- The runtime is responsible of switching from Symbolic to Numeric (and vice versa)
- No need to explicitly sym/dblquad (as in matlab)
- Rule Based decisions
- Includes simplications/transformations to detect usual expressions
- Numeric calculation is done only when needed

```plaintext
var formal_scalar_product = \sin(m*\textit{X})^{\textit{Y}} \cdot (0..\pi); 
var formal_scalar_product = \sin(\textit{X})^{\textit{Y}} \cdot (0..\pi); 
var numeric_scalar_product = \sin((1+\textit{X})/\cos(\textit{Y}))^{\textit{Y}} \cdot (0..\pi); 
var formal_simplifiable = \sin(\textit{X})^{2} + \cos(\textit{X})^{2}; 
```
Hadra Integration: Native Code (C/C++), Java / Scala

- Seamless integration with C/C++
- Uses JNA/JNA under the hoods
- Uses a specific annotation @native

```java
@native("c")
class StandardAccess {
    void printf(string format, object ... args);
    int scanf(string format, object ... args);
}
byte[32] bytes;
StandardAccess.scanf("%s", bytes);
StandardAccess.printf("%s [%s] %s %s!\n", "Your message", Native.toString(bytes), "is printed in C", "5.5.0");
```

- Seamless integration with Java/Scala

```java
JFrame f("java frame")
    .[visible=true; title="example"];
    f+Button("click me");
```
# Hadra vs the world

<table>
<thead>
<tr>
<th>Feature</th>
<th>Hadra</th>
<th>Java</th>
<th>Julia</th>
<th>C/C++</th>
<th>Python</th>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator overloading</td>
<td>yes**</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Superscript/Subscript</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Matrices / Complex</td>
<td>hadrumaths</td>
<td>library</td>
<td>yes</td>
<td>library</td>
<td>library</td>
<td>yes</td>
</tr>
<tr>
<td>Control structures overloading (redefine for/while)</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Single file project (with dependencies)</td>
<td>yes</td>
<td>no</td>
<td>Comp. opt.</td>
<td>Comp. Opt.</td>
<td>yes</td>
<td>Mex or javaaddpath</td>
</tr>
<tr>
<td>Paradigm</td>
<td>func/OO</td>
<td>OO</td>
<td>Proc</td>
<td>OO/Proc</td>
<td>OO</td>
<td>Proc, supports OO</td>
</tr>
<tr>
<td>BLAS and LAPACK</td>
<td>hadrumaths</td>
<td>JBLAS library</td>
<td>Seamless integration</td>
<td>Library integration</td>
<td>Library integration</td>
<td>MEX file + Library</td>
</tr>
<tr>
<td>Elastic Numeric Calculation</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Performance

- Current version of Hadra generates Java sources then compiles to ByteCode (for validation purposes)
- Small Performance enhancements due to literal optimizations
  - Regexp
  - Dates
  - Primitive types
- Performance tested against
  - https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html
  - Considered: the best results

<table>
<thead>
<tr>
<th>Language</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadra</td>
<td>108%</td>
</tr>
<tr>
<td>Java</td>
<td>115%</td>
</tr>
<tr>
<td>Julia</td>
<td>100%</td>
</tr>
<tr>
<td>C</td>
<td>65%</td>
</tr>
<tr>
<td>Python</td>
<td>3984%</td>
</tr>
</tbody>
</table>
Conciseness

- Comparing Hadra source length to Java equivalent code
- Using sample code
  - #1: Sci code: using operator overloading in hadrumaths
  - #2: Java purely procedural
  - #3 Data classes, Typical Java code
  - #4 average of all the above
Why Hadra

1. First OO Programming Language is “Small Talk”
   • Hadra means “Lots of talk” in Tunisian
   • It means also “Interesting Thing”

2. Hadra builds upon existing work and libraries in the Laboratory:
   ○ Hadrumaths
   ○ Hadruwaves
   ○ Hadruplots

3. The prefix hadru and the name hadra come from “Hadrumet”, the Phoenician name of Sousse, the city where the authors are from
Conclusion

Existing Sci Languages
- DSL / GPL Blurry boundaries
- No clear winner

Proposed a new Language
- Readable, Concise, Simple
- Based on JVM: portable

TODO: Tooling
- Under construction, Netbeans/Intellij Integ.
- Syntax Highlighting in Kate/Sublime etc.

TODO: Sources & Perf
- To be published shortly under OSS License
- (now as private GITHUB repository)
Thank you

taha.bensalah@gmail.com
taha.bensalah@eniso.u-sousse.tn
http://github.com/thevpc