Array programming languages

Alexander Harms

Institute for software engineering and programming languages

January 18, 2016
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>Introduction</td>
</tr>
<tr>
<td>II.</td>
<td>MATLAB</td>
</tr>
<tr>
<td>III.</td>
<td>APL</td>
</tr>
<tr>
<td>IV.</td>
<td>QUBE</td>
</tr>
<tr>
<td>V.</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
Array programming languages
Also known as vector or multidimensional languages

Array oriented programming paradigms

Multidimensional arrays as primary data structures

Arrays can be vectors, matrices, tensors or scalar values

Operations can apply to an entire set of data values without explicit loops of scalar operations

Used for numerical simulations, data analysis and evaluation
What is the reason for array programming
- Often easier and faster than traditional programming languages
- Function libraries allows to avoid loops
- A fast and efficient way of solving numerical problems
- Fast computing by multicore processors combined with implizit parallelsization
Introduction

- Can mostly be used without knowledge about computer architecture
- Very similar to common math notation
- Programmer can think about the data without thinking about how to handle loops
MATLAB
MATLAB (Matrix LABoratory), developed by MathWorks

- Initial release in 1984
- Commercial product
- Multi-paradigm, numerical computing environment
- Main features: matrix manipulation, function and dataplotting, algorithm implementation and interfacing with other programs
- Fundamental data type is an n-dimensional array of double precision
The MATLAB system consists of five parts:
- MATLAB as a language
- Working environment
- Graphic system
- Mathematical function library
- Application program interface

MATLAB code is written in the command window, characterized by the prompt `>>`

Code is like interactive calculations or executing text files
Declaring variables

```matlab
>> x = 3
x =
  3
>> x = 'Hello';
>> array = 1:5:21
array =
  1 5 10 15 20
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
  1 2 3
  4 5 6
  7 8 9
```
Matlab provides a powerful function library

```
>>array = linspace (0, pi , 7)
array =
   1 0.5236 1.0472 1.5708 2.0944 2.6180 3.1416
>>sum(array)
ans =
   11.9956
>> A = [1 1 1; 2 2 2; 3 3 3];
>> B = zeros(3:3);
>>A.*B
ans =
   1 1 1
   2 2 2
   3 3 3
```
No loops needed, so developer can focus on numerical problem

Off-by-one mistakes and others are ruled out

Programs are saved as MATLAB-files (m-files)

m-flies are scripts or functions

Scripts are a collection of commands, functions run calculations

MATLAB can also handle graphics
MATLAB graphics

```matlab
>> x = [ 1; 2; 3; 4; 5];
>> y = [ 0; 0.25; 3; 1.5; 2];
>> figure  % opens new figure window
>> plot(x,y)
```

![Graph of MATLAB graphics](image)
APL
- Named after a 1964 published book ‘A Programming Language’

- Developed by Dr. Kenneth E. Iverson (1920-2004)
  - Mathematician
  - Special interest in mathematical notation
  - Disliked the standard notation
  - Developed new system in 1957
  - Started to work for IBM in 1960
  - Wrote extended version (‘Iverson’s Better Math’)
  - IBM disliked the name, changed to ‘A Programming Language’

- APL was a notation for expressing mathematical expressions first
APL’s special set of non-ASCII symbols
- APL code is evaluated from right to left
- Just a few rules like parentheses for changing evaluation order
- Interactively written
- No need to declare variables
- Basic data structure is the array
- Special set of non-ASCII symbols is needed
APL code example

```
3x4+5
27
(3x4)+5
17
x←5
x←“Hello“
A←5+1 2 3 ⊢ A = 6 7 8
B←1 1 1
A+B
7 8 9
6 8 1 L 3 5 9
3 5 1
```
APL code can be very short. For the problem “Sum of 1 to 5“ a C-solution could be:

```c
#include <stdio.h>
int main(){
    int i = 0;
    int sum = 0;
    for (i = 1; i<=5; sum += i++);
    printf("Sum: %d/n", sum);
    return 0;
}
```

The APL-solution would be:

`+/⍳5`
Easy handling of array data
Powerful function library, no need for counts or loops
Code can be written very short
Amount of unusual symbols leads to cryptic code
Challenge to write APL ‘one-liner’ is hard to resist

\[
\text{life} \leftarrow \{↑1 \omega \vee.^3 4=+/,-1 0 1.⊖1 0 1.⌽⊂\omega}\]

- APL works best on multi core processors combined with implicit parallelization
- Mostly used in small projects, universities and research institutes
- Lateron, Iverson developed ‘J’
- based on APL, but uses only ASCII-symbols
QUBE

<table>
<thead>
<tr>
<th>Introduction</th>
<th>MATLAB</th>
<th>APL</th>
<th>QUBE</th>
<th>Conclusion</th>
</tr>
</thead>
</table>

A. Harms

26.01.16
- Developed by K. Trojahner in 2011
- Use dependent array types to check programs at compile time
- Combination of type checking and automatic theorem proving
- Rules out large classes of array boundary violations
o Arrays are characterized by their rank and shape

o The rank is a natural number of ist number of axes

o The shape describe the extent of each axis
QUBE extends QUBE\textsubscript{core} with a richer syntax, more base types and others like I/O

QUBE\textsubscript{core} comprises three layers
- QUBE\textsubscript{λ} (applied λ-calculus with dependent types)
- QUBE→ (integer vectors)
- QUBE[] (multidimensional arrays)
QUBE fun

• forms the basis of QUBE

• Most significant features
  • Dependent types
  • Refinement types

• In following
  • x: values
  • T: types
  • e: expressions
Refinement type \( \{x:T|e\} \)

- Describes the subset of values \( x \) of type \( T \) that satisfy the boolean expression \( e \).

- Example: the type ‘nat‘ are all integers with a value bigger than 0

\[
\text{type nat} = \{x: \text{int} | 0 \leq x\}
\]

- If \( e \) is true by any \( x \), the type \( \{x:T|\text{true}\} \) is equivalent to \( T \)
Dependent function type \( x: T_1 \rightarrow T_2 \)

- Binds the variable \( x \) of the domain type \( T_1 \) in the codomain type \( T_2 \)

- Example: addition of two integer numbers

\[
val+: x.int \rightarrow y.int \rightarrow \{v.int | v = x + y\}
\]

- Allows the result type to vary according to the supplied argument
Code example

```plaintext
let a = [1,2,3] in
let b = [1,2,1] in
let b = a.(2)<- a.(0) in
a.(2)

(* vector constructor*)

(* vector modification*)

=>* a.(1) = 3
```
QUBE vector

- Adds support for integer vectors
- QUBE vector includes syntax for
  - defining,
  - accessing and
  - manipulating integer vectors
Vector constructor $[e]$

- Defines a vector with elements $e$.
- If all elements evaluate to integers, the resulting vector is a value.

Vector selection $e.(e_i)$

- Selects the element at index $e_i$ from the vector $e$. 
Vector modification $e.(e_i) \leftarrow ee$

- The element at the index $e_i$ is replaced with the new element $ee$

Constant-value vector expression $vec\; en\; ee$

- defines a vector of non-constant length $en$ that only contains copies of the element $ee$
- If the integer $n$ is not negative and the element evaluates to a value $v$, the entire expression evaluates to a vector that contains $n$ copies of $v$.

```
let n = 1 + 3 in
vec n 0 => *[0,0,0,0]
```
QUBE array

- Adds support for multidimensional arrays and rank-generic programing

Array constructor \([ e : T][n] \)

- defines a multidimensional array with elements \(e\) of the element type \(T\) and shape \(n\) (and thus rank \(|n|\))

- The shape must be a natural number and the array elements must be equivalent to the product of the shape vector
QUBE arrays array constructor \([e : T][n]\)

The example shows the array modification \(e\)a.\([e] \leftarrow ee\)

\[
[1,2,3,4 :\text{int} | [2,2]].[[0,1]] \leftarrow 0 \Rightarrow [1,0,3,4:\text{int} | [2,2]]
\]
Conclusion
Introduction

MATLAB

APL

QUBE

Conclusion

- Powerful for solving numerical problems and big matrices operations
- Large sets of functions, which allow to avoid error-prone loops
- APL programs are typically very short programs
- MATLAB provides a large set of external libraries for every situation
- QUBE focuses on ruling out as many mistakes at compile time as possible
Array programming languages do not fit for every problem

APLs code is very cryptic

Fitting and user friendly graphic interfaces are rare

Array programming is very powerful combined with multicore processors and implicit parallelization
Any questions?