ParCA2: Architecture and Experiments

G.I.Malaschonok
Tambov State University
Internatsionalnaya, 33, 392000 Tambov, Russia
email: malschonok@ya.ru
MMCP'2009 – Dubna – JINR (07.07-11.07.09)

07.07.2009

Introduction (Short history of CAS)
1952

Computer algebra systems began to appear in the 1950-1960s. The paper of Lusternik L.A., Abramov A.A., Shestakov V.I. and Shura-Bura M.P. "Mathematical problem solving on automatic digital machines". Moscow, Acad. Sci. of the USSR, 1952. (Люстерник Л.А., Абрамов А.А., Шестаков В.И., Шура-Бура М.Р. Решение математических задач на автоматических цифровых машинах. М.: Изд-во АН СССР, 1952) was endeed the **first paper in the field of computer algebra**.

1963

The **first computer algebra system** was done by the **Martin Veltman**, who designed a program for symbolic manipulation of mathematical equations, especially for High Energy Physics, called Schoonschip (Dutch "clean ship") in **1963**.
Martinus J.G. Veltman was born in 1931 in Netherlands. He studying mathematics and physics at Utrecht University in 1948. He obtained his PhD in theoretical physics in 1963. During an extended stay at Stanford Linear Accelerator Center in 1963-64 he designed the computer program Schoonschip.

1971

In 1971, **Gerardus 't Hooft** was completing his PhD under the supervision of Veltman. They showed that if the symmetries of Yang-Mills theory were to be broken according to the Higgs method, then **Yang-Mills theory can be renormalized**. This result is one of the greats achievement of twentieth century physics. Veltman felt that he had done most of the preliminary work and written the program which made the dissertation possible. However, most of the credit went to 't Hooft. [http://www.uu.nl/uupublish/homeuu/homeenglish/aboututrechtuniv/corporate nobelprizewinner/veltman/23066main.html]

1999

Veltman received his credit later. He was awarded**the Nobel Prize for Physics in 1999** together with 't Hooft. They were honored "for elucidating the quantum structure of electroweak interactions in physics they placing this theory on a strong mathematical foundation due to the explicit calculation of physical quantities.

1964–

Using LISP as programming basis, **Carl Engelman created MATHLAB in 1964** at MITRE. Later MATHLAB was made available to users on PDP-6 and PDP-10. Today it can still be used on SIMH-Emulations of the PDP-10. MATHLAB ("mathematical laboratory") should not be confused with MATLAB ("matrix laboratory") which is a system for numerical computation built 15 years later.
The **first popular systems** were muMATH, Reduce, Derive and Macsyma; a copyleft version of Macsyma called Maxima is actively being maintained. The current **market leader is Mathematica and Maple**. The other well known computer algebra systems are Scratchpad, MuPAD, Magma, CoCoA.

1987–

The **era of calculator CAS** was started in 1987, when Hewlett-Packard introduced the first hand held calculator CAS. The Texas Instruments company in 1995 released the TI-92 calculator with an advanced CAS based on the software Derive. This, along with its successors (including the TI-89 series and the newer TI-Nspire CAS released in 2007) featured a reasonably capable and relatively inexpensive hand-held computer algebra system.

ParCA-1
2003

The project **Parallel Computer Algebra (ParCA)** have been started in **Tambov University** in 2003.
During the last 5 years we have obtained an extensive collection of useful classes, polynomial and matrix algorithms, etc.

We have done a lot of experiments on cluster machines:

- (1) a Myrinet-IBM machine of Tambov University (16 processors)
- (2) Russian Academy of Sciences Joint Supercomputer Center (4000-processors).

These experiments demonstrated:

- (a)the acceleration of matrix and polynomial multiplication operations more then 70% and
- (b)the acceleration of matrix operations of inversion family more then 50%.

We say that acceleration is equal 100% if the real time of some operation is n-times bigger for one machine than for the cluster of the n-machines.
These experiments let us understand that parallel computer algebra may become a realy efficient system.

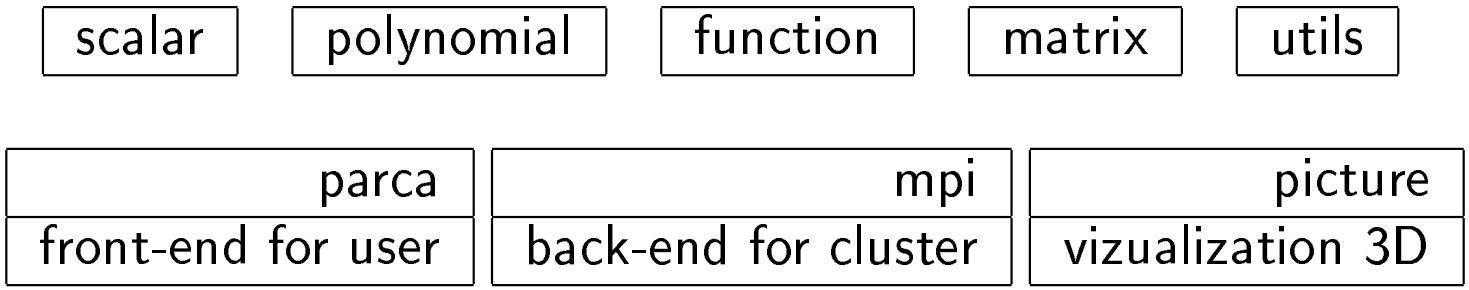
ParCA-2
2008

A new stage of the ParCA project is connected with **two problems.**

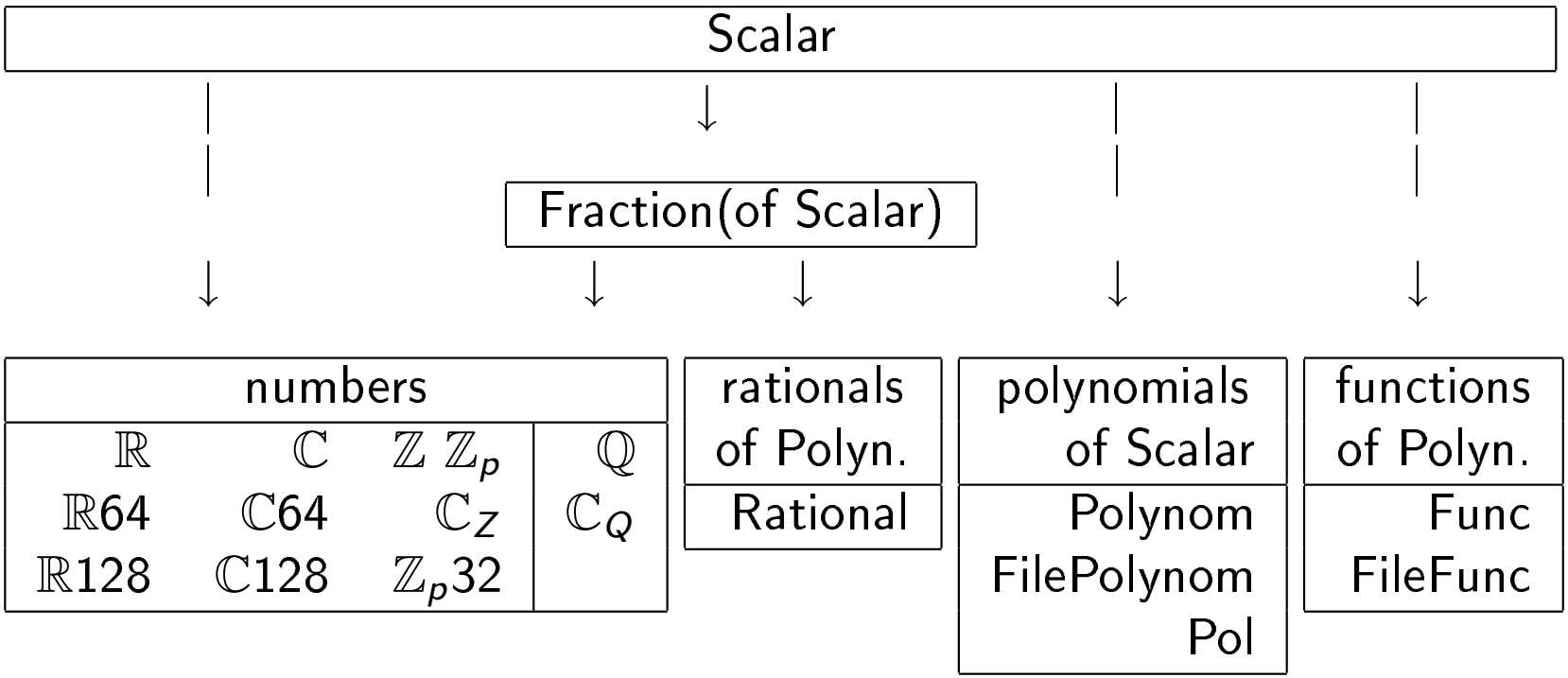
First problem is the problem of complicated class structure. We have to duplicate many algorithms for different number rings.
The second problem is an absent of large objects of data. Each object in ParCA-1 must be less than core memory.
That is why at the end of 2008 year we have started the ParCA-2 project.
Two main new features of this project are a

- **(1)** new classes structure a
- **(2)** new file-oriented data.

ParCA-2 packages



Class Scalar (abstract)



The abstract methods of the Scalar class cover all possible scalar operations :

add | subtract | multiply | divide | power | mod

modInverse | divideAndRemainder | GCD | extendedGCD

random | valueOf() | xNumberValue() | toString

Daughter classes of the Scalar class

are divided into several groups. There are:

numbers,
polynomials,
rational functions and
transcendental functions.

12 classes for numbers:

6 exact number classes and 6 rounded number classes.

The **exact sets** of numbers are: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{C}_Z$, $\mathbb{C}_Q$, $\mathbb{Z}_p$ and $\mathbb{Z}p32$.
We denote
$\mathbb{C}_Z = \{a + ib : a, b \in \mathbb{Z}, i^2 = -1\}$ and
$C_Q = \{a + ib : a, b \in Q, i^2 = -1\}$,
Zp=Z/pZ, with arbitrary prime number $p \in \mathbb{Z}$ and
$\mathbb{Z}p32 = \mathbb{Z}/p\mathbb{Z}$, with prime number p less than maximal positive number in the 4-bite word.
The last class of numbers is the most fast and the most often used in computer algebra applications.

The **rounded sets** of numbers are: $\mathbb{R}, \mathbb{C}, \mathbb{R}64, \mathbb{C}64, \mathbb{R}128$ and $\mathbb{C}128$.
Here we use the next notation. $\mathbb{R}64$ is a standard 64 bit-length floating point real number. $\mathbb{R}$ is an arbitrary-length real number. $\mathbb{R}128$ is a pare of a 64 bit-length floating point real number and an additional 64 bit-length word for an order. Three complex classes $\mathbb{C}$, $\mathbb{C}64$ and $\mathbb{C}128$ are obtained from the classes $\mathbb{R}$, $\mathbb{R}64$ and $\mathbb{R}128$, consequently.

Polynomial class of many variables is an only one for any of 12 number types.

The class Rational is a daughter class of Fraction. The numerator and denominator of rational element are polynomials. The other daughter class of Fractions is $\mathbb{Q}$-number class. The numerator and denominator of $\mathbb{Q}$-number are $\mathbb{Z}$-numbers.
The other daughter class of Scalar is the class of Functions. Class Function is obtained as a composition of elementary transcendental functions and polynomial functions.

Therefore the total **number of scalar classes is 60**: 12 numbers classes, 12 polynomial classes, 12 rational classes and 12 function classes.

We think that later the scalar class of infinite series may be done too.

Matrices

The **classes of Matrix, Vector and Tensor** have the Scalar type elements. So matrix method may be applied to any Scalar class.

The main **matrix methods** are:
"adjoint", "det", "kernel", "toEchelonForm", "inverse".
The main method is the recursive block method **"adjDet"**.

This method returns the matrix **determinant** together with **adjoint and echelon matrices**. It is easy to obtain the kernel matrix from echelon matrix and inverse matrix from adjoin matrix and determinant.

We have started to construct the classes of **file-matrices and file-polynomials**. We expect that with these new file-classes we will have possibility to solve problems with a very large input data.

Example
Parallel algorithm for polynomial multiplication

PolynMult (Polynom a, Polynom b, Polynom prod) // $a = a_1 + a_0$, $b = b_1 + b_0$
Input: a, b; // input polynomials
Output: prod // product of polynomials a and b.

```
    if (max(len(a),len(b))<m) then prod=a*b;
    else if (len(a)>len(b))
        then PolynMult(Polynom a1, Polynom b, Polynom prod1);
             PolynMult(Polynom a0, Polynom b, Polynom prod0);
        else PolynMult(Polynom a, Polynom b1, Polynom prod1);
             PolynMult(Polynom a, Polynom b0, Polynom prod0);
        prod=prod1+prod0;
end.
```