PART I

Numerical algorithms as a tool that unites scientific disciplines

Jerzy Respondek

The idea of computer simulation

They rely on **computer simulation** of the basic **laws of physics* - in the form of mathematical models -** to predict „in the computer" the development of various pchysical phenomena or behaviour of technical constructions.

It should be emphasized here that in many cases mathematical models of the phenomena have been known for centuries (e.g. the mentioned Newton's principles of dynamics).

However, earlier the scope of applications of mathematical methods in other fields of knowledge was relatively small due to the limited possibilities of available tools, and in particular the lack of sufficiently fast computers.

Example 1 - missile range

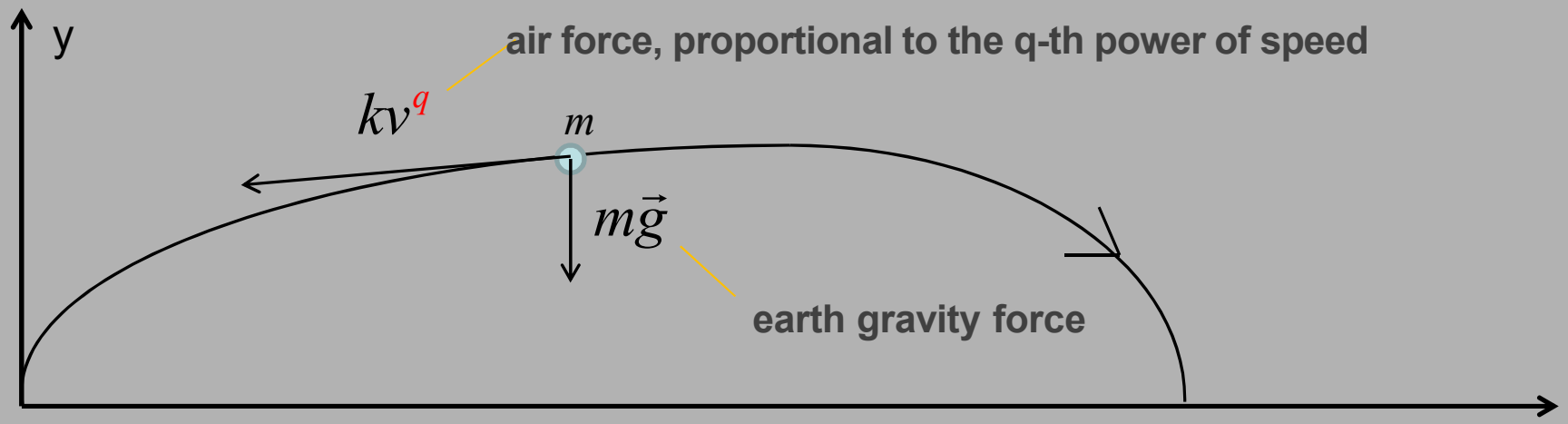
Without the use of computers, "on formulas", it is possible to calculate the range of missile range without taking into account air resistance. Proper formula:

$$L = \frac{V_0^2 \sin(2\alpha)}{g}$$

However, this formula is far too inaccurate for missile flights.
(i.e. gives results that are too different from reality)

- **The mathematical model, taking into account the air resistance, gives the next slide.**
- **We build it on the same physical basis, i.e. the Newton's second law.**

Example 1 - missile range cont.



General form of 2nd Newton's dynamics principle: $\vec{F} = m\vec{a}$

After taking into account the acting forces, and the fact that acceleration is the so-called derivative of speed:

$$m\vec{g} - kv^q \cdot \frac{\vec{v}}{|\vec{v}|} = m\frac{d\vec{v}}{dt}$$

- For *q=1* (air resistance force proportional to speed) this equation can still be solved without the help of a computer.

Example 1 - missile range cont.

$$m\vec{g} - kv^q \frac{\vec{v}}{|\vec{v}|} = m\frac{d\vec{v}}{dt}$$

- However, the reality is closer to the model of air resistance assuming that the resistance increases proportionally to the square of speed, i.e. $q=2$.
- However, for $q=2$, the solution to this equation requires the so-called numerical methods - using computers.

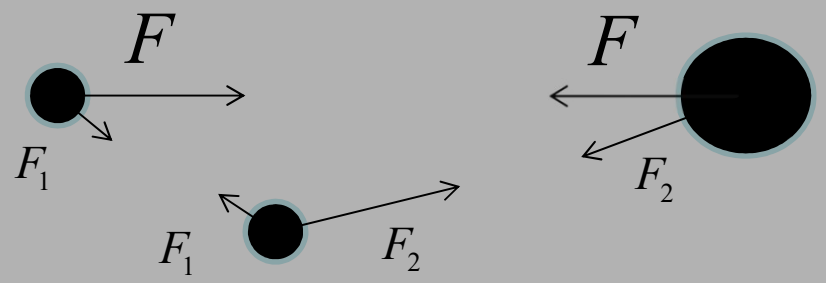
Thus, this typical example allows us to observe the general rule:

- **The laws of physics helpful, among others in predicting the range of missile flight, were formulated by Newton as early as in 17th century,**
- **however, computer science have significantly expanded the real range of applying these rights.**

Example 2 - N body problem

In the case of free movement of bodies in a vacuum **(e.g. stars), with the help of Newton's equation, their motion can be predicted for at most 2 bodies (!) without the use of computers.**

- **In the case where the number of bodies N >= 3, the problem can be solved only using a computer and approximate methods.**



Simulation at the molecular level

In place of Newton's principles of dynamics, in the "microworld" the behavior of particles is governed by the laws of quantum mechanics, in particular the Schrödinger equation (1940s).

However, at the time of its formulation, it could only be used for the simplest compounds, e.g. a hydrogen atom, due to its unique complexity.

This is a very limited application, since the laws of quantum mechanics "govern" the behavior of all chemical and biological compounds.

CHEMISTRY TEXTBOOKS OFTEN - TRADITIONALLY - TEACH "THE DIFFERENCES BETWEEN PHYSICAL PHENOMENA AND CHEMICAL TRANSFORMATION".

Other applications: COMPUTER GAMES

- Until quite recently, the realism of computer games was associated only with the quality of graphics.
- Modern games are characterized not only by high-quality graphics, but also by the artificial intelligence of computer opponents.
- Currently, more and more games go further, include the so-called "game physics".
- Thanks to this, e.g. sound waves and water behave as real, reacting to players' actions on an ongoing basis, and cars realistically deform in the event of collisions.

PART II

Synthesis methods of the fast matrix multiplication algorithms

Jerzy Respondek

The general aims of this lecture is:

- To present how the matrix multiplication algorithms domain evolved, creating a separate branch of so-called **Fast Matrix Multiplication**

- The naive, definition-based implementation gives ***O(n³)*** complexity:

$$C = AB \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \qquad i = 1,\ldots,m, \quad j = 1,\ldots,l$$

- For rectangular matrices we have ***m x l*** elements, for each element ***n*** multiplications are performed inside the sum, thus we need (***m x l x n***)
- For square matrices ***m = l = n***, thus **n^3** multiplications are needed (in case of multiplying by the mathematical definition)

The first attempt to make the matrix multiplication faster

- The first step in this direction was made **no by Strassen**.
- Winograd (1968) proposed a modified algorithm for calculation the **inner product of vectors**, but simultaneously showed how to apply that new algorithm to matrix multiplication decreasing the number of scalar multiplications to be performed.

- For two vectors ***x*** and ***y*** he proposed to calculate the inner products in two steps. In the first step we calculate auxiliary numbers:

$$\xi = \sum_{i=1}^{\lfloor n/2 \rfloor} x_{2i-1} x_{2i}, \qquad \eta = \sum_{i=1}^{\lfloor n/2 \rfloor} y_{2i-1} y_{2i}$$

Next, in the second step, with the use of previously calculated auxiliary numbers, the inner product is given by

$$x^T y = \begin{cases} x_{even}^T y_{even} = \left[\sum_{i=1}^{\lfloor n/2 \rfloor} (x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - \xi - \eta \right], & for\ even\ n \\ x_{even}^T y_{even} + x_n y_n, & for\ odd\ n \end{cases}$$

- By aggregating the products by pairs, he decreased the number of scalar multiplications roughly by half.

Winograd decreased the number of multiplications to multiply matrices, but he did not changed the asymtotic complexity, which is still ***O(n³)*** for both the definition based algorithm as well as its Winograd modification

The (very) famous Strassen result

- Strassen proposed to divide matrices into 4 sub-matrices:

$$C = AB, \qquad \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]_C = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]_A \cdot \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]_B$$

- Next we calculate **7** auxiliary products:

$$m_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$m_2 = (A_{21} + A_{22})B_{11}$$
$$m_3 = A_{11}(B_{12} - B_{22})$$
$$m_4 = A_{22}(B_{21} - B_{11})$$
$$m_5 = (A_{11} + A_{12})B_{22}$$
$$m_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$m_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

- The result ***C=AB*** is:

$$\left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]_C = \left[\begin{array}{c|c} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ \hline m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{array}\right]$$

We apply this method recurvively.

- Time complexity:

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \qquad O\left(n^{\log_2 7}\right) \approx O\left(n^{2.807}\right)$$

Is this result the best possible for 2x2 matrix size ?

- Hopcroft and Kerr (1969) proved that at least just **7** multiplications are necessary to multiply 2x2 matrices in any non-commutative* algorithm.
- Winograd (1971) proved that even with use of commutativity* of the matrix entries the minimal multiplication number still is **7** for 2x2 problem.

- But Probert (1976) showed that at least **15** additions are necessary**.
- While the Strassen's scheme utilizes **18** additions/subtractions.

(*) About that property later.

(**) For scheme with 7 multiplications;
definitional-based algorithm requires obviously merely 4.

Optimal scheme with 15 +/- :
Winograd (1973) modification of Strassen scheme

- 3 series of auxiliary term
- Important is the order of (and in) the 3 series

$$
\begin{aligned}
S_1 &= A_{21} + A_{22,} \\
S_2 &= S_1 - A_{11}, \\
S_3 &= A_{11} - A_{21}, \\
S_4 &= A_{12} - S_2, \\
S_5 &= B_{12} - B_{11}, \\
S_6 &= B_{22} - S_5, \\
S_7 &= B_{22} - B_{12}, \\
S_8 &= S_6 - B_{21}
\end{aligned}
\qquad
\begin{aligned}
M_1 &= S_2 S_6, \\
M_2 &= A_{11} B_{11}, \\
M_3 &= A_{12} B_{21}, \\
M_4 &= S_3 S_7, \\
M_5 &= S_1 S_5, \\
M_6 &= S_4 B_{22}, \\
M_7 &= A_{22} S_8,
\end{aligned}
\qquad
\begin{aligned}
T_1 &= M_1 + M_2 \\
T_2 &= T_1 + M_4 \\
T_3 &= T_1 + M_5
\end{aligned}
$$

The final product ***C= AB*** have the form:

$$
C = \left[\begin{array}{c|c} M_2 + M_3 & T_3 + M_6 \\ \hline T_2 - M_7 & T_2 + M_5 \end{array} \right]
$$

Observation:

This scheme for 2x2 in 7 multiplications is less convenient for the parallelization.

Commutativity

- The fast matrix multiplication algorithms domain distinguishes two main classes of algorithms, i.e. **commutative** and **non-commutative.**

- The first **commutative** class requires the elements of the matrices belong to a commutative field with respect to multiplication (e.g. real numbers).
- It causes that the **commutative** matrix multiplication algorithms cannot be applied recursively.
- That follows from the fact, that during the recursion we come to scalars at the last recursion level, but on the previous levels we encounter just matrices.
- As we know for matrices in general ***AB<>BA***.

The 3x3 case

- After Strassen discovery there was a pursue to make the matrix multiplication even faster
- One of the most promissing „base” size is 3x3, even up to now (!).
- Definition gives $\mathbf{3^3=27}$ scalar multilications.
- The Winograd inner product gives **24** *, but requires commutativity
- Gastinel (1970) proposed a generalized Strassen formula for any size, which for 3x3 gives non-commutative algorithm with **25** multiplications.
- Musinski (1973) reduced this number to **24**.
- Brocket and Dobkin proposed a commutative algorithm with **23** multiplications, but it required additional * and / by 2.

- **Ladermann** (1975) proposed a non-commutative algorithm with **23** multiplications.
- **It is up to now (!) best possible algorithm for the 3x3 size.**

The 3x3 case - Ladermann (1976) construction – non-commutative case

- The 23 aggregated product have the form:

$$\begin{cases} m_1 = (a_{11}+a_{12}+a_{13}-a_{21}-a_{22}-a_{32}-a_{33})b_{22} \\ m_2 = (a_{11}-a_{21})(-b_{12}+b_{22}) \\ m_3 = a_{22}(-b_{11}+b_{12}+b_{21}-b_{22}-b_{23}-b_{31}+b_{33}) \\ m_4 = (-a_{11}+a_{21}+a_{22})(b_{11}-b_{12}+b_{22}) \\ m_5 = (a_{21}+a_{22})(-b_{11}+b_{12}) \\ m_6 = a_{11}b_{11} \\ m_7 = (-a_{11}+a_{31}+a_{32})(b_{11}-b_{13}+b_{23}) \\ m_8 = (-a_{11}+a_{31})(b_{13}-b_{23}) \\ m_9 = (a_{31}+a_{32})(-b_{11}+b_{13}) \\ m_{10} = (a_{11}+a_{12}+a_{13}-a_{22}-a_{23}-a_{31}-a_{32})b_{23} \\ m_{11} = a_{32}(-b_{11}+b_{13}+b_{21}-b_{22}-b_{23}-b_{31}+b_{32}) \\ m_{12} = (-a_{13}+a_{32}+a_{33})(b_{22}+b_{31}-b_{32}) \end{cases}$$

$$\begin{cases} m_{13} = (a_{13}-a_{33})(b_{22}-b_{32}) \\ m_{14} = a_{13}b_{31} \\ m_{15} = (a_{32}+a_{33})(-b_{31}+b_{32}) \\ m_{16} = (-a_{13}+a_{22}+a_{23})(b_{23}+b_{31}-b_{33}) \\ m_{17} = (a_{13}-a_{23})(b_{23}-b_{33}) \\ m_{18} = (a_{22}+a_{23})(-b_{31}+b_{33}) \\ m_{19} = a_{12}b_{21} \\ m_{20} = a_{23}b_{32} \\ m_{21} = a_{21}b_{13} \\ m_{22} = a_{31}b_{12} \\ m_{23} = a_{33}b_{33} \end{cases}$$

- Linearly combined as follows:

$$\begin{cases} c_{11} = m_6 + m_{14} + m_{19} \\ c_{12} = m_1 + m_4 + m_5 + m_6 + m_{12} + m_{14} + m_{15} \\ c_{13} = m_6 + m_7 + m_9 + m_{10} + m_{14} + m_{16} + m_{18} \\ c_{21} = m_2 + m_3 + m_4 + m_6 + m_{14} + m_{16} + m_{17} \\ c_{22} = m_2 + m_4 + m_5 + m_6 + m_{20} \end{cases}$$

$$\begin{cases} c_{23} = m_{14} + m_{16} + m_{17} + m_{18} + m_{21} \\ c_{31} = m_6 + m_7 + m_8 + m_{11} + m_{12} + m_{13} + m_{14} \\ c_{32} = m_{12} + m_{13} + m_{14} + m_{15} + m_{22} \\ c_{33} = m_6 + m_7 + m_8 + m_9 + m_{23} \end{cases}$$

Brent diophantine equation

Brent in his report (19705) proposed to define aggregated products in the general form:

$$m_p = \left(\sum_{i_1=1}^{m} \sum_{i_2=1}^{n} \alpha_{i_1 i_2}^{(p)} a_{i_1 i_2} \right) \left(\sum_{j_1=1}^{n} \sum_{j_2=1}^{l} \beta_{j_1 j_2}^{(p)} b_{j_1 j_2} \right), \quad p = 1, \ldots, T \qquad c_{k_1 k_2} = \sum_{p=1}^{T} \gamma_{k_1 k_2}^{(p)} m_p$$

where **a_*, b_*.** are elements of the matrices to be multiplied

The result of the sum must adhere to definitional

(where we deliberately emphasized “1” coefficient)

$$c_{k_1 k_2} = \sum_{k} 1 \cdot \left(a_{k_1 k} b_{k k_2} \right)$$

- Most desired are those constructions, which requires that all those coefficients are from the set {-1,0,1} , like the Strassen construction is.
- Some valuable constructions for 3x3 case, e.g. Brockett and Dobkin (1976, 1978), requires also multiplying or division by 2, though.

The 3x3 case: Brent diophantine equation (cont.)

This equation explains also Strassen construction, but – unfortunately – very quickly leads to galactic size problems:

- for 2x2 we have 64 equations with 84 unknowns
- for 3x3 we obtain 729 equations and 621 unknowns:

$$\sum_{t=1}^{23} \alpha_{ijp}\beta_{ijp}\gamma_{mnp} = \delta_{mi}\delta_{jk}\delta_{ln}\,, \qquad i,j,k,m,n,p = 1,2,3$$

- Laderman claims that solved this equations with no computer usage, but did not disclosed how he did that.
- But the results is correct, with **23** auxiliary $\boldsymbol{m_i}$ terms

- Up to now the Laderman's result is not beaten for ***3x3***.
- Blaser (2003) showed that at least **19** *-s are necessary for ***3x3*** size.
- Since $\log_2 7 \sim \log_3 21.8$ to improve Strassen result we need algorithm with at most **21** multiplications.
- The solution obained by other approach on next slides.

Brent diophantine equation solution for Strassen's algorithm

Let us remind the Strassen's scheme of:

- aggregated products $\mathbf{m}_*$: - linear combinations: $\Sigma\gamma_*\mathbf{m}_*$; γ in {-1,0,1}

$$\begin{cases} m_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \\ m_2 = (A_{21} + A_{22})B_{11} \\ m_3 = A_{11}(B_{12} - B_{22}) \\ m_4 = A_{22}(B_{21} - B_{11}) \\ m_5 = (A_{11} + A_{12})B_{22} \\ m_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \\ m_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \end{cases}$$

$$\left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]_C = \left[\begin{array}{c|c} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ \hline m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{array}\right]$$

- The Strassen algorithm can be represented in form of 3 series,
- Each consisting of of seven 2x2 matrices:

$$A^{(p)}_{1\le p\le 7} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}\right)$$

$$B^{(p)}_{1\le p\le 7} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}\right)$$

$$\Gamma^{(p)}_{1\le p\le 7} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\right)$$

Arbitrary precision algorithms (APA)

- Intelligible will be to present this notion by the example, for 3x3.

- Schonhage (1980) proposed the following **21** auxiliary products:

$$u_{ii} = (a_{i1} + \varepsilon^2 a_{i2})(\varepsilon^2 b_{1i} + b_{2i}), \quad u_{ij} = (a_{i1} + \varepsilon^2 a_{j2})(b_{2i} - \varepsilon b_{1i}) \qquad (i \neq j)$$
$$v_{ii} = (a_{i1} + \varepsilon^2 a_{i3}) b_{3i}, \qquad v_{ij} = (a_{i1} + \varepsilon^2 a_{j3})(b_{3i} + \varepsilon b_{1i}) \qquad (i \neq j)$$
$$w_i = a_{i1}(b_{2i} + b_{3i})$$

And the entries of ***D=AB*** matrix are given by:

$$d_{ij} = \frac{1}{\varepsilon^2}(u_{ij} + v_{ij} - w_i) + \frac{1}{\varepsilon}(v_{ji} - v_{jj})$$

where ε is an real, scalar, additional arbitrary parameter.

Arbitrary precision algorithms (APA) (cont.)

Now one can easy check:

$$D = AB + \varepsilon R$$

So for $\varepsilon \rightarrow 0$ the result ***D*** tends to the exact value od the desired product **AB.**

Unfortunately, considering formula:

$$d_{ij} = \frac{1}{\varepsilon^2}(u_{ij} + v_{ij} - w_i) + \frac{1}{\varepsilon}(v_{ji} - v_{jj})$$

it is <u>impossible</u>, to simple state ε***=0***.

So we obtain so-called arbitrary precision algorithms:

- the result is arbitrary close to the exact value, with decreasing ε, but never is exact.
- In favour we multiply **3x3** matrix by **21** multiplications, giving exponent $\log_3 21 \sim 2.771$, better than Strassen's **2.807**.

What if we still need exact algorithm (EC) ?

With help comes the method Bini (1979) how to construct the **exact** algorithm (EC) on the base of **arbitrary precision algorithm** (APA):

- We find the highest degree of ε, let's say ***d***
- Calculate the desired matrix product **D=AB** **(d+1)** times, for different ε parameter values (but now they do not need be small (!))
- Find the desired product ***D*** as an linear combinations of **(d+1)** matrices, but with its coefficients (α_i) parameters choosen in a way, which removes positive powers of ε :

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \varepsilon_1 & \varepsilon_2 & \cdots & \varepsilon_{d+1} \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon_1^d & \varepsilon_2^d & \cdots & \varepsilon_{d+1}^d \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{d+1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Its another application of Vandermonde matrix and its inversion algorithms.

Relation between exact (EC) and Arbitraty Precision Matrix Multiplication (APA)

We rearrange the above sum with respect to the consecutive powers of ε_* :

$$S = (\alpha_1 + \alpha_2 + \alpha_3)AB + (\alpha_1\varepsilon_1 + \alpha_2\varepsilon_2 + \alpha_3\varepsilon_3)R_1 + (\alpha_1\varepsilon_1^2 + \alpha_2\varepsilon_2^2 + \alpha_3\varepsilon_3^2)R_2$$

What about time efficiency?

- It can be prooved Romani (1979) that if the APA algorithm is of the class:

$$O(APA(n)) = O(n^{\omega})$$

- So the corresponding exact algorithm builded by the Vandermonde composition posesses class:

$$O(EC(n)) = O(\log n \cdot n^{\omega})$$

Methods of algorithm construction for not-square matrices:

What is going on ?

We explain it by the example:

- Let us assume that we have a fast formula to multiply matrix of the size **(6x4) by (4x2)***

  (*) Mathematically, the inner dimensions must be compatiple

- We want o yield from that fast algorithm, to multiply matrices of the size **(2x4) by (4x6)**.
- The solution is given by the matrix equality

  (transposition time is neeglegible - quadratic)

$$AB = (B^T A^T)^T$$

This way we tacke with the source problem by corresponding one:

$$(2\times 4)\cdot(4\times 6) \Rightarrow (6\times 4)\cdot(4\times 2)$$

Methods for not-square matrices: (cont.)

Let us denote the matrix dimensions by:

$$(a \times b) \cdot (b \times c) \;\hat{=}\; (a,b,c)$$

There is available fully elegant universal method by Hopcroft, Musinski (1973) how on the base of fast algorithm for a given one series of dimensions, with < ***T*** > multiplications:

$$(a,b,c)$$

construct algorithms with the same < ***T*** > number of scalar multiplications,

for all the possible permutations of the dimensions, i.e.(*):

(a,c,b)
(b,a,c)
(b,c,a)
(c,a,b)
(c,b,a)

Hopcroft, Musinski (1973) *Duality applied to matrix multiplication,* SIAM J on Computing, Vol.2 (3) pp. 159-173.

(*) So-called permutation rule, or duality of matrix multiplication

Bini's results for 12x12 matrices – the partial matrix multiplication notion

- We use: APA algorithms, permutation rule and multiplicative rule.
- For an exact algorithm, the following, so-called, **partial matrix multiplication,** for matrices of the form:

$$Z = XY = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & 0 \end{bmatrix} \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$$

requires **6** scalar multiplications.

- Bini et al. (1979) showed, how to do that in **5** scalar multiplications, but in an **arbitrary-precision way** (APA).

The **5** aggregated products proposed by Bini have the following form:

$$\begin{cases} m_1 = (x_{21} + \varepsilon x_{11}) y_{11} \\ m_2 = x_{12} y_{12} \\ m_3 = x_{21} (y_{11} + y_{12} + \varepsilon y_{21}) \\ m_4 = (x_{12} + x_{21})(y_{12} + \varepsilon y_{21}) \\ m_5 = (x_{12} + \varepsilon x_{11})(y_{12} + \varepsilon y_{22}) \end{cases}$$

Result for 12x12 matrices – the partial matrix multiplication notion (cont).

- We linearly combine them as follows:

$$\begin{cases} z_{11} = (m_1 - m_2 - m_3 + m_4)/\varepsilon \\ z_{21} = m_1 \\ z_{12} = (m_5 - m_2)/\varepsilon \\ z_{22} = -m_2 + m_4 \end{cases}$$

as result we obtain the approximating formula with **d=1** degree correction terms:

$$Z \approx XY + \varepsilon \begin{bmatrix} 0 & x_{11}y_{22} \\ x_{11}y_{11} & x_{12}y_{21} + x_{21}y_{21} \end{bmatrix}$$

By the symmetry we can really easy obtain the corresponding formulas for the product of the form:

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & 0 \end{bmatrix} \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & x'_{12} \\ x'_{21} & x'_{22} \end{bmatrix} \begin{bmatrix} y'_{11} & y'_{12} \\ y'_{21} & y'_{22} \end{bmatrix}$$

Result for 12x12 matrices – the partial matrix multiplication notion (cont).

- Now we can combine the two symmetric versions:

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & 0 \end{bmatrix}\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \quad AND \quad \begin{bmatrix} 0 & x'_{12} \\ x'_{21} & x'_{22} \end{bmatrix}\begin{bmatrix} y'_{11} & y'_{12} \\ y'_{21} & y'_{22} \end{bmatrix}$$

like in the diagram:

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x'_{12} \\ x'_{21} & x'_{22} \end{bmatrix}\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} + z'_{11} & z_{22} + z'_{12} \\ z'_{21} & z'_{22} \end{bmatrix}$$

Thus, by combining two symmetric versions of the same partial matrix multiplication of the size **(2x2)x(2x2)**, with zero one entry, we obtained non-partial algorithm for the problem by **10** multiplications, with gain equal **2** multiplication over the definitional **12** multiplications.

Bini's results for 12x12 matrices (cont.)

- By the permutation rule we obtain from **(3,2,2)** algorithm for **(2,3,2)** and **(2,2,3)** **(**Hopcroft, Musinski (1973)).
- Next we use the multiplication rule, obtaining problem:

(3*2*2, 2*3*2, 2*2*3) = (12,12,12)

The cost is **10x10x10=1000**, so the complexity exponent:

$$\log_{12} 1000 = 2.7799$$

Results for rectangular matrices

- Hopcroft & Kerr (1971) generalized Strassen's result to ***(px2)x(2xn)*** product in – roughly – **75%** scalar products amount.
- On that basis Probert (1974) presented algorithm for **(4x2)x(2x4)** in **26** multiplications, instead of „definitional-based" **32** (81%).
- Hopcroft & Musinski (1973) considered case **(3x2)x(2x3)** in **15** multiplications (83% of 3*2*3=18)

Pursue for other dimensions

- For **n=3** since the Laderman result (1976) (**23** *-s) for exact and Schonhage (1980) (**21** *-s) for approximate **no progress is** (for above 40 years), despite many endeavours.
- For **n=4** the best solution is to use the Strassen result twice recusively, in **7*7=49** scalar multiplications.

A good insight into the effords made in the pursue gives 5x5 case:

- Fidducia (1972) – **115** multiplications (**definition: 5^3=125**)
- Fisher (1974) – **110**
- Sykora (1977) – **109**
- Sykora (1977) – **105**
- Schachtel (1978) – **103**
- Makarov (1986) – **102**
- Makarov (1987) – **100**
- Sedoglavic (2017) – **99** (after above 30 years!)
- Sedoglavic (2019) – **98**

Pursue for other dimensions

- We have also general methods, for arbitrary dimension **nxn** matrices:

- Gastinel (1971) – **n^3-(n-1)** multiplications (**definition: n^3**)
- Sykora (1977) – **n^3-$(n-1)^2$** multiplications

- Sykora's algorithm as one of the main tool uses Latin squares

- Laderman's algorithm for 3x3 matrix in 23 multiplications is a special case of Sykora's algorithm.
- Winograd's algorithm multiplying 2x2 matrix in 7 multiplications and 15 additions/subtractions also is a special case of Sykora's algorithm.
- Strassen's algorithm is a special case of Gastinel's algorithm.

Actual state

But there works are of less practical meaning, due to large constants in complexity.

[] Coppersmith D., Winograd S. **(1990)** *„Matrix multiplication via arithmetic progressions”*, Journal of Symbolic Computations

$$O\left(n^{2.375477}\right)$$

[] Stothers Andrew (**2010**) *„On the Complexity of Matrix Multiplication”*.

$$O\left(n^{2.373}\right)$$

[] Le Gall François **(2014)** *„Powers of tensors and fast matrix multiplication”*.

$$O\left(n^{2.3728}\right)$$

PART III

Applications of fast matrix multiplication algorithms

Jerzy Respondek

EXAMPLE OF APPLICATION

Matrix inverting can be implemented in no worse time than multiplication

Proof :(***A*** is symmetrical & positive)

Let us – *temporarily* - assume that the *A* matrix, which we want to invert, is:

- ***dim A= n*x*n***, where ***n=2^q***
- **1) symmetrical 2) positively defined**

The algorithm sketch:

we divide the given ***A*** *n*x*n* matrix to four $(n/2)x(n/2)$ sub-matrices:

$$A = \begin{pmatrix} B & C^T \\ C & D \end{pmatrix}$$

Next we define the ***A's*** Schur residual with respect to the ***B*** matrix:

$$S = D - CB^{-1}C^T$$

The inversion we construct by known block matrix algebra rules:

$$A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^T S^{-1} CB^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix} \qquad S = D - CB^{-1}C^T$$

So we need to obtain four matrix terms:

1. CB^{-1}
2. $(CB^{-1})C^T$
3. $S^{-1}(CB^{-1})$
4. $(CB^{-1})^T (S^{-1}CB^{-1})$

We distinguished by the respective colors the common sub-terms

So ***nxn*** matrix inverting may be performed by:

- four $(n/2)$x$(n/2)$ multiplication
- two $(n/2)$x$(n/2)$ inverting of the matrices ***B*** and ***S***.

So the **matrix inverting algorithm can be implemented in a recusive way by inverting the respective two matrices of the half size in time:**

$$I(n) \le 2I(n/2) + 4M(n) + O(n^2)$$

where ***I(n)*** is the inverting complexity, and ***M(n)*** is the multiplication time complexity

Finally: $I(n) = O(M(n))$ Q.E.D.

What if the *A* matrix is does <u>not</u> fulfill the assumptions of:

1. symmetricity
2. positive definition

?

We utylize the known matrix equality:

$$A^{-1} = (A^T A)^{-1} A^T$$

Known property:

For any non-singular ***A*** matrix, the matrix :

$$\boldsymbol{A^T A}$$

is:

- non-singular,
- **<u>positive defined,</u>**
- **<u>symmetric</u>**

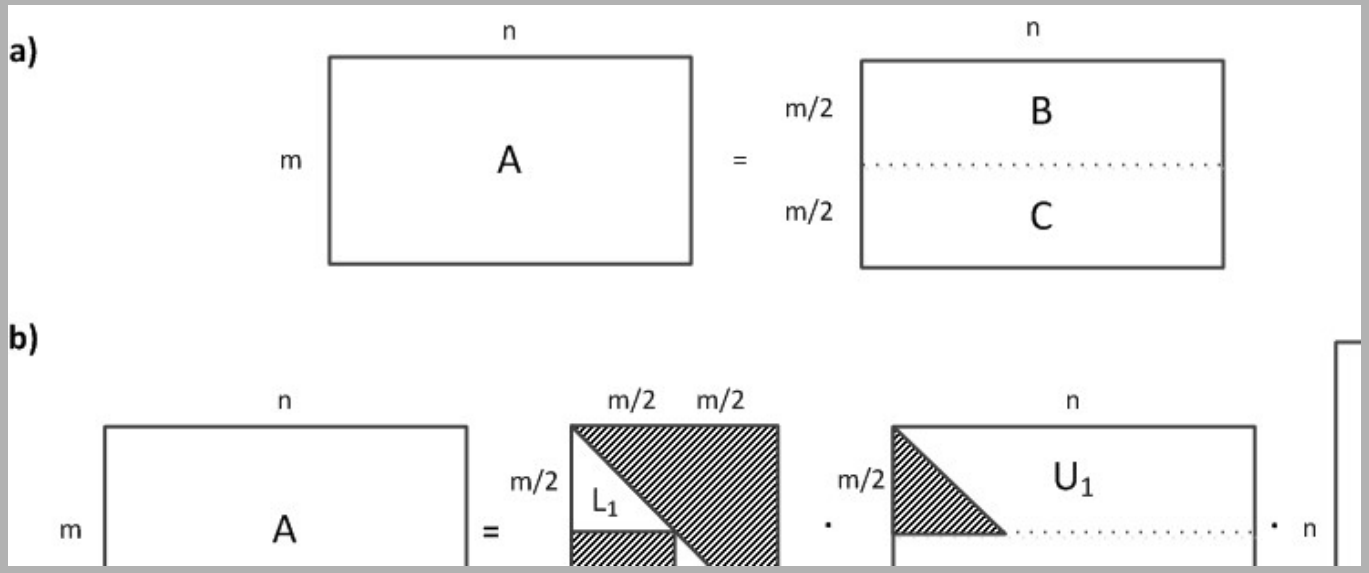
Another example: LU decomposition in no worse time, than the available matrix multiplication algorithm

- We use the following recursive algorithm:

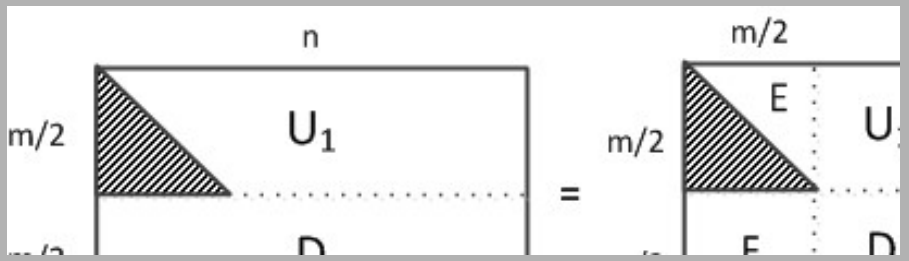
Step 1: divide **A** matrix into **(m/2)xn** sub matrices **B** and **C**

Step 2: call **FACTOR(B, m/2, n)**

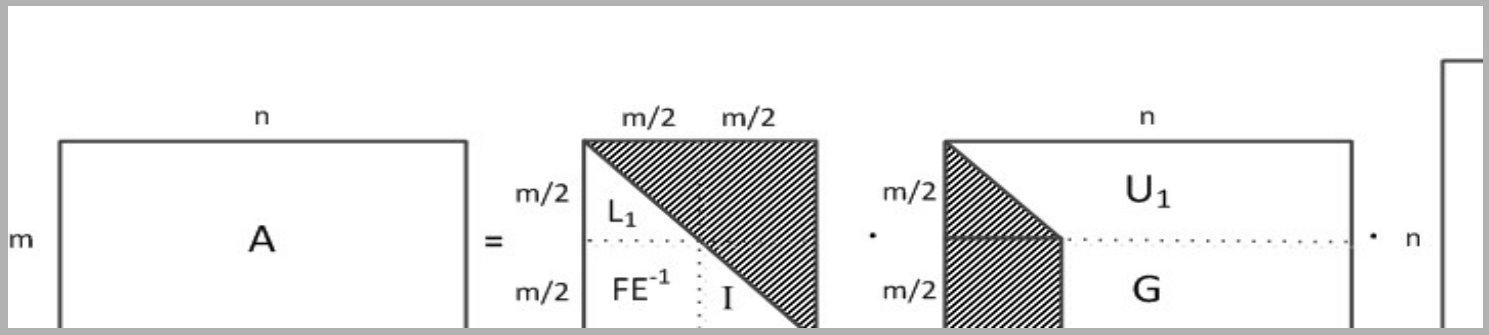
- We obtain:



Let **E** and **F** be the matrices built of the leading **m/2** columns of the $\mathbf{U_1}$ and **D** matrices, respectively



Step 3: $G:=D-FE^{-1}U_1$



Step 4: **call FACTOR(G', m/2, n-m/2)** (**G'** is the right part of **G**)

LU decomposition is now ready with time complexity equal to utilized matrix multiplication (e.g. 2.807 for Strassen).

Bunch J., Hopcroft J.: *Triangular factorization and inversion by fast matrix multiplication*. Mathematics of Computation, Vol. 28 (125), pp. 231÷236, 1974.

A survey of applications

Solving the system of linear equations Ax=b

There are two methods:

A)

- **invert a matrix**
- **calculate $A^{-1}X$**

B)

- **Decompose to LU**
- **Solve the *LUx=b* systems in two steps, in each of the triangular case.**

Determinant calculation

- **Decompose given matrix to LU**
- **Next use the property, usable especially in triangular case, in linear time:**

$$\det[A] = \det[L] \cdot \det[U] \cdot \det[P]$$

Polynomial calculations:

- Borodin (1975) shows how to evaluate a polynomial at a large number of points at once by matrix multiplication in the same time

Rank of the matrix and related problems:

- Ibarra et. al. (1982) proposed a generalized LUP decomposition, here called the SQP decomposition, applicable also to matrices without the full row rank.
- The results of the article can be also used to calculate the rank of a matrix together with the respective non-singular minor in matrix multiplication time.

Characteristic polynomial of a matrix:

Keller-Gehrig (1985) proposed three algorithms to calculate the matrix characteristic polynomial coefficients, with different generality and efficiency:

- Alg. 1 works in ***O(logn*M(n))*** time, but is applicable only in special cases of the input matrix.
- Alg. 2 is the fastest, achieving ***O(M(n))*** time, i.e. the same as the used matrix multiplication algorithm, but is not fully general, either.
- Alg. 3 works in ***O(logn*M(n))*** time, but works for any square input matrix.

Problems related to Krylov matrix:

- Dumas (2005) proposes an ***O(logn*M(n))*** time algorithm to find a minimal polynomial of a Krylov matrix.
- Bini (1994) showed how to calculate the Krylov matrix in ***O(logn*M(n))*** time.

Formal grammar problems

- Valiant (1975) applies matrix black-boxed paradigm to achieve ***M(n)*** time in the problem of context-free recognition.

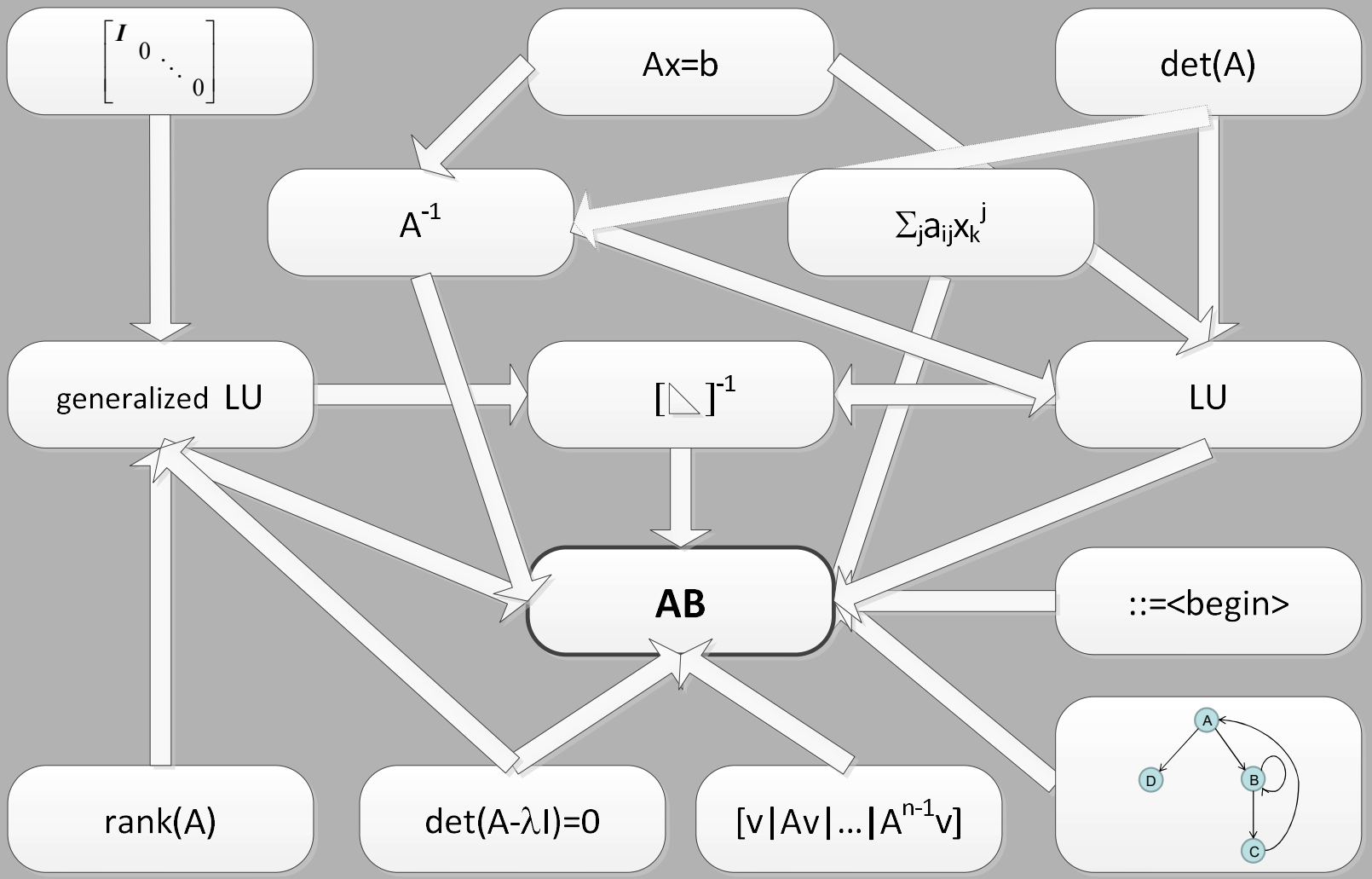
Compiler construction

- For the efficient memory management it is important which of the compiled program functions are recursive.
- That can be detected by calculating the so-called transitive closure of a graph, which is expressed as a certain matrix polynomial.
- Munro (1971) shows how to utilize in this problem the matrix multiplication in a recursive way, obtaining an ***O(M(n))*** time algorithm.

Graph paths problems:

- Seidel (1992) computes the shortest distances between all pairs of vertices of an undirected, unweighted graph in ***O(logn*M(n))*** time.
- Alon et. al. (1991) generalizes these results to the case when the weights are between ***0*** and ***B*** with ***O(B^2*logn*M(n))*** time.

In the scientists community working on the matrix multiplication algorithm there is a popular hypothesis that it is possible to multiply the matrices in $O(n^{2+\varepsilon})$ time, for any (!) positive ε.



Dependency graph of the

black-boxed matrix multiplication algorithms

Other issues connected with the application of fast matrix multiplication

- In practice, the importance of surveyed in this lecture applications of the FMM algorithms are even higher.

The following architectures:

- **ARM v9**
- **x86 –** so-called AMX commands

are introducing just the matrix multiplication operation on the hardware level.

Thus it is important to have algorithms, which can yield from the MM in other operations, or even branches of computer science.

Other issues connected with the application of fast matrix multiplication

- In practice, the implementation of any fast matrix multiplication algorithm is definitely a non-trivial task.
- It must consider the hardware capabilities of a given computer system, especially parallelization and vectorization issues, cache and memory management.
- Nonetheless, the quite recent work from 2016 on Strassen's algorithm:

[] Huang J., Smith T.M., Henry G.M., Geijn R.A.: *Strassen's Algorithm Reloaded*. Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, USA, 2016.

Proved that fast matrix multiplication algorithms can be efficiently implemented:

- even for small matrices,
- not quite square
- for multi-core architecture.

The generic matrix algorithms on the matrix multiplication

REMARK

- As we can see, involving both mathematical and algorithm construction methods often enables to improve the asympthotical class of the algorithm
- The above mentioned, well-known algorithms operate on an arbitrary form matrices
- The square matrix contains of N^2 elements, so the theoretical efficiency limit is $O(N^2)$, though such an effectiveness is impossible to achieve for the arbitrary form matrices.*

SOLUTION

- Better efficiency is attainable by designing the **special** algorithms for a **selected** classes of matrices.

- The following can be seen as the desired future research directions:

- **Construction of the parallel algorithm for inverting the confluent Vandermonde matrices.**
- **Adaptation of the algorithm to the vector-oriented hardware units, like Intel AVX.**
- **Combination of both above.**
- **CUDA implementation.**

Remark: **The N-body problem already has followed all this directions!**

In the special matrices currently there are emerging first works, e.g.:

[] Pedro Alonso, Miguel O. Bernabéu, Victor M. García, Antonio M. Vidal, *Implementation and tuning of a parallel symmetric Toeplitz eigensolver*, **Journal of Parallel and Distributed Computing**, 71 (3) (2011), pp. 485-494.

Very thanks for Your attention!