OPTIMIZATION TECHNIQUES FOR SMALL MATRIX MULTIPLICATION

Charles Éric Drevet[†], Md. Nazrul Islam[⋆] and Éric Schost[⋆]

[†]École Polytechnique, Palaiseau, France

[⋆]Ontario Research Centre for Computer Algebra, The University of Western Ontario, London, Canada

Overview

Our goal: improving the number of multiplications for small matrix product, by combining previous results in an optimal manner.

Why?

- small matrices with large entries: evaluation of holonomic functions, Padé Hermite, Hensel lifting for polynomial systems
- actual complexity still poorly understood

Base cases (over a non-commutative ring)

dimension	author	# multiplications
$(2\times 2\times 2)$	Strassen (1969) Winograd (1971)	7
$(3\times 3\times 3)$	Laderman (1976)	23
$(5\times 5\times 5)$	Makarov (1987)	100
$(a\times 2\times c)$	Hopcroft-Kerr (1971)	$(3ac+\max(a,c))/2$
$(a\times b\times c)$ $(b\times c\times a)$	pair of products Pan (1984)	$abc+ab+bc+ac$
$(a\times a\times a)$ a even	trilinear aggregating (TA) Pan (1982, 84); $(\star)$ is new	$(a^3+12a^2+11a)/3$ $(\star)$ $(a^3+11.25a^2+32a+27)/3$
$(a\times a\times a)$ a odd	slight improvement on Pan's results	$(a^3+15a^2+14a-6)/3$

Base cases (over a commutative ring)

dimension	author	# multiplications
$(3\times 3\times 3)$	Makarov (1986)	22
$(a\times b\times c)$ b even	Waksman (1970)	$b(ac+a+c-1)/2$
$(a\times b\times c)$ b odd	Waksman (1970)	$(b-1)(ac+a+c-1)/2+ac$

Previous work

- Probert and Fischer (1980): square sizes up to 40.
- Smith (2002): rectangular sizes up to $(28\times 28\times 28)$
- Mezzarobba (2007): commutative case, square sizes up to 28.

Improved Padding

Main idea: When using algorithms recursively, peeling and padding techniques are used. We can avoid some useless products by exploiting "sparsity" (of the algorithm).

Example: To multiply two square matrices A, B of size 3 with Strassen's algorithm, we pad them in size 4, and subdivide as

$$\tilde{A}_{1,1}=\begin{bmatrix} a_{1,1} & a_{1,2}\\ a_{2,1} & a_{2,2}\end{bmatrix},\ \tilde{A}_{1,2}=\begin{bmatrix} a_{1,3} & 0\\ a_{2,3} & 0\end{bmatrix},\ \tilde{A}_{2,1}=\begin{bmatrix} a_{3,1} & a_{3,2}\\ 0 & 0\end{bmatrix},\ \tilde{A}_{2,2}=\begin{bmatrix} a_{3,3} & 0\\ 0 & 0\end{bmatrix}$$

The 7 recursive products are

$$\gamma_1=\tilde{A}_{2,2}(\tilde{B}_{2,1}-\tilde{B}_{1,1})\quad \gamma_2=(\tilde{A}_{2,1}-\tilde{A}_{1,1})(\tilde{B}_{1,1}+\tilde{B}_{1,2})\quad \dots$$

and the product $\tilde{C}=\tilde{A}\tilde{B}$ is

$$\begin{bmatrix} \tilde{C}_{1,1}=\gamma_1+\gamma_6+\gamma_7-\gamma_4 & \tilde{C}_{1,2}=\gamma_5+\gamma_4\\ \tilde{C}_{2,1}=\gamma_1+\gamma_3 & \tilde{C}_{2,2}=\gamma_5-\gamma_3+\gamma_2+\gamma_7. \end{bmatrix}$$

Naive remark: $\tilde{A}_{2,2}$ has one row and one column full of zeros, so 2 multiplications suffice for γ_1.

Less naive: γ_2 is used only to compute $\tilde{C}_{2,2}$, and $\tilde{C}_{2,2}$ has only one non-zero term, so 2 multiplications suffice.

Total: 25 products.

To make this automatic: describe an algorithm by three sequences of matrices. For Strassen's algorithm, we get

	(linear combination of A's entries)	(linear combination of B's entries)	(where it is used in C)
γ_1 :	$\begin{bmatrix} 0 & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix} -1 & 0\\ 1 & 0\end{bmatrix}$	$\begin{bmatrix} 1 & 1\\ 0 & 0\end{bmatrix}$
γ_2 :	...	...	...

Exploiting the zeros in these matrices, we can reduce the sizes of recursive calls.

Building a table

We start from a database of algorithms

- `Strassen`, `Winograd`, `Laderman`, ...
- `Winograd2`, obtained by applying a symmetry of `Winograd`
- `mul211`, `mul121` and `mul112`, which describe product in sizes $(2\times 1\times 1)$, $(1\times 2\times 1)$, $(1\times 1\times 2)$

Then, for a given size:

- try all algorithms, various subdivision schemes
- for a given algorithm, and a given subdivision, determine the size of the recursive calls, and look up their cost
- other techniques: pairing products, TA, ...

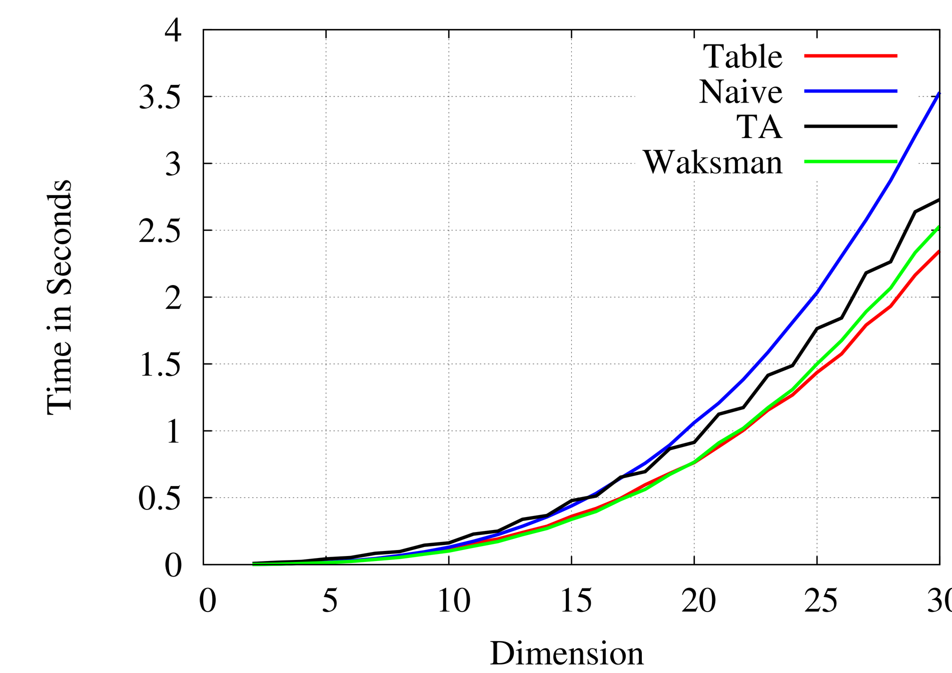
Implementation: proof-of-concept, few optimizations

- based on GMP (integers) and NTL (polynomials)
- commutative base rings: $\mathbb{Z}$, $\mathbb{F}_p[x]$
- non-commutative base rings: differential operations and linear recurrences with polynomial coefficients

Results

	commutative			non commutative		
dim.	# mul.	algorithm	previous	# mul.	algorithm	previous
9	**472**	`mul121`	473	**522**	`mul121`	527
11	**825**	`mul121`	831	**923**	`Strassen`	992
13	**1318**	`mul121`	1333	**1450**	`Strassen`	1580
14	**1525**	`mul121`	1561	**1728**	`Strassen`	1743
15	**1941**	`mul121`	2003	**2108**	`Winograd2`	2300
18	**3060**	`Hopcroft`	3231	**3306**	`TA`	3342
20	**4158**	`Strassen`	4165	**4340**	`TA`	4380
21	**4938**	`Strassen`	5261	**5365**	`Strassen`	5610
22	**5440**	`mul121`	5610	**5566**	`TA`	5610
23	**6382**	`Hopcroft`	6843	**6806**	`TA`	7048
24	**6900**	`Hopcroft`	6909	**7000**	`TA`	7048
25	**8083**	`mul121`	8710	**8448**	`TA`	8710
26	**8658**	`TA`	8710	**8658**	`TA`	8710
27	**9994**	`mul121`	10612	**10330**	`TA`	10612
28	**10556**	`TA`	10612	**10556**	`TA`	10612

Timings with polynomials of degree 100 over $\mathbb{F}_{9001}$ as entries:



Timings with differential operators of order 10, with polynomial coefficients of degree 10 over $\mathbb{F}_{9001}$ as entries: