Fast Matrix Product Algorithms: From Theory To Practice

Thomas Sibut-Pinote
Inria, École Polytechnique, France

Éric Schost
University of Waterloo, ON,Canada

November 2nd, 2015

Motivation

- Complexity of matrix product $\Rightarrow$ complexity of linear algebra;
- $\omega = \mathit{inf}\ \{\theta \mid \text{ it takes } n^\theta \text{ operations to multiply in } \mathcal{M}_n(\mathbb{K})\} \in [2,3]$;
- Strassen '69 : $\omega < 2.81$ (used in practice);
- Le Gall '14 : $\omega < 2.3728639$ (theoretical).

Motivation

- Complexity of matrix product $\Rightarrow$ complexity of linear algebra;
- $\omega = inf \left\{\theta \mid \text{ it takes } n^\theta \text{ operations to multiply in } \mathcal{M}_n(\mathbb{K})\right\} \in [2, 3]$;
- Strassen '69 : $\omega < 2.81$ (used in practice);
- Le Gall '14 : $\omega < 2.3728639$ (theoretical).

- **Can we bridge the gap a little?**

Problem Statement

Let $\langle m, n, p \rangle$ denote the bilinear map:

$$\begin{aligned} \mathcal{M}_{m,n}(\mathbb{K}) \times \mathcal{M}_{n,p}(\mathbb{K}) &\longrightarrow \mathcal{M}_{m,p}(\mathbb{K}) \\ (A, B) &\mapsto A \cdot B. \end{aligned}$$

Goal: determine the arithmetic complexity of $\langle m, n, p \rangle$.

Problem Statement

Let $\langle m, n, p \rangle$ denote the bilinear map:

$$\mathcal{M}_{m,n}(\mathbb{K}) \times \mathcal{M}_{n,p}(\mathbb{K}) \longrightarrow \mathcal{M}_{m,p}(\mathbb{K})$$
$$(A, B) \mapsto A \cdot B.$$

Goal: determine the arithmetic complexity of $\langle m, n, p \rangle$.

Known: naive algorithm in *mnp* operations:

$$\forall i \in [\![1, m]\!], \forall j \in [\![1, p]\!],\ [AB]_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Problem Statement

Let $\langle m, n, p \rangle$ denote the bilinear map:

$$\begin{aligned} \mathcal{M}_{m,n}(\mathbb{K}) \times \mathcal{M}_{n,p}(\mathbb{K}) &\longrightarrow \mathcal{M}_{m,p}(\mathbb{K}) \\ (A, B) &\mapsto A \cdot B. \end{aligned}$$

Goal: determine the arithmetic complexity of $\langle m, n, p \rangle$.

Known: naive algorithm in *mnp* operations:

$$\forall i \in [\![1, m]\!], \forall j \in [\![1, p]\!],\ [AB]_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}.$$

Can we do better?

Strassen's Algorithm

Strassen's algorithm: $\langle 2,2,2\rangle$ in 7 multiplications (instead of $2\cdot 2\cdot 2 = 8$):

$$
\begin{aligned}
\alpha_1 &= (a_{1,2} - a_{2,2}), & \beta_1 &= (b_{2,1} + b_{2,2}), & p_1 &= \alpha_1\beta_1 \\
\alpha_2 &= (a_{2,1} - a_{1,1}), & \beta_2 &= (b_{1,2} + b_{1,1}), & p_2 &= \alpha_2\beta_2 \\
\alpha_3 &= a_{1,1}, & \beta_3 &= (b_{1,2} - b_{2,2}), & p_3 &= \alpha_3\beta_3 \\
\alpha_4 &= a_{2,2}, & \beta_4 &= (b_{2,1} - b_{1,1}), & p_4 &= \alpha_4\beta_4 \\
\alpha_5 &= (a_{2,1} + a_{2,2}), & \beta_5 &= b_{1,1}, & p_5 &= \alpha_5\beta_5 \\
\alpha_6 &= (a_{1,2} + a_{1,1}), & \beta_6 &= b_{2,2}, & p_6 &= \alpha_6\beta_6 \\
\alpha_7 &= (a_{1,1} + a_{2,2}), & \beta_7 &= (b_{1,1} + b_{2,2}), & p_7 &= \alpha_7\beta_7
\end{aligned}
$$

Strassen's Algorithm

Strassen's algorithm: $\langle 2,2,2\rangle$ in 7 multiplications (instead of $2 \cdot 2 \cdot 2 = 8$):

$$
\begin{array}{lll}
\alpha_1 = (a_{1,2} - a_{2,2}), & \beta_1 = (b_{2,1} + b_{2,2}), & p_1 = \alpha_1\beta_1 \\
\alpha_2 = (a_{2,1} - a_{1,1}), & \beta_2 = (b_{1,2} + b_{1,1}), & p_2 = \alpha_2\beta_2 \\
\alpha_3 = a_{1,1}, & \beta_3 = (b_{1,2} - b_{2,2}), & p_3 = \alpha_3\beta_3 \\
\alpha_4 = a_{2,2}, & \beta_4 = (b_{2,1} - b_{1,1}), & p_4 = \alpha_4\beta_4 \\
\alpha_5 = (a_{2,1} + a_{2,2}), & \beta_5 = b_{1,1}, & p_5 = \alpha_5\beta_5 \\
\alpha_6 = (a_{1,2} + a_{1,1}), & \beta_6 = b_{2,2}, & p_6 = \alpha_6\beta_6 \\
\alpha_7 = (a_{1,1} + a_{2,2}), & \beta_7 = (b_{1,1} + b_{2,2}), & p_7 = \alpha_7\beta_7
\end{array}
$$

$$c_{1,1} = p_1 + p_4 - p_6$$

$$c_{1,2} = p_4 + p_5$$

$$c_{2,1} = p_3 + p_6$$

$$c_{2,2} = p_2 + p_3 - p_5 + p_7$$

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix}$$

Strassen's Algorithm

Strassen's algorithm: $\langle 2,2,2\rangle$ in 7 multiplications (instead of $2\cdot 2\cdot 2 = 8$):

$$\begin{array}{lll}
\alpha_1 = (a_{1,2} - a_{2,2}), & \beta_1 = (b_{2,1} + b_{2,2}), & p_1 = \alpha_1\beta_1 \\
\alpha_2 = (a_{2,1} - a_{1,1}), & \beta_2 = (b_{1,2} + b_{1,1}), & p_2 = \alpha_2\beta_2 \\
\alpha_3 = a_{1,1}, & \beta_3 = (b_{1,2} - b_{2,2}), & p_3 = \alpha_3\beta_3 \\
\alpha_4 = a_{2,2}, & \beta_4 = (b_{2,1} - b_{1,1}), & p_4 = \alpha_4\beta_4 \\
\alpha_5 = (a_{2,1} + a_{2,2}), & \beta_5 = b_{1,1}, & p_5 = \alpha_5\beta_5 \\
\alpha_6 = (a_{1,2} + a_{1,1}), & \beta_6 = b_{2,2}, & p_6 = \alpha_6\beta_6 \\
\alpha_7 = (a_{1,1} + a_{2,2}), & \beta_7 = (b_{1,1} + b_{2,2}), & p_7 = \alpha_7\beta_7
\end{array}$$

$$c_{1,1} = p_1 + p_4 - p_6$$
$$c_{1,2} = p_4 + p_5$$
$$c_{2,1} = p_3 + p_6$$
$$c_{2,2} = p_2 + p_3 - p_5 + p_7$$
$$C = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix}$$

Observe:

$$C = p_1\gamma_1 + p_2\gamma_2 + p_3\gamma_3 + p_4\gamma_4 + p_5\gamma_5 + p_6\gamma_6 + p_7\gamma_7.$$

where

$$\gamma_1 = E_{1,1}, \quad \gamma_2 = E_{2,2}, \quad \gamma_3 = E_{2,1} + E_{2,2}, \quad \gamma_4 = E_{1,1} + E_{1,2},$$
$$\gamma_5 = E_{1,2} - E_{2,2}, \quad \gamma_6 = E_{2,1} - E_{2,2}, \quad \gamma_7 = E_{2,2} \qquad E_{i,j} \textit{ canonical basis}$$

Strassen's Algorithm

Strassen's algorithm: $\langle 2,2,2\rangle$ in 7 multiplications (instead of $2\cdot 2\cdot 2 = 8$):

$$\begin{array}{lll}
\alpha_1 = (a_{1,2} - a_{2,2}), & \beta_1 = (b_{2,1} + b_{2,2}), & p_1 = \alpha_1\beta_1 \\
\alpha_2 = (a_{2,1} - a_{1,1}), & \beta_2 = (b_{1,2} + b_{1,1}), & p_2 = \alpha_2\beta_2 \\
\alpha_3 = a_{1,1}, & \beta_3 = (b_{1,2} - b_{2,2}), & p_3 = \alpha_3\beta_3 \\
\alpha_4 = a_{2,2}, & \beta_4 = (b_{2,1} - b_{1,1}), & p_4 = \alpha_4\beta_4 \\
\alpha_5 = (a_{2,1} + a_{2,2}), & \beta_5 = b_{1,1}, & p_5 = \alpha_5\beta_5 \\
\alpha_6 = (a_{1,2} + a_{1,1}), & \beta_6 = b_{2,2}, & p_6 = \alpha_6\beta_6 \\
\alpha_7 = (a_{1,1} + a_{2,2}), & \beta_7 = (b_{1,1} + b_{2,2}), & p_7 = \alpha_7\beta_7
\end{array}$$

$$c_{1,1} = p_1 + p_4 - p_6$$
$$c_{1,2} = p_4 + p_5$$
$$c_{2,1} = p_3 + p_6$$
$$c_{2,2} = p_2 + p_3 - p_5 + p_7$$
$$C = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix}$$

Observe:

$$C = p_1\gamma_1 + p_2\gamma_2 + p_3\gamma_3 + p_4\gamma_4 + p_5\gamma_5 + p_6\gamma_6 + p_7\gamma_7.$$

where

$$\gamma_1 = E_{1,1}, \quad \gamma_2 = E_{2,2}, \quad \gamma_3 = E_{2,1} + E_{2,2}, \quad \gamma_4 = E_{1,1} + E_{1,2},$$
$$\gamma_5 = E_{1,2} - E_{2,2}, \quad \gamma_6 = E_{2,1} - E_{2,2}, \quad \gamma_7 = E_{2,2} \qquad E_{i,j} \textit{ canonical basis}$$

Tensor notation: $\sum_{i=1}^{7} \alpha_i \otimes \beta_i \otimes \gamma_i.$

Tensors and algorithms

General tensor notation identified with a bilinear map:

$$\langle m, n, p\rangle = \sum_{i=1}^{m} \sum_{j=1}^{p} \sum_{k=1}^{n} a_{i,k} \otimes b_{k,j} \otimes c_{i,j}.$$

Representing $\langle m, n, p\rangle$ as $\sum_{i=1}^{r} \alpha_i \otimes \beta_i \otimes \gamma_i$ gives an **algorithm**.

Example: The elementary tensor $(a_{1,2} + a_{3,5}) \otimes b_{2,4} \otimes (c_{1,4} + c_{2,4})$ reads as the algorithm

$$\begin{aligned} tmp &\leftarrow (a_{1,2} + a_{3,5}) \cdot b_{2,4} \\ c_{1,4} &\leftarrow tmp \\ c_{2,4} &\leftarrow tmp \end{aligned}$$

Composition

$t \otimes t'$: computes the composition of two tensors.

To multiply A of size (mm', nn') by B of size (nn', pp'), decompose A and B into blocks:

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{1,1} & \cdots & B_{1,p} \\ \vdots & & \vdots \\ B_{n,1} & \cdots & B_{n,p} \end{bmatrix}$$

where $A_{i,j}$ of size (m', n'), $B_{j,k}$ of size (n', p').

If $t = \langle m, n, p \rangle$ and $t' = \langle m', n', p' \rangle$:

$$t \otimes t' \simeq \langle mm', nn', pp' \rangle.$$

Composition

$t \otimes t'$: computes the composition of two tensors.

To multiply A of size (mm', nn') by B of size (nn', pp'), decompose A and B into blocks:

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{1,1} & \cdots & B_{1,p} \\ \vdots & & \vdots \\ B_{n,1} & \cdots & B_{n,p} \end{bmatrix}$$

where $A_{i,j}$ of size (m', n'), $B_{j,k}$ of size (n', p').

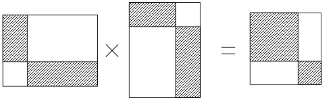
If $t = \langle m, n, p \rangle$ and $t' = \langle m', n', p' \rangle$:

$$t \otimes t' \simeq \langle mm', nn', pp' \rangle.$$

Also set $t^{\otimes k} = \underbrace{t \otimes t \otimes \cdots \otimes t}_{k \text{ times}} \simeq \langle m^k, n^k, p^k \rangle$.

Direct Sum of Tensors

$t \oplus t'$: **computes two independent matrix products in parallel.**



We will denote $s \odot t$ for $\underbrace{t \oplus t \oplus \cdots \oplus t}_{s \text{ times}}$.

Rank and ω

Definition (Rank of a Tensor t)

$$R(t) := \min \left\{ r \mid t \text{ can be written as } \sum_{i=1}^{r} x_i \otimes y_i \otimes z_i \right\}$$

Rank and ω

Definition (Rank of a Tensor t)

$$R(t) := \min \left\{ r \mid t \text{ can be written as } \sum_{i=1}^{r} x_i \otimes y_i \otimes z_i \right\}$$

$R(\langle m, n, p \rangle)$ is the minimal number of multiplications for $\langle m, n, p \rangle$.

Rank and ω

Definition (Rank of a Tensor t)

$$R(t) := \min \left\{ r \mid t \text{ can be written as } \sum_{i=1}^{r} x_i \otimes y_i \otimes z_i \right\}$$

$R(\langle m, n, p \rangle)$ is the minimal number of multiplications for $\langle m, n, p \rangle$.

Definition (Linear Algebra Exponent)

$$\omega := \inf\{\tau \mid \text{ There exists an algorithm to multiply } n \times n \text{ matrices in } \mathcal{O}(n^{\tau}) \text{ additions and multiplications}\} (\in [2, 3])$$

Rank and ω

Definition (Rank of a Tensor t)

$$R(t) := \min \left\{ r \mid t \text{ can be written as } \sum_{i=1}^{r} x_i \otimes y_i \otimes z_i \right\}$$

$R(\langle m, n, p \rangle)$ is the minimal number of multiplications for $\langle m, n, p \rangle$.

Definition (Linear Algebra Exponent)

$$\omega := \inf\{\tau \mid \text{ There exists an algorithm to multiply } n \times n \text{ matrices in } \mathcal{O}(n^\tau) \text{ additions and multiplications}\} (\in [2, 3])$$

Theorem

$$\inf\{\tau \mid R(\langle n, n, n \rangle) = \mathcal{O}(n^\tau)\} = \omega$$

Back to Strassen's Algorithm

Theorem (Strassen '69)

$R(\langle 2,2,2\rangle) \leq 7$, *hence* $\omega \leq \log_2(7) \simeq 2.81$.

Idea: $R\left(\langle 2^k, 2^k, 2^k\rangle\right) \leq 7^k$ by induction on k.
Cut into blocks of size 2^{k-1} and proceed recursively.

Back to Strassen's Algorithm

Theorem (Strassen '69)

$R(\langle 2,2,2\rangle) \leq 7$, *hence* $\omega \leq \log_2(7) \simeq 2.81$.

Idea: $R\left(\langle 2^k, 2^k, 2^k\rangle\right) \leq 7^k$ by induction on k.
Cut into blocks of size 2^{k-1} and proceed recursively.

Lemma

$R(\langle m,n,p\rangle) \leq r \Rightarrow R(\langle mnp, mnp, mnp\rangle) \leq r^3$.

Idea: If we can do $\langle m,n,p\rangle$ in r operations, then we can obtain $\langle n,p,m\rangle$ and $\langle p,m,n\rangle$ in r operations. Then we compose them.

Back to Strassen's Algorithm

Theorem (Strassen '69)

$R(\langle 2,2,2\rangle) \leq 7$, *hence* $\omega \leq \log_2(7) \simeq 2.81$.

Idea: $R\left(\langle 2^k, 2^k, 2^k\rangle\right) \leq 7^k$ by induction on k.
Cut into blocks of size 2^{k-1} and proceed recursively.

Lemma

$R(\langle m,n,p\rangle) \leq r \Rightarrow R(\langle mnp, mnp, mnp\rangle) \leq r^3$.

Idea: If we can do $\langle m,n,p\rangle$ in r operations, then we can obtain $\langle n,p,m\rangle$ and $\langle p,m,n\rangle$ in r operations. Then we compose them.

Theorem

$$R(\langle m,n,p\rangle) \leq r \Rightarrow \omega \leq \frac{3\log(r)}{\log(mnp)}.$$

- $R(\langle mnp, mnp, mnp\rangle) \leq r^3$;
- Proceed recursively for $\langle (mnp)^k, (mnp)^k, (mnp)^k\rangle$ just like for the $\langle 2,2,2\rangle$ case.

Bini's Approximate Algorithms ('79)

Idea: $K \rightsquigarrow K[\varepsilon]$

Definition (degenerate rank of a tensor t)

$$\underline{R}(t) := \min\{r \mid \exists \underline{t}(\varepsilon), \quad \underline{t}(\varepsilon) = \sum_{i=1}^{r} u_i(\varepsilon) \otimes v_i(\varepsilon) \otimes w_i(\varepsilon)$$
$$\text{with } \underline{t}(\varepsilon) = \varepsilon^{q-1} t + \varepsilon^q t_1(\varepsilon) \text{ and } q > 0\}.$$

Algorithmically, one can obtain t by computing $t(\varepsilon)$ modulo ε^q.

Bini's Approximate Algorithms ('79)

Idea: $K \rightsquigarrow K[\varepsilon]$

Definition (degenerate rank of a tensor t)

$$\underline{R}(t) := \min\{r \mid \exists \underline{t}(\varepsilon), \quad \underline{t}(\varepsilon) = \sum_{i=1}^{r} u_i(\varepsilon) \otimes v_i(\varepsilon) \otimes w_i(\varepsilon)$$
$$\text{with } \underline{t}(\varepsilon) = \varepsilon^{q-1} t + \varepsilon^q t_1(\varepsilon) \text{ and } q > 0\}.$$

Algorithmically, one can obtain t by computing $\underline{t}(\varepsilon)$ modulo ε^q.

Theorem (Bini '79)

$$\underline{R}(\langle m, n, p \rangle) \leq r \Rightarrow \omega \leq \frac{3 \log(r)}{\log(mnp)}$$

Consequence: $\omega < 2.79$.

The τ-theorem

Theorem (τ-theorem, Schönhage '81)

If

$$\underline{R}\left(\bigoplus_{i=1}^{s} \langle m_i, n_i, p_i \rangle\right) \leq r,$$

and

$$\sum_{i=1}^{s} (m_i\, n_i\, p_i)^{\beta} = r,$$

then

$$\omega \leq 3\,\beta.$$

Consequence (Schönhage again): $\omega < 2.55$.

Crucial for recent records (including Le Gall '14: $\omega < 2.37287$)

Towards a Practical Use of the τ-Theorem

Theoretical Obstacles

- The τ-theorem gives great bounds on ω but it is not seen as a way to build 'concrete' matrix product algorithms (non-effective proofs).
- 'Degenerate rank $\Leftrightarrow$ rank' relies on the fact that computing with polynomials is asymptotically negligible.

Theoretical Contributions

- More constructive proof of the τ-theorem (an algorithm).
- Get rid of ε **and** use the τ-theorem constructively! (for specific kinds of tensors)

Sketch of the constructive proof

Suppose $t(\varepsilon)$ is a degeneration of $\bigoplus_{i=1}^{s} \langle m_i, n_i, p_i \rangle$.

$$\left(\bigoplus_{i=1}^{s} \langle m_i, n_i, p_i \rangle \right)^{\otimes k} \approx \bigoplus_{\substack{\mu = (\mu_1, \cdots, \mu_s) \\ \mu_1 + \cdots + \mu_s = k}} \text{(several matrix products } (M, N, P))$$

Sketch of the constructive proof

Suppose $t(\varepsilon)$ is a degeneration of $\bigoplus_{i=1}^{s} \langle m_i, n_i, p_i \rangle$.

$$\left(\bigoplus_{i=1}^{s} \langle m_i, n_i, p_i \rangle \right)^{\otimes k} \approx \bigoplus_{\substack{\mu=(\mu_1,\cdots,\mu_s)\\ \mu_1+\cdots+\mu_s=k}} \underbrace{\binom{k}{\mu_1,\cdots,\mu_s} \odot \left\langle \underbrace{\prod_{i=1}^{s} m_i^{\mu_i}}_{M}, \underbrace{\prod_{i=1}^{s} n_i^{\mu_i}}_{N}, \underbrace{\prod_{i=1}^{s} p_i^{\mu_i}}_{P} \right\rangle}_{\binom{k}{\mu} \text{ matrix products } (M,N,P) \text{ in parallel}}$$

Sketch of the constructive proof

Suppose $t(\varepsilon)$ is a degeneration of $\bigoplus_{i=1}^{s} \langle m_i, n_i, p_i \rangle$.

$$\left(\bigoplus_{i=1}^{s} \langle m_i, n_i, p_i \rangle \right)^{\otimes k} \approx \bigoplus_{\substack{\mu=(\mu_1,\cdots,\mu_s) \\ \mu_1+\cdots+\mu_s=k}} \underbrace{\binom{k}{\mu_1, \cdots, \mu_s} \odot \left\langle \underbrace{\prod_{i=1}^{s} m_i^{\mu_i}}_{M}, \underbrace{\prod_{i=1}^{s} n_i^{\mu_i}}_{N}, \underbrace{\prod_{i=1}^{s} p_i^{\mu_i}}_{P} \right\rangle}_{\binom{k}{\mu} \text{ matrix products } (M, N, P) \text{ in parallel}}$$

In the same way,

$$t(\varepsilon)^{\otimes k} \simeq \bigoplus_{\mu} t_\mu(\varepsilon).$$

1. Choose $t_\mu(\varepsilon) \Rightarrow$ we can do $\binom{k}{\mu}$ $\langle M, N, P \rangle$ matrix products in parallel effectively (with ε's).
2. Compute $\langle M^l, N^l, P^l \rangle = \langle M^{l-1}, N^{l-1}, P^{l-1} \rangle \otimes \langle M, N, P \rangle$ recursively like previously, using t_μ to gain operations at each stage.

Pan's aggregation tables ('84)

Builds a family of tensors computing independent matrix products to improve ω:

- Input: table with various tensors. Example:

$\sum_{i=0}^{m-1}\sum_{k=0}^{p-1} x_{i,0} \otimes y_{0,k} \otimes \varepsilon^2 z_{k,i}$	$\langle m, 1, p \rangle$
$\sum_{i=0}^{m-1}\sum_{k=0}^{p-1} \varepsilon u_{0,k,i} \otimes \varepsilon v_{k,i,0} \otimes w_{0,0}$	$\langle 1, (m-1)(p-1), 1 \rangle$

- Every row gives a matrix product (actually, some variables to adjust);
- Aggregate terms by summing over columns,

here: $t = \sum_{i=0}^{m-1}\sum_{k=0}^{p-1} (x_{i,0} + \varepsilon u_{0,k,i}) \otimes (y_{0,k} + \varepsilon v_{k,i,0}) \otimes (\varepsilon^2 z_{k,i} + w_{0,0})$.

$$t = \varepsilon^2 \left(\langle m, 1, p \rangle \oplus \langle 1, (m-1)(p-1), 1 \rangle\right) + t_2$$

Correction term

$$t = \sum_{i=0}^{m-1} \sum_{k=0}^{p-1} (x_{i,0} + \varepsilon u_{0,k,i}) \otimes (y_{0,k} + \varepsilon v_{k,i,0}) \otimes (\varepsilon^2 z_{k,i} + w_{0,0})$$

To apply the τ-theorem we want:

$$t \; = \; \varepsilon^2 (\langle m, 1, p\rangle \oplus \langle 1, (m-1)(p-1), 1\rangle) + \text{terms of higher degree in } \varepsilon$$

Let us remove terms of degree 0 and 1, hence the corrected term:

$$t_1 = t - \left(\sum_{i=0}^{m-1} x_{i,0}\right) \otimes \left(\sum_{k=0}^{p-1} y_{0,k}\right) \otimes w_{0,0}.$$

We get the output:

$$t_1 = \varepsilon^2 (\langle m, 1, p\rangle \oplus \langle 1, (m-1)(p-1), 1\rangle) + \varepsilon^3 t_2$$

Hence $\underline{R}(\langle m, 1, p\rangle \oplus \langle 1, (m-1)(p-1), 1\rangle) \leq mp + 1$.

Consequence: $\omega < 2.55$ with $m = 4, p = 4$.

Combined use with the τ-theorem

Every matrix variable appears with the same degree in ε: homogenous tensor.

Theorem (S,S-P '12)

Let t be a homogenous tensor.
If we apply the algorithm of the constructive proof of the τ-theorem to t, for any μ and $k > 1$, the resulting tensor $t_\mu(\varepsilon)$ can be written as

$$t_\mu(\varepsilon) = \varepsilon^q t_1,$$

where t_1 does not contain any ε.

Consequence

Set $\varepsilon = 1$ in $t_\mu(\varepsilon)$: get an ε-free tensor computing disjoint matrix products.
Even better: set $\varepsilon = 1$ in $t(\varepsilon)$ **before** extracting t_μ from $t(\varepsilon)^{\otimes k}$.

We can get rid of the ε while still benefiting from the τ-theorem!

Example

Example: $2 \odot \langle 4, 9, 4 \rangle$ in 243 multiplications (instead of $2 \cdot (4 \cdot 9 \cdot 4) = 288$) with:

$$t_1 = \sum_{i=0}^{m-1} \sum_{k=0}^{p-1} (x_{i,0} + \varepsilon u_{0,k,i}) \otimes (y_{0,k} + \varepsilon v_{k,i,0}) \otimes (\varepsilon^2 z_{k,i} + w_{0,0})$$
$$- \left(\sum_{i=0}^{m-1} x_{i,0} \right) \otimes \left(\sum_{k=0}^{p-1} y_{0,k} \right) \otimes w_{0,0}.$$

with $m = p = 4$, $k = 2$ and $\mu = (1, 1)$ in the τ-theorem. This gives an ω-equivalent of ~ 2.90.

Example

Example: $2 \odot \langle 4,9,4\rangle$ in 243 multiplications (instead of $2 \cdot (4 \cdot 9 \cdot 4) = 288$) with:

$$t_1 = \sum_{i=0}^{m-1}\sum_{k=0}^{p-1}(x_{i,0} + \varepsilon u_{0,k,i}) \otimes (y_{0,k} + \varepsilon v_{k,i,0}) \otimes (\varepsilon^2 z_{k,i} + w_{0,0}) - \left(\sum_{i=0}^{m-1} x_{i,0}\right) \otimes \left(\sum_{k=0}^{p-1} y_{0,k}\right) \otimes w_{0,0}.$$

with $m = p = 4$, $k = 2$ and $\mu = (1,1)$ in the τ-theorem. This gives an ω-equivalent of ~ 2.90.

Better, with the same tensor: $\mu = (4,2), k = 6, m = p = 4$:
$15 \odot \langle 256, 81, 256\rangle$ matrix products in 23604048 multiplications, ω-equivalent ~ 2.80.

Example

Example: $2 \odot \langle 4, 9, 4 \rangle$ in 243 multiplications (instead of $2 \cdot (4 \cdot 9 \cdot 4) = 288$) with:

$$t_1 = \sum_{i=0}^{m-1} \sum_{k=0}^{p-1} (x_{i,0} + \varepsilon u_{0,k,i}) \otimes (y_{0,k} + \varepsilon v_{k,i,0}) \otimes (\varepsilon^2 z_{k,i} + w_{0,0})$$
$$- \left(\sum_{i=0}^{m-1} x_{i,0} \right) \otimes \left(\sum_{k=0}^{p-1} y_{0,k} \right) \otimes w_{0,0}.$$

with $m = p = 4$, $k = 2$ and $\mu = (1, 1)$ in the τ-theorem. This gives an ω-equivalent of ~ 2.90.

Better, with the same tensor: $\mu = (4, 2), k = 6, m = p = 4$:
$15 \odot \langle 256, 81, 256 \rangle$ matrix products in 23604048 multiplications, ω-equivalent ~ 2.80.
Even better, not built explicitly: $\mu = (10, 5)$, ω-equivalent ~ 2.729.

Software implementation in OCaml

- Parse degenerate tensors as Pan-style aggregation tables;
- Compose tensors symbolically;
- Extract a given coefficient $\mu \odot \langle \prod m_i^{\mu_i}, \prod n_i^{\mu_i}, \prod p_i^{\mu_i} \rangle$ following the τ-theorem;
- Test of tensors by applying them to random matrices;
- Maple code generation which computes the rank of a subterm of a power of tensor without actually computing it;
- C++ code generation implementing a given tensor.

Specifics of doing this in OCaml

- Static typing much helpful;
- Caveat: some algebraic computations had to be recoded;
- Symbolic computations on algorithms akin to compilation passes: AST manipulation;
- Some interaction with Maple : generating code to do some computations;
- Parametricity: Export possible to Latex, C++, Maple.

How to Use this Result and Implementation, Future Work

Roadmap of use

- Try out new or modified Pan Tables $\Rightarrow$ extract good algorithms;
- Optimize corresponding code as much as possible (cache, other algorithms at leaves, ...).

Future work

Finish trying out all Pan tables.

This work showed improvements in ω are not purely theoretical results.
$\Rightarrow$ Adapt other theoretical improvements to build concrete tensors?

Thank you for your attention!

Any questions?