SINBAD Consortium 2010

Kronecker product optimization in SPOT

Sébastien PACTEAU

SLIM
University of British Columbia

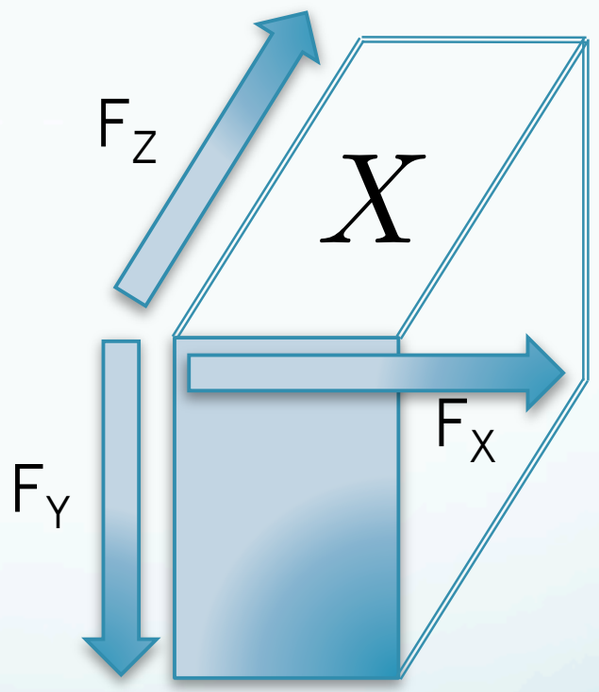
Introduction

Kronecker product use

- Multidimensional transforms on sets of data
- Example: n-D Fourier transform F

$$F(X) = (F_n \otimes F_{n-1} \otimes \ldots \otimes F_1) X(:)_{\text{dim1}}$$
$$(+\text{final reshaping})$$

where : $X(:)_{\text{dim1}}$ data set X collapsed along dimension 1



Introduction

- **Kronecker product interest**

  - KP decomposition **induces weak dependences** on multiplied data (presence of block diagonal operators)

    - possibility of <u>parallel calculations</u>

    - <u>optimization</u> analyzing data's selected parts

➔ **Possible speeding-up of computations**

Program

I. Initial Kronecker product (KP)

1. General approach
2. KP multiplication algorithm in SPOT
3. Initial result

II. KP optimization

1. Permutations
2. Replacement of the *for* loop
3. Final comparison

I. Initial Kronecker product

1. General approach

- **Definition**

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1c_A}B \\ \vdots & \ddots & \vdots \\ a_{r_A1}B & \cdots & a_{r_Ac_A}B \end{pmatrix}$$

- **KP representation in SPOT**
  - $K = \mathrm{opKron}(A, B)$
  - ... or with as many operators as wished: $K = \mathrm{opKron}(A_1, A_2, ..., A_n)$

- SPOT KP is an **implicit operator**:
  - never evaluated directly
  - only defined by its interaction with others operators when multiplying them
  - works on the selected parts of the right-hand multiplied operator

$$(A \otimes B)X = \mathrm{KP}(A, B)X$$

In Matlab

$$(A \otimes B)X = \mathrm{Function}(A, B, X)$$

In SPOT

➔ **faster than Matlab's KP multiplication on a vector**

Example:

A=rand(80); B= rand(80); X=rand(1600,1);

- In **MATLAB: 1.67s** on average

- In **SPOT: 0.03s** on average

➔ **56 times faster than Matlab's KP**

2. KP multiplication algorithm in SPOT

- Being given X a vector: $(A \otimes B)X = (A \otimes I_{r_B})(I_{c_A} \otimes B)X$

$$(A \otimes B)X = \begin{pmatrix} a_{11} & 0 & \cdots & a_{1c_A} & 0 & \cdots \\ 0 & \ddots & & 0 & \ddots & \\ \vdots & & a_{11} & \vdots & & a_{1c_A} \\ a_{r_A1} & 0 & \cdots & a_{r_Ac_A} & 0 & \cdots \\ 0 & \ddots & & 0 & \ddots & \\ & a_{r_A1} & & & & a_{r_Ac_A} \end{pmatrix} \begin{pmatrix} B & 0 & \\ & \ddots & \\ 0 & & B \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{c_Ac_B} \end{pmatrix}$$

r_B (over the repeated diagonal matrices); c_A repetitions of B

Repeating of diagonal matrices with the coefficients of A

Block diagonal matrix

- **Selection on the multiplied parts of X**

- **Successive multiplications** can be done with the underlying operators A and B

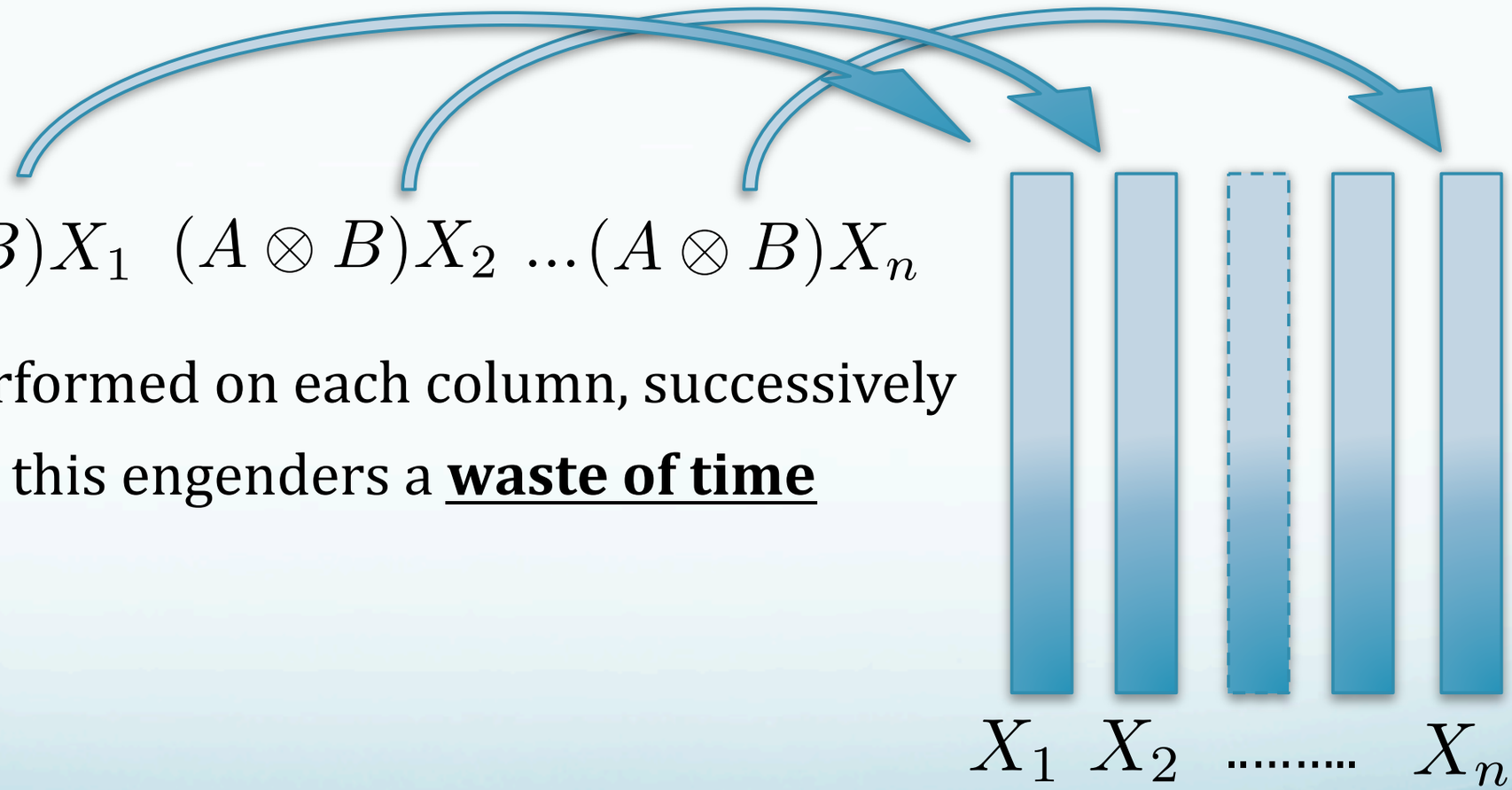
Beginning of the for loop

- reshaping of X → $\mathbf{X}_{\text{reshaped}}$
- multiplication by the operator $\mathbf{B}$: $\mathbf{X}_{\text{reshaped}} \times \mathbf{B} = \mathbf{X'}$
- Transposition of $\mathbf{X'}$ → $\mathbf{X''}$

End of the for loop

+ final collapsing

- **If *X* is a matrix:** SPOT performs a *for* loop on the columns of *X*



- KP performed on each column, successively
- ... but, this engenders a **waste of time**

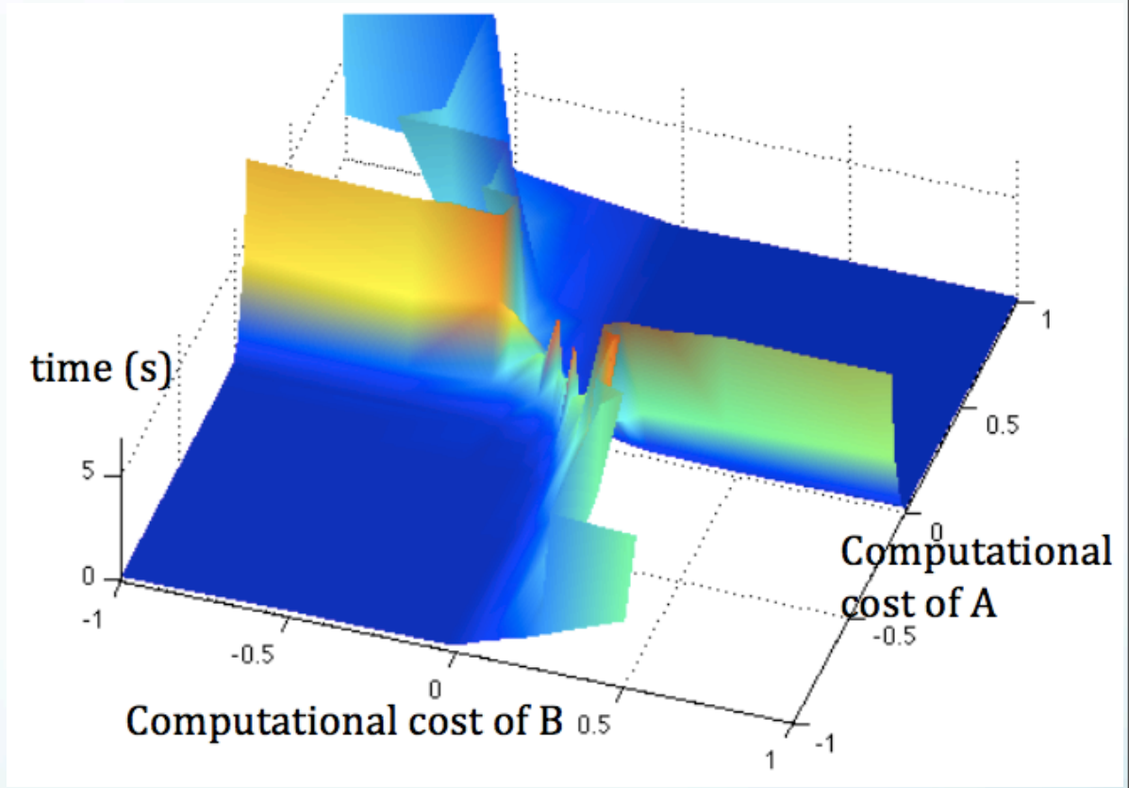
3. Initial result

- Computation of $(A \otimes B)X$, with X , identity matrix

  - **2 high computation time** zones appear

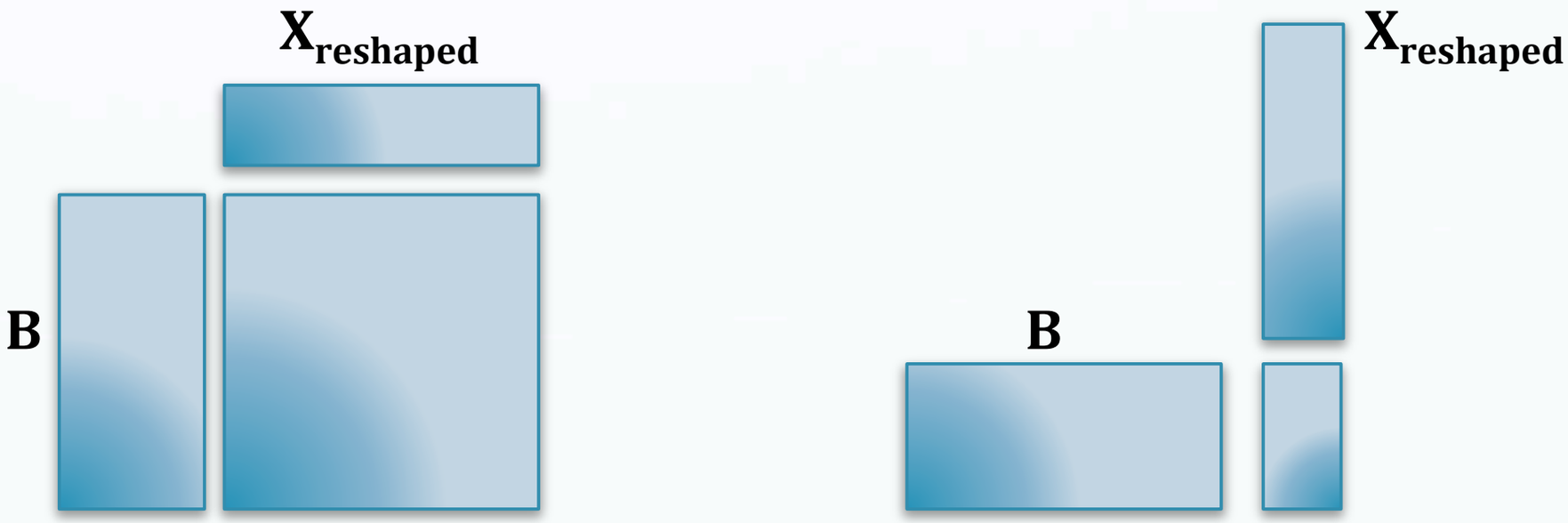
→ **Mainly due to <u>non optimal computational costs</u>**

$$C = \frac{r - c}{rc}$$



Computation time to calculate $(A \otimes B)$
(A and B with a fixed size of 2,500 elements each)

Explanation



High computational cost Low computational cost

Low computational cost → smaller intermediate matrix

Challenge

- **“to produce the smallest intermediate matrices”**

II. Kronecker product optimization

1. Permutations

- $(A \otimes B)X = (A \otimes I_{r_B})(I_{c_A} \otimes B)X = (I_{r_A} \otimes B)(A \otimes I_{c_B})X$

- More generally: **n! possibilities**

$$
\begin{aligned}
A_1 \otimes A_2 \otimes A_3 &= (A_1 \otimes I_{r_2 r_3})(I_{c_1} \otimes A_2 \otimes I_{r_3})(I_{c_1 c_2} \otimes A_3) \\
&= (A_1 \otimes I_{r_2 r_3})(I_{c_1 r_2} \otimes A_3)(I_{c_1} \otimes A_2 \otimes I_{c_3}) \\
&= (I_{r_1} \otimes A_2 \otimes I_{r_3})(A_1 \otimes I_{c_2 r_3})(I_{c_1 c_2} \otimes A_3) \\
&= (I_{r_1} \otimes A_2 \otimes I_{r_3})(I_{r_1 c_2} \otimes A_3)(A_1 \otimes I_{c_2 c_3}) \\
&= (I_{r_1 r_2} \otimes A_3)(A_1 \otimes I_{r_2 c_3})(I_{c_1} \otimes A_2 \otimes I_{c_3}) \\
&= (I_{r_1 r_2} \otimes A_3)(I_{r_1} \otimes A_2 \otimes I_{c_3})(A_1 \otimes I_{c_2 c_3})
\end{aligned}
$$

- **Which is the best permutation ?**

  "The one describing operators with **increasing computational costs**"

  → Addition of a method calculating the best permutation

  - Performs a quick sort
  - Initializes a *best_permutation* property

  → Multiplication according to *best_permutation* property

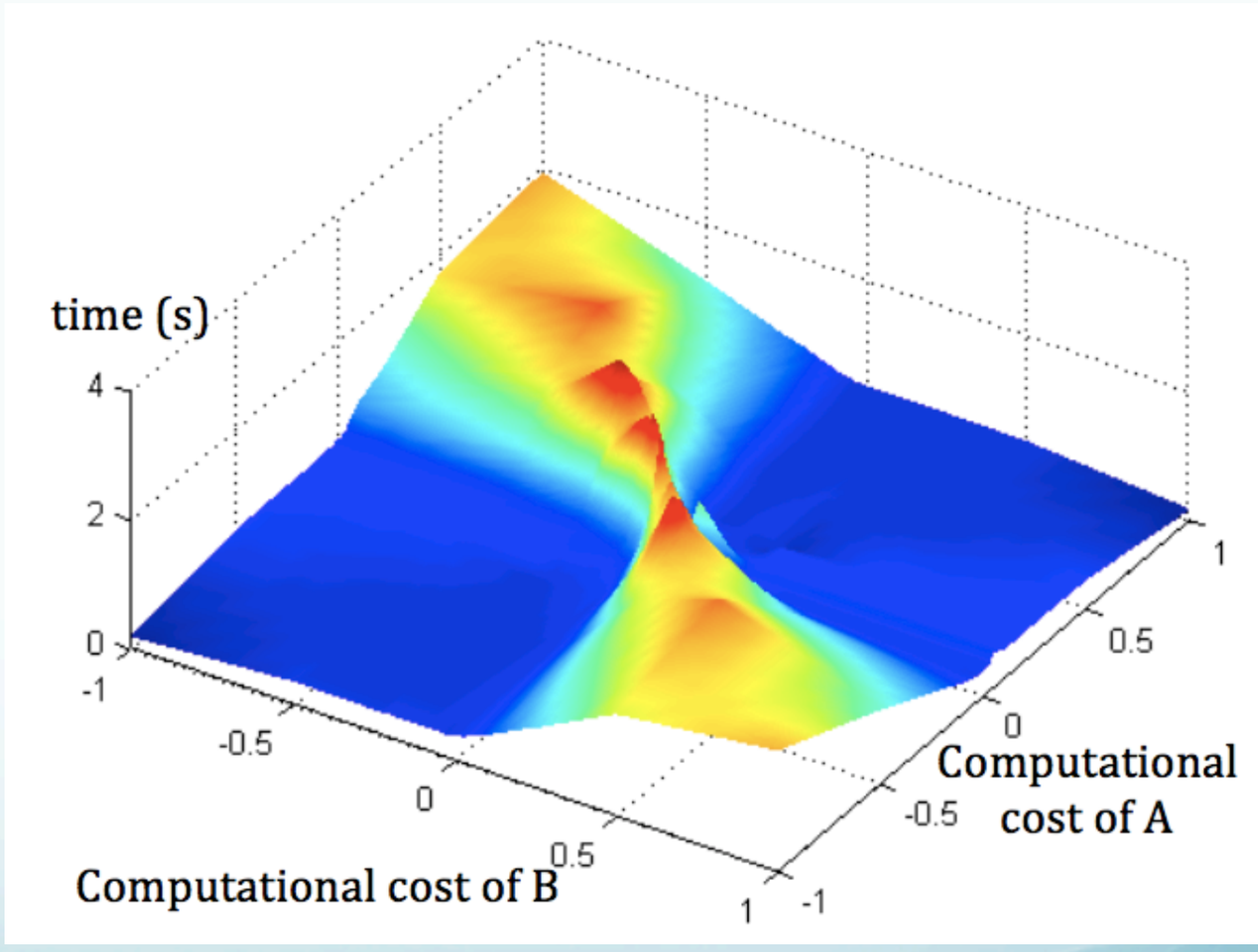
Result

Both high computation time zones have been considerably reduced

- **Average execution time**

- divided by 12 in high computation time zones

- about the same elsewhere



2. Replacement of the *for* loop

- Replaced by an **implicit implementation**
- *for* loop ⇔ **KP with an identity matrix**

$$I_n \otimes (\otimes_i A_i)$$

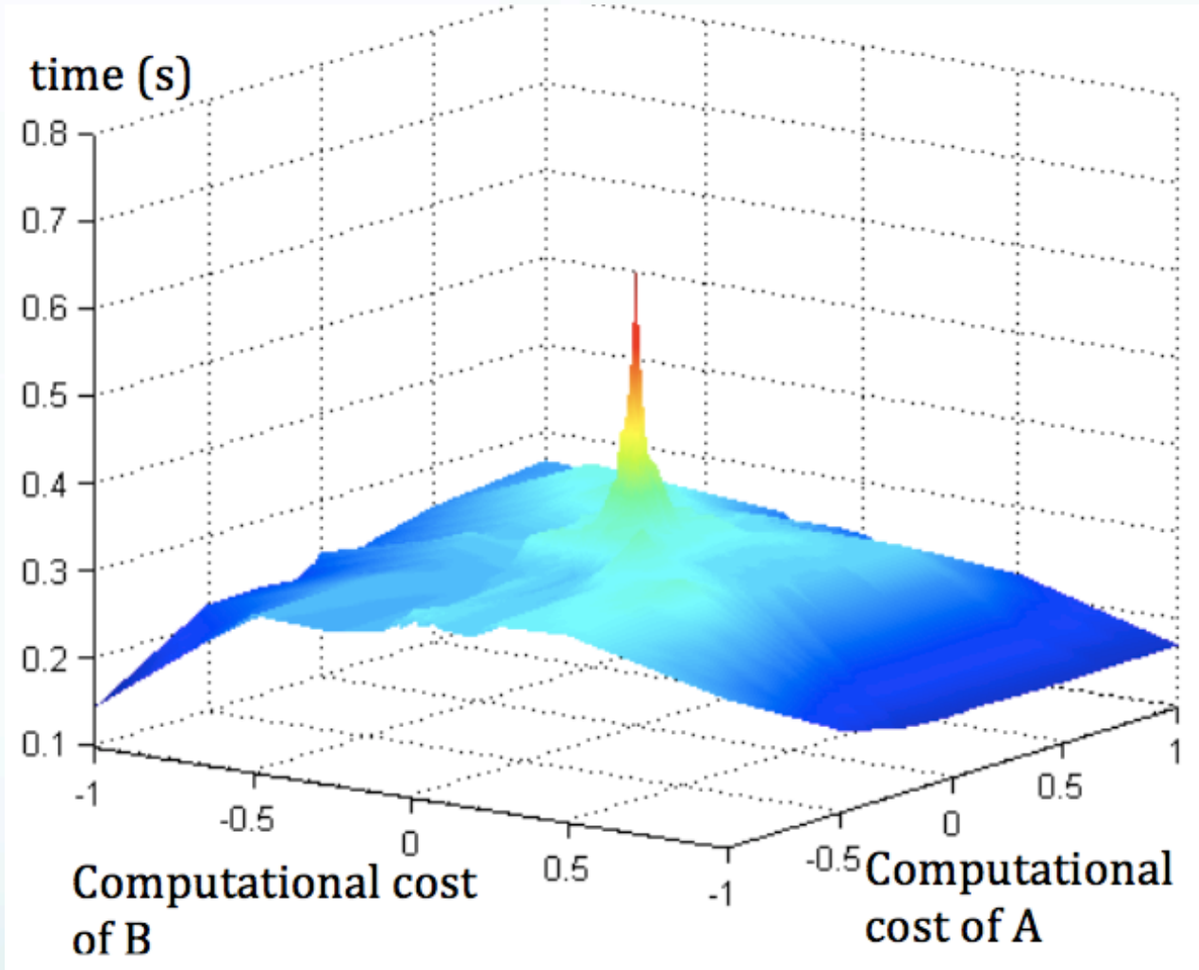
where, ***n*** : number of columns of the right-hand multiplied operator, *X*

- Equivalent to **make the size** of the first identity matrix in KP decomposition $\prod_i (I_? \otimes A_i \otimes I_?)$ being ***n* times bigger.**

Result

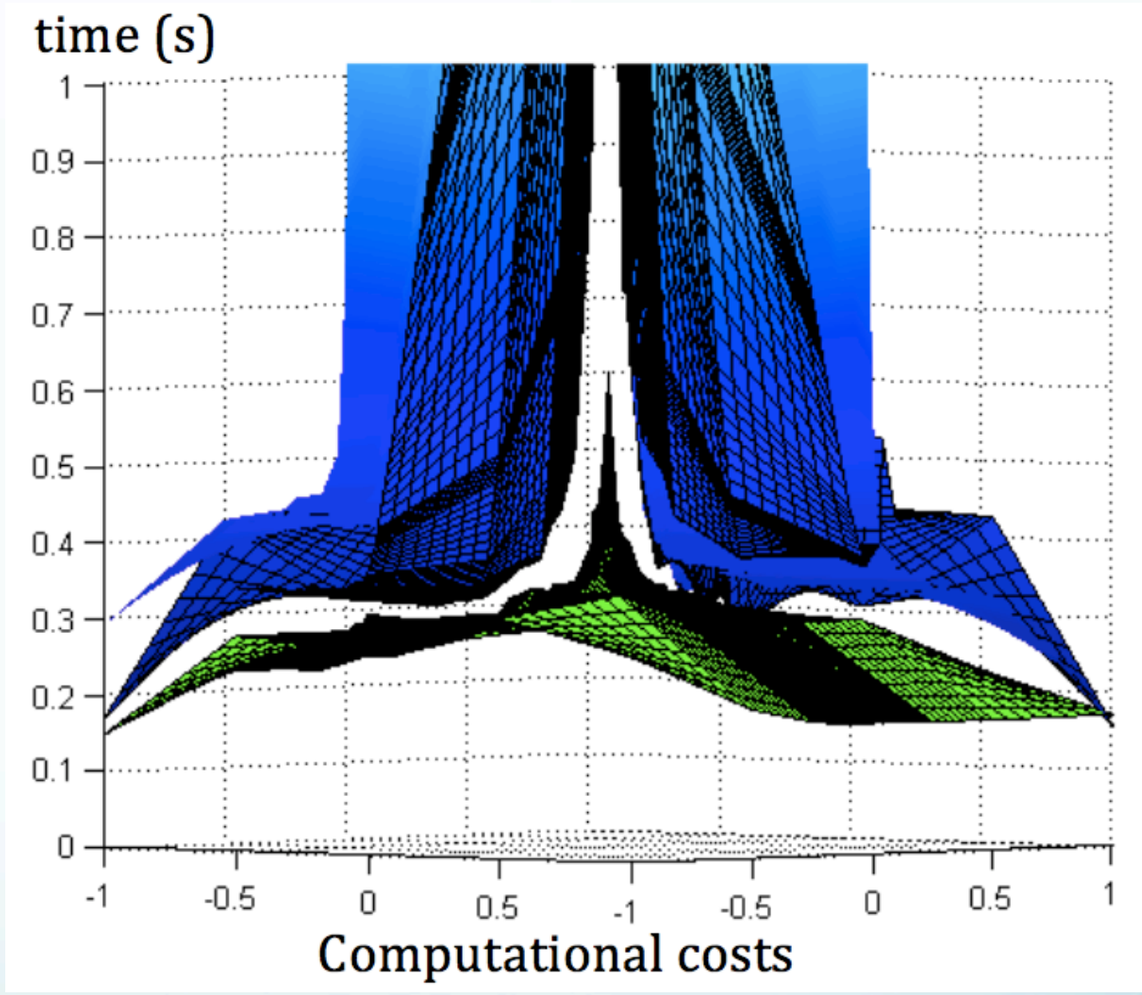
- **Calculation time improvement of 1,170%** compared to the initial KP multiplication function.

- **Highest computation time in the center** since it corresponds to larger intermediate matrices in calculations.



3. Final comparison

- Calculation time behavior
  - In "normal" zones
    - **divided by 2**
  - In previously high computation time zones
    - **reduced up to 50 times**
  - In the central peak region
    - **divided by 4**



Conclusion

- **Optimization of the <u>serial</u> Kronecker product multiplication**
  - Average gain of time of **1,170 %**

- **Improvements carried out in 2 steps**
  - Permutations → smaller intermediate matrices
  - Replacement of the *for* loop

Conclusion

- **Main advantage** of SPOT's Kronecker product
  - KP is **not entirely calculated** (consideration of its effects)
- **Next step**
  - **parallel** Kronecker product (in pSPOT)
- **Expectations**
  - **Improvements in long time processes**