Square Partitioning for the Computation of Parallel Matrix Multiplication

O.Beaumont, L.Eyraud-Dubois, T.Lambert

18 mai 2016

Table of Contents

Outer Product and Matrix Multiplication

The Outer Product :

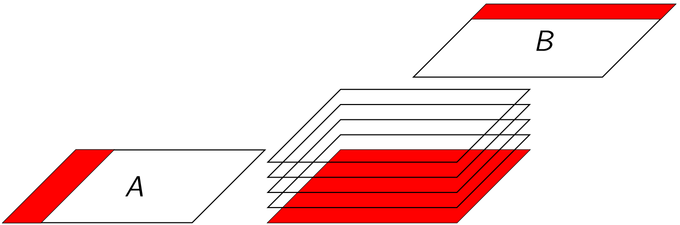
$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \otimes \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{pmatrix}$$

The Matrix Multiplication (Cannon's Algorithm) :

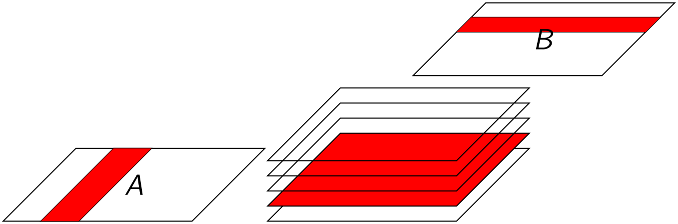
If $A = \begin{pmatrix} A_1 & \dots & A_N \end{pmatrix}$ and $B = \begin{pmatrix} B_1 \\ \dots \\ B_N \end{pmatrix}$ with $A_i = \begin{pmatrix} a_{1,i} \\ \dots \\ a_{N,i} \end{pmatrix}$ and $B_j = \begin{pmatrix} b_{1,j} & \dots & b_{N,j} \end{pmatrix}$ then

$$A \times B = \sum_{i=1}^{N} A_i \otimes B_i$$

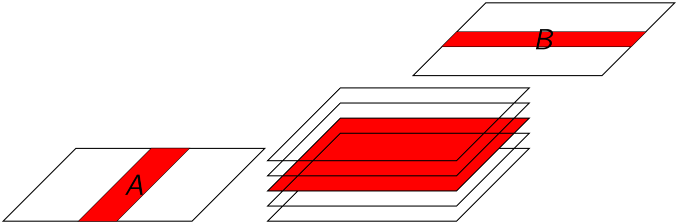
Cannon's Algorithm



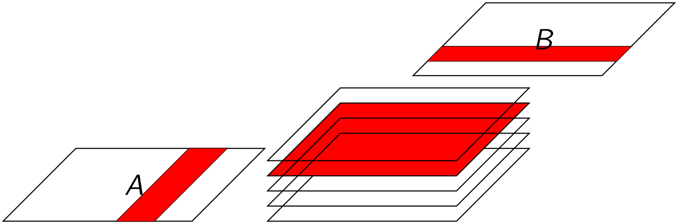
Cannon's Algorithm



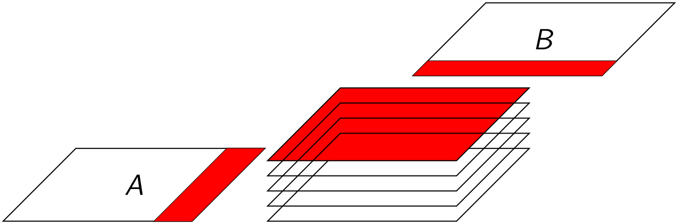
Cannon's Algorithm



Cannon's Algorithm



Cannon's Algorithm



Allocation of a Parallel Matrix Multiplication

Inputs :

- Two matrices $A = \begin{pmatrix} A_1 & \dots & A_N \end{pmatrix}$ and $B = \begin{pmatrix} B_1 \\ \dots \\ B_N \end{pmatrix}$
- A set of p processors P_k, each with a speed s_k.

Output :

- For each $A_l \otimes B_l$ and each processor P_k, the set $W_{l,k}$ of the tasks $a_{i,l} b_{l,j}$ associated to it.
- For each processor P_k the contents of its local memory, $m_{k,l,row}$ and $m_{k,l,columns}$.

Allocation of a Parallel Matrix Multiplication

- We want an optimal makespan (computation time). Therefore $|W_{l,k}|$ is fixed and correlated to the relative speed of P_k.
- We want to minimize communication. Therefore $|m_{k,l,row}|$ and $|m_{k,l,columns}|$ must be as low as possible.

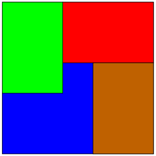
Problem

Given a platform of processors with known speeds, return an makespan-optimal scheduling that minimize the amount of communication.

Formal Problem Statement

Problem

Given a square $[0,1] \times [0,1]$ *and a set of areas, return a partition of the square into areas of the sizes given which minimizes the sum of the half-perimeter of the covering rectangles.*

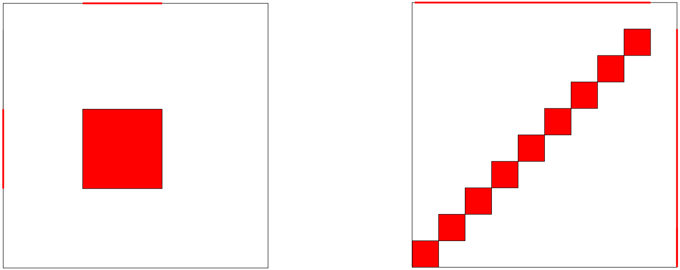


$$
\begin{aligned}
& 2/5 + 3/5 \\
+ \quad & 3/5 + 2/5 \\
+ \quad & 3/5 + 3/5 \\
+ \quad & 2/5 + 3/5 \\
= \quad & 21/5
\end{aligned}
$$

Table of Contents

A Remark

- To optimize the data reuse it is better to give area shaped as squares.



- In fact it is the lower bound (half-perimeter $\geq 2 * \sqrt{\text{surface}}$).

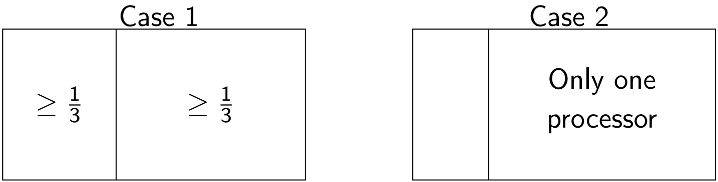
An Already Studied Problem

- More constrained problem : Split a square in several rectangle of fixed area and minimize the semi-perimeters.
- Already studied, NP-complete but there exists a 5/4-approximation (Nagamochi et al.) that becomes a $\frac{2}{\sqrt{3}}$-approximation ($\frac{2}{\sqrt{3}} \simeq 1.15$) for slightly heterogeneous platforms (Fügenschuh et al.).

Nagamochi's Algorithm

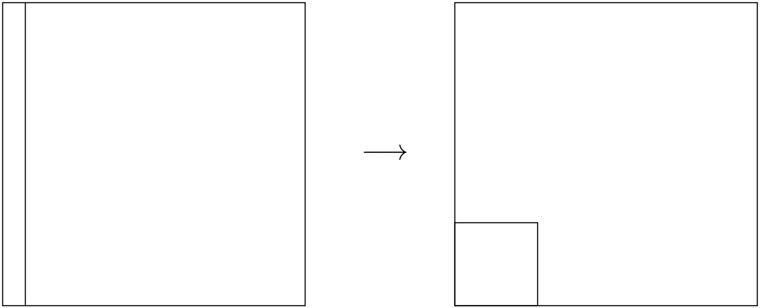
A divide and conquer algorithm :

- Sort the processors by increasing speed.
- Recursively split the current rectangle in two and apply the algorithm on each subrectangle.



A New Approach for Pathological Cases (Lastovetsky et al.)

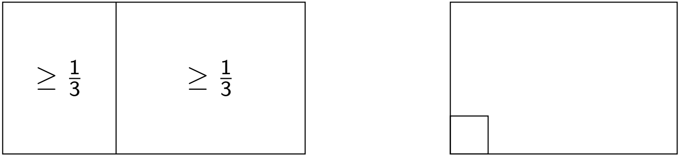
- The problem is rectangle with an aspect ratio greater than 3.
- When a rectangle is too elongated we transform it into a square.



A New Algorithm

A variant of the divide and conquer algorithm :

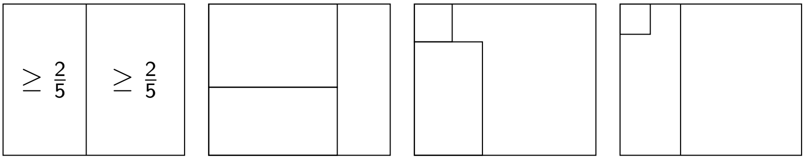
- Sort the processors by increasing speed.
- Recursively split the current rectangle in two :
  - If it is possible in two rectangle with aspect ratio under 3.
  - Else into a squared zone and its complement.



Result : A $\sqrt{\frac{3}{2}}$-approximation ($\sqrt{\frac{3}{2}} \simeq 1.22$).

Non-Rectangular Recursive Partitioning (NRRP)

NRRP is a refinement of the previous algorithm where some new subcases are added :

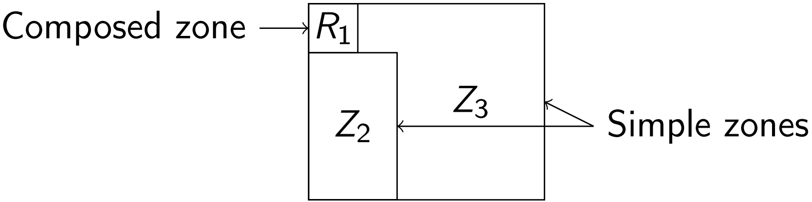


Result : A $\frac{2}{\sqrt{3}}$-approximation ($\frac{2}{\sqrt{3}} \simeq 1.15$).

Sketch of Proof

We distinguish two kinds of zones created during a step :

- Simple zones : terminal zones allocated to a single processor.
- Composed zones : union of future simple zones on which we recursively apply the algorithm.



Local Invariants

Lemma

At each call of NRRP, the produced composed zones are rectangles with an aspect ratio less than 2.5.

Lemma

At each call of NRRP, if $\{Z_1, \ldots, Z_k\}$ is the set of the produced simple zones, A_{Z_i} their areas and w_{Z_i} and h_{Z_i} the width and the height of their covering rectangles then :

$$\frac{\sum_{i=1}^{k} w_{Z_i} + h_{Z_i}}{2\sum_{i=1}^{k}\sqrt{A_{Z_i}}} \leq \frac{2}{\sqrt{3}}.$$

We recall that $\frac{a+b}{c+d} \leq max(\frac{a}{c}, \frac{b}{d})$.

Experimental Validation

We propose an experimental validation :

- Platforms are composed of n CPUs, m Xeon Phi accelerators and p GPUs ($n \in \{1, 2, 4, 8, 12, 16, 24, 32, 64\}$, $m, p \in [0, 8]$). Accelerators are between 15 and 25 times faster than CPU and GPUs are between 25 and 35 times faster than CPUs.
- We compare NRRP with Nagamochi's one and the Column Based approach.
- Column Based heuristic computes (with dynamic programming) the best possible partitioning under the assumption that the original square is split into several columns.

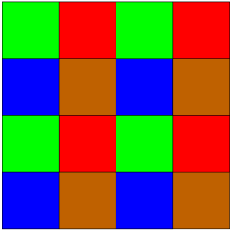
Practical Results

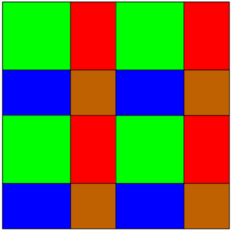
- Column Based is good for very large platforms (below 1.05 times the bound) but this ratio greatly increase for small instances (can be more than 1.5 times the bound). In contrast, NRRP and Nagamochi are far better on small instances, with a little advantage for NRRP on really small ones.
- NRRP has a far smaller worst case, around 1.1 times the bound, the worst ratio is 1.3 for Nagamochi and more than 1.5 for Colum Based.
- A heuristic which chooses the best of the three has a worst case of 1.08 times the bound, below 1.02 on average.

Table of Contents

Discrete Partitioning

- In practice we are interested in splitting matrices, that are discrete squares.
- Therefore we can't cut it anywhere.
- Direct rounding can be quite bad for load balancing.

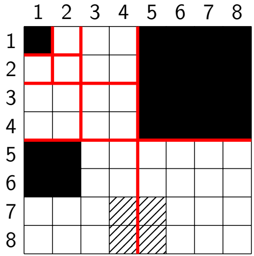




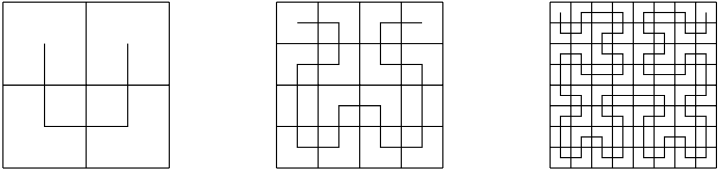
Recursive decomposition

Definition (q-square)

We denote as q-squares all the subsquares of the square $[1, N] \times [1, N]$ which are of the form $[1 + n_1 \times 2^q, (n_1 + 1) \times 2^q] \times [n_2 \times 2^q, (n_2 + 1) \times 2^q]$ where $n_1, n_2 \in [0, N/2^q - 1]^2$.



Hilbert Curve



Property

If S_q is a q-square, then there exists an integer n such that $H([n, n+4^q-1]) = S_q$.

Property

For all $n \in [1, N^2-1]$, if we denote $(i_1, j_1) = H(n)$ and $(i_2, j_2) = H(n+1)$, then $i_1 = i_2$ or $j_1 = j_2$.

Algorithm

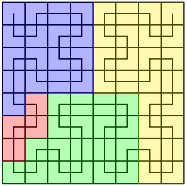
Algorithm 1: Fractal Partition

$s_{tot} = 0$

foreach $k \in [1, p]$ **do**

$\quad A_k = H([s_{tot}, s_{tot} + s_k])$

$\quad s_{tot} = s_{tot} + s_k + 1$



$s_1 = 17$

$s_2 = 4$

$s_3 = 19$

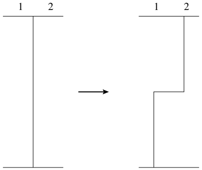
$s_1 = 24$

Claim

This is a $\frac{3\sqrt{3}}{\sqrt{11}}$-approximation ($\simeq 1.56$).

Conclusion

- First approximation ratio for the discrete problem.
- In practice a more trivial solution works better.



- However space-filling curves are a very classical way to partition data set.
- It also seems easier to generalize for higher dimension and for a variant with sparse matrices.

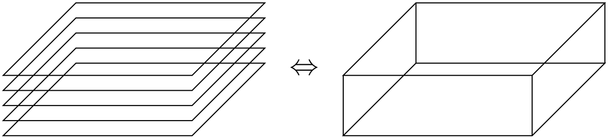
Table of Contents

Motivation

- We recall that the matrix multiplication of scheduling is just the scheduling of the equivalent basic tasks :

$$t_{i,j,k} : c_{i,j} \leftarrow c_{i,j} + a_{i,k} b_{k,j}$$

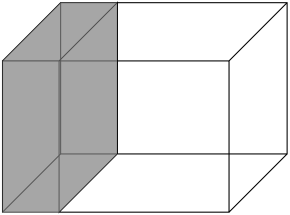
- Cannon's algorithm schedules the tasks by set with fixed k.
- In this section we propose to schedule the whole set of tasks.

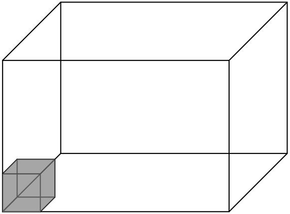


- A new problem : division of a cube (or a cuboid for non-square matrix) into volumes minimizing the half-surface of the covering cuboid.
- Problem without pre-existing work as far as we know.

Approximation Algorithm

- We prove the NP-completeness of the problem.
- We propose a variant of NRRP in the 3D case (3D-NRRP) with only two cases.





- The algorithm can be shown to be a $\frac{5}{6^{2/3}}$-approximation ($\frac{5}{6^{2/3}} \simeq 1.51$).

Table of Contents

Perspectives

- Add new analysis to 3D-problem (in particular its discrete variant), if possible find an easier NP-completeness proof.
- Begin the study of nD-problem (tensor product).
- Begin the study of a similar problem but with sparse matrices (Fast Multipole Method).