On the Effect of Replication of Input Files on the Efficiency and the Robustness of a Set of Computations

Thomas Lambert
PhD supervised by Olivier Beaumont and Lionel Eyraud-Dubois

September 8 2017
A Problem of Plagiarism

- 50 examination papers and we want to detect all plagiarisms
- Compare two examination papers: 5 minutes
- \( \frac{50 \times 49}{2} \times 5 = 6125 \text{ min} \geq 100 \text{ hours} \text{ if done alone!} \)
A Problem of Plagiarism

- 50 examination papers and we want to detect all plagiarisms
- Compare two examination papers: 5 minutes
- $\frac{50 \times 49}{2} \times \frac{5}{10} = 612.5 \text{ min} \approx 10 \text{ hours}$ if we are 10?
A Problem of Plagiarism

- Two examiners: ♂ ♂, three examination papers: A B C
- Compare paper A with papers B and C

Diagram:

- A B → A ↔ B
- Waiting
- End

We need to replicate to improve parallelism. There is a cost (ink, paper, copy time).

T.Lambert
On the Effect of Replication of Input Files
A Problem of Plagiarism

- Two examiners: A, B, C, three examination papers: A, B, C
- Compare paper A with papers B and C

We need to replicate to improve parallelism
- Replicating induces a cost (ink, paper, copy time)
Replication of Input Files

- Examination papers → input files
- Replication increases parallelism
- Cost: storage, communication (data movement)
Communication

- Communication: every data movement (uploading input files, communications between processors, ...), not free!
  - Delay (latency, bandwidth)
  - Energy consumption
- Challenge: decrease communication without degrading completion time (makespan) → communication-avoiding algorithms

**Objective of the Thesis**

Focus on practical problems with input file replication issues with communication-avoiding concerns
HPC (High-Performance Computing)

- Reproduce a computer architecture at larger scale (supercomputer)
- Many processing units (CPU, GPU, ...) linked with very fast network
- Centralised storage "far" from processing units
- Focus on scientific computations
- Example of communication issue: how to avoid too many uploads from central storage?
Cloud Computing and Big Data

- Several node consisting in processing unit and storage
- Data is closer to processors and replicated at several places
- The overall bandwidth between nodes is limited
- Focus on data-intensive applications
- Example of communication issue: how to process tasks locally (vs sending input data)?
Matrix Product and MapReduce

- **Matrix product:**
  - Core applications of many scientific computations (HPC oriented problem)
  - Replication of the input data is necessary for parallelism (shared input data) and done during execution
  - Replications are the objective function to minimize

- **"Map" phase of MapReduce applications:**
  - Paradigm of distributed computing (Big Data oriented problem)
  - Input files are replicated and placed on computers before the beginning of the execution
  - Replication is an input of the problem and have to be use in the best possible way
Table of Contents

1 Introduction

2 Communication-Avoiding Parallel Matrix Multiplication

3 Improving Data Locality of Map Tasks in MapReduce

4 Conclusion
Formal Problem Statement

- Model initially introduced by Kalinov and Lastovetsky (2001)

**Problem (PERI-SUM)**

*Given a square* $[0, N] \times [0, N]$ *and a set of values, return a partition of the square into zones of the given surfaces minimizing the sum of the half-perimeter of the covering rectangles.*

With $N = 5$ and surfaces $6, 6, 6, 6, 1$

\[
\begin{align*}
2 + 3 \\
+ & \quad 3 + 2 \\
+ & \quad 3 + 2 \\
+ & \quad 2 + 3 \\
+ & \quad 1 + 1 \\
= & \quad 22
\end{align*}
\]
Lower Bound

- To optimize the data reuse it is better to assign zones shaped as squares

- In fact this is the lower bound \( \text{half-perimeter} \geq 2 \times \sqrt{\text{area}} \) \(^{(Ballard \ et \ al. \ (2011))}\)
Related Work

- General studies:
  - NP-completeness (Beaumont et al.)
  - Optimal solutions for 2 and 3 processors (Lastovetsky et al.)

- Focusing on rectangle-shaped zones:
  - 7/4-approximation (Beaumont et al.)
  - Improvement to a 5/4-approximation (Nagamochi and Abe)
  - In the case of homogeneous or weakly heterogeneous surfaces: a $\frac{2}{\sqrt{3}}$-approximation ($\frac{2}{\sqrt{3}} \approx 1.15$) (Fügenschuh et al.)
Contributions in Communication-Avoiding Matrix Product

- **Square partitioning model:**
  - Simulations of hybrid strategies to prove reliability of the approach
  - Approximation algorithms:
    - Divide and conquer (NRRP, $\frac{2}{\sqrt{3}} \approx 1.15$-approximation)
    - Space-filling curve based (SFCP, $\frac{3\sqrt{3}}{\sqrt{11}} \approx 1.57$-approximation)

- **Cube partitioning model:**
  - Introduction of the model
  - NP-Completeness
  - Extension of 2D approximation algorithms:
    - Replication under the third dimension
    - Divide and conquer (3D-NRRP, $\frac{5}{6^{2/3}} \approx 1.51$-approximation)
    - Space-filling curve based (3D-SFCP, $\frac{7^{5/3}}{6^{2/3}} \approx 1.64$-approximation)

- **Practical implementation with StarPU library**
Nagamochi and Abe’s Algorithm

A divide and conquer algorithm:
- Sort the processors by increasing speed
- Recursively split the current rectangle and the set of processors in two and apply the algorithm on each subrectangle

\[
\begin{array}{|c|c|}
\hline
s_1 & s_2 & s_3 & s_4 & s_5 \\
\hline
\end{array}
\]

Case 1

\[
\begin{array}{|c|c|}
\hline
\geq \frac{1}{3} & \geq \frac{1}{3} \\
\hline
\end{array}
\]

Case 2

Only one processor

\[
\begin{array}{|c|c|}
\hline
s_1 & s_2 & s_4 & s_3 \\
\hline
\end{array}
\]
An Approach for Pathological Cases (Lastovetsky et al.)

- The problem are "tall and skinny" rectangles
- When a rectangle is too tall and skinny, transform it into a square
SNRRP (Simple Non-Rectangular Recursive Partitioning)

A variant of the divide and conquer algorithm:

- Sort the processors by increasing speed
- Recursively split the current rectangle and the set of processors in two:
  - If it is possible, in two rectangles with aspect ratio below 3
SNRRP (Simple Non-Rectangular Recursive Partitioning)

A variant of the divide and conquer algorithm:

- Sort the processors by increasing speed
- Recursively split the current rectangle and the set of processors in two:
  - If it is possible, in two rectangles with aspect ratio below 3
  - Otherwise into a squared zone and its complement

\[
\begin{array}{c|c}
\geq \frac{1}{3} & \geq \frac{1}{3} \\
\end{array}
\]

Result: A $\sqrt{\frac{3}{2}}$-approximation ($\sqrt{\frac{3}{2}} \approx 1.22$)
Improving SNRRP, Aspect Ratio

\[ \rho = \frac{\max(w, h)}{\min(w, h)} \]

- \( \rho = 1 \) → square, \( \rho \) big → tall and skinny rectangle

**Lemma**

For a rectangle of aspect ratio \( \rho \):

\[ \frac{\text{Half-perimeter}}{\text{Lower Bound}} = \frac{\rho + 1}{2\sqrt{\rho}} \]

- \( x \mapsto \frac{x+1}{2\sqrt{x}} \) increasing on \([1, +\infty[\)
- We aim for covering rectangles with the smallest possible aspect ratio
Lemma (Nagamochi and Abe (2007))

Assuming the surfaces are sorted in increasing order:

- If such splitting is possible, then the aspect ratio of both rectangles is below 3
- If it is not possible then there is only one processor on the right
## Improving SNRRP

| $\geq \frac{1}{\mu}$ | $\geq \frac{1}{\mu}$ |

### Lemma

**Assuming the surfaces are sorted in increasing order:**

- *If such splitting is possible, then the aspect ratio of both rectangles is below $\mu$*
- *If it is not possible then there is only one processor on the right*
Lemma

Assuming the surfaces are sorted in increasing order:

- If such splitting is possible, then the aspect ratio of both rectangles is below $\mu$.
- If it is not possible then there is only one processor on the right.
Non-Rectangular Recursive Partitioning (NRRP)

NRRP is a refinement of the previous algorithm where notably

\[ \mu = \frac{5}{2} \]
NRRP is a refinement of the previous algorithm where notably
\[ \mu = \frac{5}{2} \]
Non-Rectangular Recursive Partitioning (NRRP)

NRRP is a refinement of the previous algorithm where notably $\mu = \frac{5}{2}$

Result: A $\frac{2}{\sqrt{3}}$-approximation ($\frac{2}{\sqrt{3}} \approx 1.15$)
We distinguish two kinds of zones created during a step:

- Simple zones: terminal zones allocated to a single processor
- Composed zones: zones on which the algorithm is recursively applied

\[ \text{Simple zones} \rightarrow R_1 \rightarrow Z_2 \rightarrow Z_3 \rightarrow \text{Composed zone} \rightarrow \text{Simple zones} \]
Local Invariants

Lemma (Invariant for composed zones)

At each call of NRRP, the produced composed zones are rectangles with an aspect ratio less than $\frac{5}{2}$.

\[
\frac{\max(w_1,h_1)}{\min(w_1,h_1)} < \frac{5}{2}
\]
Lemma (Invariant for simple zones)

Assuming previous invariant, 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:

\[
\frac{\sum_{i=1}^{k} w_{Z_i}}{2} + h_{Z_i} \leq \frac{2}{\sqrt{3}}
\]

\[
\frac{\sum_{i=1}^{k} \sqrt{A_{Z_i}}}{2} \leq \frac{2}{\sqrt{3}}
\]
Formal Problem Statement

Matrix product: \( C_{i,j} \leftarrow C_{i,j} + A_{i,k}B_{k,j} \) for all \( i, j, k \)

**Problem**

*Given a cuboid \([0, N] \times [0, N] \times [0, N]\) and a set of values, return a partition of the cuboid into polyhedrons of the given volumes minimizing the sum of the half-area of the covering cuboids of each polyhedron.*
Some strong similarities between 2D and 3D partitioning:
- NP-completeness (reduction from Partition Problem)
- We can adapt the lower-bound: half-area $\geq 3 \times \text{volume}^{2/3}$, the best possible shape for a polyhedron is a cube and the cuboid shapes improve the data reuse
- Cuboid with bad aspect ratio ($\frac{\text{largest dimension}}{\text{smallest dimension}}$) can degrade the ratio

We propose an adaptation of SNRRP: 3D-NRRP
3D-NRRP

- Sort the processors by increasing speed
- Recursively split the current cuboid in two:
  - If it is possible, in two cuboids with aspect ratio below 3
  - Otherwise into a cubed polyhedron and its complement or a "squeezed" cube and its complement

\[
s_1 \ s_2 \ s_3 \ s_4 \ s_5
\]

Result: A \( \frac{5}{6^{2/3}} \)-approximation \( \frac{5}{6^{2/3}} \approx 1.51 \) in the case where the cuboid is cubic
3D-NRRP

- Sort the processors by increasing speed
- Recursively split the current cuboid in two:
  - If it is possible, in two cuboids with aspect ratio below 3
  - Otherwise into a cubed polyhedron and its complement or a "squeezed" cube and its complement

\[
\begin{align*}
\tilde{A} & = \frac{5}{6^{2/3}} \\
\tilde{A} & \approx 1.51
\end{align*}
\]

Result: A \( \frac{5}{6^{2/3}} \)-approximation \( \frac{5}{6^{2/3}} \approx 1.51 \) in the case where the cuboid is cubic
General Settings

- Implemented with StarPU and run on a PlaFRIM node
- Platform with 4 GPUs and $2 \times 10$ CPUs
- Task allocation made with NRRP algorithm and additional dynamic settings:
  - **Static**: Respect initial repartition until the end
  - **RandSteal**: If idle, a processor execute a task attributed to another processor and randomly chosen
  - **EffectiveSteal**: If idle, a processor execute the task attributed to another processor that implies the less data movement
- We also consider **DMDA**, a purely dynamic and by default StarPU scheduling strategy which aim at minimal completion time
Makespan Results

<table>
<thead>
<tr>
<th>Strategies</th>
<th>N = 7680</th>
<th>N = 15360</th>
<th>N = 23040</th>
<th>N = 30720</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RandSteal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EffectiveSteal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T.Lambert On the Effect of Replication of Input Files
Communication Results

<table>
<thead>
<tr>
<th>Strategies</th>
<th>Dynamic</th>
<th>NRRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMDA</td>
<td><img src="chart_data1.png" alt="Chart Data" /></td>
<td><img src="chart_data2.png" alt="Chart Data" /></td>
</tr>
<tr>
<td>Static</td>
<td><img src="chart_data3.png" alt="Chart Data" /></td>
<td><img src="chart_data4.png" alt="Chart Data" /></td>
</tr>
<tr>
<td>RandSteal</td>
<td><img src="chart_data5.png" alt="Chart Data" /></td>
<td><img src="chart_data6.png" alt="Chart Data" /></td>
</tr>
<tr>
<td>EffectiveSteal</td>
<td><img src="chart_data7.png" alt="Chart Data" /></td>
<td><img src="chart_data8.png" alt="Chart Data" /></td>
</tr>
</tbody>
</table>

Legend:
- N = 7680
- N = 15360
- N = 23040
- N = 30720

NRRP:
- ●

Communication (GB):
- Dynamic:
  - N = 7680: 3.3
  - N = 15360: 3.0
  - N = 23040: 2.7
  - N = 30720: 2.4

NRRP:
- N = 7680: 2.4
- N = 15360: 2.7
- N = 23040: 3.0
- N = 30720: 3.3

T.Lambert: On the Effect of Replication of Input Files
Experiment Summary

- **Makespan:**
  - NRRP-based strategies close from DMDA but the dynamic part of scheduling is necessary

- **Communication**
  - Differences between RandSteal and EffectiveSteal are not negligible
  - Decrease of transferred GB with NRRP-based strategies in comparison to DMDA

<table>
<thead>
<tr>
<th></th>
<th>DMDA</th>
<th>NRRP-EffectiveSteal</th>
</tr>
</thead>
<tbody>
<tr>
<td>N = 7680</td>
<td>≃ 3.2 GB</td>
<td>≃ 2.7 GB (−15.6%)</td>
</tr>
<tr>
<td>N = 15360</td>
<td>≃ 14.5 GB</td>
<td>≃ 10.5 GB (−27.6%)</td>
</tr>
<tr>
<td>N = 23040</td>
<td>≃ 33 GB</td>
<td>≃ 24 GB (−27.2%)</td>
</tr>
<tr>
<td>N = 30720</td>
<td>≃ 58 GB</td>
<td>≃ 41 GB (−29.3%)</td>
</tr>
</tbody>
</table>

- We significantly reduce communication without degrading the makespan ✓
Table of Contents

1. Introduction
2. Communication-Avoiding Parallel Matrix Multiplication
3. Improving Data Locality of Map Tasks in MapReduce
4. Conclusion
Map/Reduce

- Distributed computing paradigm popularized by Google
- Since several open source implementations, notably Hadoop
- The principle is to split a job into several smaller tasks, two phases:
  - "Map" phase: Distribution of the tasks between the processors where each is processed
  - "Reduce" phase: Upload of the partial results that are aggregated to obtain the final result
Map/Reduce: an example

Goal: To count the number of occurrences of each letter in the world "abracadabra".

```
"abracadabra"
map

('r',1)
('b',1)
('a',1)
('d',1)
('a',1)
('b',1)
('a',1)
('c',1)
('a',1)
('b',1)
('r',1)

reduce

('a',5)
('b',2)
('d',1)
('r',2)
('c',2)
```
Map Phase

- We focus on the "Map" phase
- Distribution of a set of independent tasks on a set of processors (one input file per task)
- In Hadoop Distributed File System (HDFS) the input files are replicated and randomly placed on processors before the execution
  - Fault tolerance
  - More available processors for dynamic allocation
  - Data storage cost linked to the number of replication
- We assume tasks and processors are homogeneous (same processing time for all tasks and processors, no "stragglers")
Contributions in Data Locality of Map Tasks in MapReduce

- Model of the issue with a graph problem
- Optimal algorithms for two metrics on this graph problem
- Study of expected value of the optimal solution for makespan metric
- Study of expected value of greedy strategy with balls-into-bins models
- Simulation based on traces of MapReduce to evaluate the reliability of the previous algorithms in heterogeneous environment
We represent the problem with a bipartite graph $G = (P, T, E)$.

- $P$ set of processors. $T$ set of tasks.
- $(p_i, t_j) \in E$ if and only if the input file of $t_j$ is present on the memory node of $p_j$. 

![Bipartite Graph Diagram]
Makespan

- **Objective**: minimization of makespan, *i.e.* the number of tasks per processors
- **Lower bound** $\left\lceil \frac{n}{m} \right\rceil$ (*n* tasks, *m* processors), "perfect" scheduling

![Diagram of tasks and processors]
Communication

- Input files placement is random, there is a risk of bad repartition
- Degradation of makespan
- A possible correction is to allow the copy and the movement of files during the execution

\[ \begin{array}{c}
\text{a} \\
\text{b} \\
\text{c} \\
\text{d} \\
\text{e} \\
\text{f}
\end{array} \rightarrow \begin{array}{c}
1 \\
2 \\
3 \\
4 \\
\text{a} \\
\text{b} \\
\text{c} \\
\text{d} \\
\text{e} \\
\text{f}
\end{array} \]

- Done dynamically
- Communication cost!
Objective Functions

- Bi-criterion problem: makespan vs communications
  - Makespan minimization: Never letting a processor being idle by constantly use data duplication $\rightarrow$ Potentially important communication cost
  - Communications minimization: Never create new duplicate of input files $\rightarrow$ Potentially a load-imbalance in the task repartition

- We consider here both cases: without communications what makespan can be expected and under makespan optimality constraint what communication cost can be expected?
Definition

An Assignment of a bipartite graph $G = (P, T, E)$ is a set $A$ of edges such that $\forall t_j \in T$, $\exists p_i$, $(p_i, t_j) \in A$. 

![Diagram showing assignment of bipartite graph](image-url)
Maximal Degree

Definition

Let $A$ be an assignment of a graph $G = (P, T, E)$. The **degree in** $A$ of a processor $p_i \in P$ (denoted $d_A(p_i)$) is the degree of $p_i$ in the induced sub-graph $G = (P, T, A)$. We denote $D(A) = \max_{p_i\in P} d_A(p_i)$ as the **maximal degree of an assignment**.

- $d_A(1) = 0$
- $d_A(2) = 3$
- $d_A(3) = 2$
- $d_A(4) = 1$

$D(A) = 3$

- The maximal degree of an assignment is the number of tasks processed by the most loaded processor.
Total Load Imbalance

Definition

Let $A$ be an assignment of a graph $G = (P, T, E)$. We denote

$$\text{Imb}(A) = \sum_{p_i \in P} d_A(p_i) - \left\lceil \frac{|T|}{|P|} \right\rceil$$

as the total load imbalance of an assignment.

The total load imbalance of an assignment is the maximal number of tasks to move to have perfect balancing.

$$d_A(1) = 3$$
$$d_A(2) = 3$$
$$d_A(3) = 0$$

$$\text{Imb}(A) = 1 + 1 = 2$$
Two Objectives

- Two possible objectives:
  - Makespan Objective: Find an assignment with minimal maximal degree
  - Communication Objective: Find an assignment with minimal total load imbalance
- If an assignment is perfect, both metrics are simultaneously minimized
- This can be generalized

**Theorem**

For each bipartite graph there is an assignment with minimal maximal degree and minimal total load imbalance.

- Can be found in polynomial time
Strategies

- **BestAssignment**:  
  - Static scheduling computing optimal assignment  
  - Dynamic correction mechanism which executes a local task at random or steals a task to the most loaded processor

- **Dynamic comparison strategies**:  
  - **Greedy**: Choose randomly a random task if any. Otherwise choose randomly a non-local task  
  - **Maestro**: Choose the local task that have the more chance to be non-locally executed. Otherwise choose randomly a non-local task
Settings

- **Heterogeneous settings**
  - Obtained from traces coming from a Hadoop cluster
  - Very different value of number of tasks number and standard deviation

| $|T| \leq 50$ | $|T| \in [50, 100]$ | $|T| \in [100, 250]$ | $|T| \in [250, 500]$ | $|T| \in [500, 1000]$ | $|T| \in [1000, 5000]$ | $|T| \geq 10000$ | Total |
|---|---|---|---|---|---|---|---|
| $NSD < 0.05$ | 49 (2, 41%) | 109 (5, 35%) | 123 (6, 04%) | 60 (2, 95%) | 39 (1, 92%) | 26 (1, 28%) | 406 (19, 94%) |
| $NSD \in [0.05, 0.1]$ | 50 (2, 46%) | 93 (4, 57%) | 61 (3, 00%) | 34 (1, 67%) | 23 (1, 13%) | 279 (13, 70%) |
| $NSD \in [0.1, 0.25]$ | 75 (3, 68%) | 110 (5, 40%) | 78 (3, 83%) | 25 (1, 23%) | 504 (24, 75%) |
| $NSD \in [0.25, 0.5]$ | 55 (2, 70%) | 68 (3, 34%) | 50 (2, 46%) | 17 (0, 83%) | 305 (14, 98%) |
| $NSD \in [0.5, 1]$ | 21 (1, 03%) | 44 (2, 15%) | 33 (1, 62%) | 18 (0, 88%) | 122 (5, 99%) |
| $NSD \geq 1$ | 5 (0, 25%) | 3 (0, 15%) | 16 (0, 79%) | 4 (0, 20%) | 14 (0, 69%) | 39 (1, 91%) | 93 (4, 57%) |
| Total | 90 (4, 42%) | 329 (16, 16%) | 651 (31, 97%) | 422 (20, 73%) | 322 (15, 82%) | 222 (10, 90%) | 2036 (100%) |
We focus only on communication metric.
\[
\frac{|T|}{|P|} = 2
\]
Heterogeneous Settings

- We focus only on communication metric.
- \( \frac{|T|}{|P|} = 10 \)
Table of Contents

1. Introduction
2. Communication-Avoiding Parallel Matrix Multiplication
3. Improving Data Locality of Map Tasks in MapReduce
4. Conclusion
Two Problems

- **Matrix Product:**
  - Core applications of many scientific computations (HPC oriented problem)
  - Replications are necessary for parallelism (shared input files) and done during execution
  - Replications are the objective function to minimize

- **MapReduce**
  - Paradigm of distributed computing (Big Data oriented problem)
  - Input files are replicated and placed on computers before the beginning of the execution
  - Replications are an input of the problem and have to be used in the best possible way
One Approach

1. Modelling:
   - Square and cube partitioning
   - Assignment, balls-into-bins, graph orientability

2. Theoretical work and algorithms:
   - Approximation algorithms (NRRP, 3D-NRRP, SFCP, 3D-SFCP)
   - Optimal algorithms (BestAssignment) and probabilistic behaviour

3. Test in "real" world:
   - Simulations and implementation
   - Simulations from traces
Static and Dynamic strategies

- We mainly look for **static** scheduling algorithm: scheduling made before execution
- Opposed to **dynamic** scheduling: scheduling made during execution
- Pros and cons:
  - Static is theoretically more effective, having more oversight
  - Static relies on models and prediction.
- Static is often considered as unreliable for practical concerns

With simple and small dynamic corrections, static strategies can keep their good theoretical performances even with (reasonably) misleading assumption
Perspectives

- Static approach works!
- Pursuit of studied problems:
  - Sparse matrix product (heterogeneous square partitioning)
  - Tensor contraction (hypercube partitioning)
  - Heterogeneous model for MapReduce
  - Shared input files MapReduce-like application
- Addition of dependencies concerns
Any question?