# Internship Jülich Supercomputer Center
## *Speech at the JSC*

Riyane SID-LAKHDAR

August the $22^{nd}$

**Abstract**

# Contents

# 1 Introduction

## 1.1 Motivation: avoid idle write time

To begin, let's consider a very common work-flow pattern: A loop where at each iteration, we do a computation and then write the generated results. And let's consider that each result is not used by the next iteration. Then the chronograph of our application would look like. The processor time would be split between computation and writing. But from the processor perspective each writing operation would be equivalent to a significant idle time.

The ideal we are aiming to would be a model where the idle writing time is parallelized Get ride of the idle time of the processor during write time. Use asynchronous write: maintain the processor working while the write is done by an independent thread.

## 1.2 Motivation: Optimize the Cube-remapper

In the previous pattern, some of you might have already recognized the working flow of the Cube-remapper. Indeed, our objective is to reimplement the

## 1.3 Problem definition

As you already know, when we deal with multithreading an parallelization, there are different performance issues that
This is why, after presenting the basis of our parallel implementation, we will try to model the behaviour of our system and fit the targeted hardware and software specifications.
Finally, using experimental assessments, we will identify different performance bottlenecks and propose different solutions. An experimental measurement of our implementations will be proposed and the gain brought compared to the state-of-the-art Cure-remapper will be described.

# 2 Environment and objective

## 2.1 The POSIX-based asynchronous I/O library

- POSIX standard library (for retro compatibility purposes). Deployed natively on most UNIX os distributions (GNU-Linux, MacOsX, freeBSD)
- The main thread:
- The writer threads (explain 1 thread per IO device. Using more threads would be harmful)

- The need to synchronization: control the memory footprint

### 2.2 The targeted objective: Cube-remapper

- Use this asynchronous approach to optimize the response time of the Cube-remapper.
- How to do that: present the Cube-remapper main loop + the compute/drop operation. The processor is stall during the write operation
- the objective is to get ride of this idle waiting time: parallelize the compute and the write operation.

## 3 Transition

So what we have presented is the environment we are going to work in. Our objective being to replace the synchronous writing used by default in the Cube-remapper by the synchronous one. What we expect from this approach is to get ride of the idle waiting time of each write where the processor is waiting for the write to be proceeded, knowing that it won't need it for the rest of its computation.

Now that we have set our objective and the tools to reach it, let's focus on the interesting part. How to handle this asynchronous model. What potential gain does this might bring to the cube-Remapper? What is the most suitable domain to use it. And what are the potential limitations or risks of using it.
Talking about potential drawbacks of this asynchronous approach may seem exaggerated at this step. But we will see that a miss-usage of this approach may easily lead to a very poor improvement or even a downgrade of the performance, mainly due to the synchronization overhead.

To answer to all these questions, we will try to model the behaviour of our pattern in both synchronous and asynchronous model. We will try to enhance our asynchronous model regarding experimental assessment and identify the parameters of the system that influence it. And finally find the domain of these parameters that allow to have an maximum gain (compared to the synchronous model).
In order to better understand the theoretical behaviour of the asynchronous I/O library, we have first designed a custom testbed. This implementation simulates the the Cube-Remapper pattern in different situations and different internal and external workloads. This implementation allows to get ride of the potential interference between the real Cube-remapper and the asynchronous I/O. Indeed, the performance of a multithreaded I/O is dependant of low level phenomenon at live memory and cache level. Thus it might be sensitive to numerous and unexpected external interference.

# 4 The asynchronous-writing approach

## 4.1 First approach: simplified theoretical model

****** After presenting the theoretical and experimental slides *********
As you may notice, this model might still be improved to fit more accurately the experimental assessment. And this is what we do through different steps to reach this second one (more details about this second model and the reasons beneath this choice might be found on my thesis...). However, let's for now keep it simple.

From these theoretical and experimental discussion, two main need to be kept in mind.
First, we have shown experimentally the validity of our model. Indeed, our modes has represented the inflexion points of the asynchronous pattern. It identifies the domain where the asynchronous pattern is may significantly vary and its asymptotic behaviour.

On the other hand, our model allows us to identify the domain where the gain compared to the synchronous case is the most significant. If we translate this to the cube remapper, this means that we have identified the trade of between compute and write time that allows the most significant improvement of our strategy.

****** Present the different variables of our test bed that we can tune: compute time, nb IoDevice, buffer size, ....

## 4.2 Multiple parallel I/O device

# 5 Transition

What is important to keep in mind from all this theoretical part is that asynchronous writing might indeed bring a significant improvement to the performances of our pattern. But this can't be done without a careful tuning of the pattern. Using asynchronous writing out of the previously spotted domain might lead to no improvement at all. Given the engineering effort and the overhead due to synchronization, this might even be considered as harmful.
******* The experimental results have been obtained thanks to a simulation test bed. We have simulated our pattern and replaced the computation by a system sleep operation******

All what we have presented till now is the theory beneath the usage of asynchronous writing in a scientific computation pattern. Some of you may already wonder why did we need to implement a test bed to simulate the behaviour of the considered pattern. Why didn't we simply use the Cube-remapper, which already implements this pattern. Well the reason for that is simple. We did not want our theoretical results to be biased or influenced by the remapper. And our next implementations and results on the Cube-remapper proves us right.

In the next section, we first briefly introduce our version of the Cube-remapper based on asynchronous-writings. Then we identify the different parts of the remapper that interfere with the asynchronous-writing. We present different optimizations of the remapper to soften these interferences. Finally, we present an experimental assessment of the remapper and evaluate the efficiency of our implementation.

# 6 The Cube-remapper optimization

## 6.1 Basic asynchronous I/O

The first graphe represent an assessment of the write operation time within the Cube re-mapper. As you can notice, the asynchronous strategy allows to bring a significant improvment to the write operation. Indeed, this strategy consist in simply queueing a request write request. The request is executed in parallel without processor stall.
This gain that you see is what we expect for the total time of the cube remapper.

However, as you can see in this second graph the total time