On the Impact of Asynchronous I/O on the *Cube re-mapper* at HPC Scale

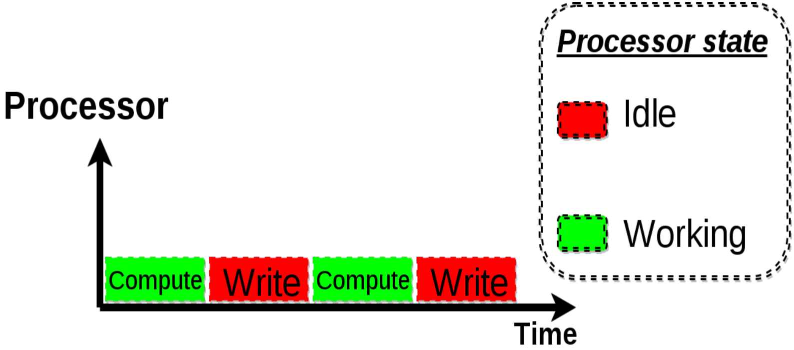
Presented by **Riyane SID LAKHDAR**,
Supervised by **Dr. Pavel SAVIANKOU**

Objective: Outperform the *Cube re-mapper*

Cube re-mapper:

- Performance profiling software
- Designed for HPC-specific applications
- Developed by the Jülich Supercomputer Center laboratory (*Scalasca* project)

Global environment: the *Cube re-mapper* pattern



Motivation: avoid idle write time

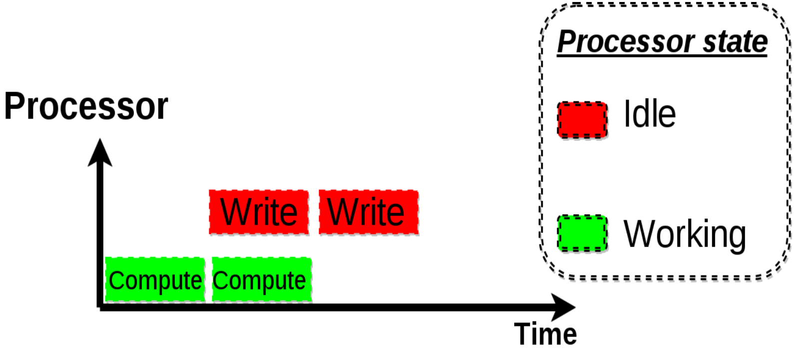
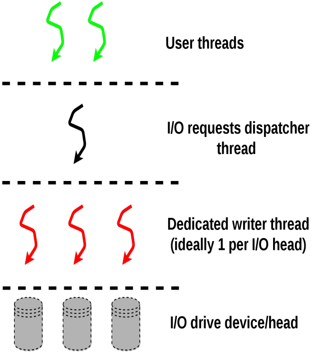


Table of Contents

Table of Contents

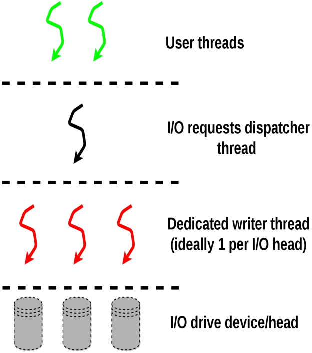
The POSIX-based asynchronous I/O (AIO.h) library



AIO.h:

- POSIX standard library
- Distributed on most UNIX OS (e
  GNU-Linux, MacOsX)
- Emulated for windows

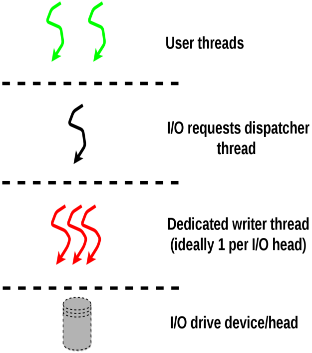
The POSIX-based asynchronous I/O (AIO.h) library



Limitations of the AIO.h

- Memory footprint might explode. Hence, RAM swap.

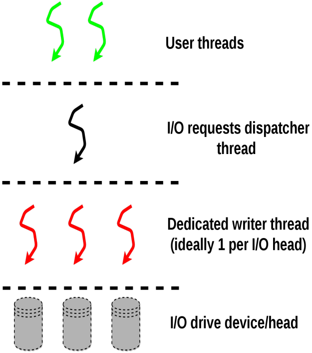
The POSIX-based asynchronous I/O (AIO.h) library



Limitations of the AIO.h

- Memory footprint might explode. Hence, RAM swap.
- Possible contention on the I/O devices.

The POSIX-based asynchronous I/O (AIO.h) library



Why this choice?

- Minimize the engineering effort
- The AIO.h is tuned to fit:
  - I/O access pattern
  - Hardware specification

The *Cube re-mapper*

```
void mainCubeRemapper
{
    Cube* inCube = new Cube(input);

    for (int i=0; i<nbMetric;++i)
    {
        File* result = openFile("w");
        compute(inCube, i, &bufer);
        write(buffer, result);
    }
}
```

The *Cube re-mapper* (asynchronous I/O version)

```
void mainCubeRemapper
{
    Cube* inCube = new Cube(input);

    for (int i=0; i<nbMetric;++i)
    {
        File* result = openFile("w");
        compute(inCube, i, &bufer);
        asynchronous_write(buffer, result);
    }
}
```

The *Cube re-mapper* (asynchronous I/O version)

```
void mainCubeRemapper
{
    Cube* inCube = new Cube(input);

    for (int i=0; i<nbMetric;++i)
    {
        File* result = openFile("w");
        compute(inCube, i, &bufer);
        asynchronous_write(buffer, result);
    }
    wait_asynchronous_write();
}
```

The *Cube re-mapper* (asynchronous I/O version)

```
void mainCubeRemapper
{
    Cube* inCube = new (
    
    for (int i=0; i<nbMe
    {
        File* result = (
        compute(inCube,
        asynchronous_write(buffer, result);
    }
    wait_asynchronous_write();
}
```

The asynchronous I/O approach

- Reduce processor stall

The *Cube re-mapper* (asynchronous I/O version)

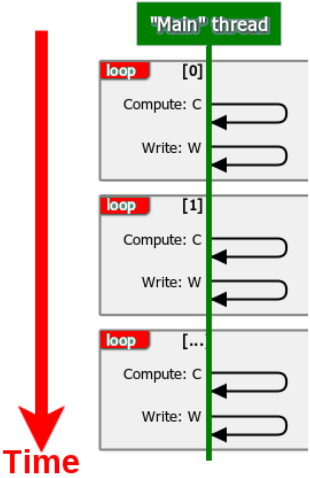
```
void mainCubeRemapper
{
    Cube* inCube = new (

    for (int i=0; i<nbMe
    {
        File* result = c
        compute(inCube, i, abuffer);
        asynchronous_write(buffer, result);
    }
    wait_asynchronous_write();
}
```

The asynchronous I/O approach

- Reduce processor stall
- Benefit from data distribution

Table of Contents

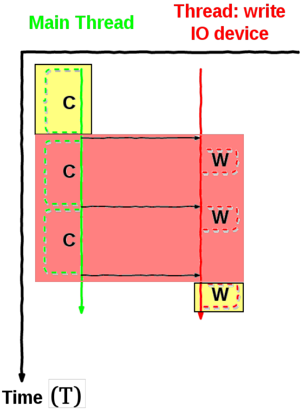
Synchronous *Cube re-mapper* model



Synchronous I/O model

- Current version of the *Cube re-mapper*
- Benchmark for the study
- $T_{synchronous} = n * (C + W)$

Simplified asynchronous model (constant *"Compute"/"Write"* time)



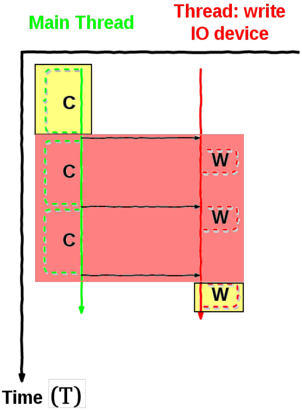
Asynchronous I/O model

$$T = C + W + (n-1) * max(C, W)$$

$$T \overset{\text{if C « W}}{\approx} n * W + C$$

$$T \overset{\text{if C » W}}{\approx} n * C + W$$

Simplified asynchronous model (constant *"Compute"/"Write"* time)



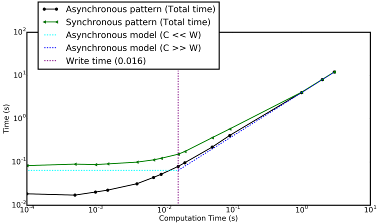
Asynchronous I/O model

$$T = C + W + (n-1) * max(C, W)$$

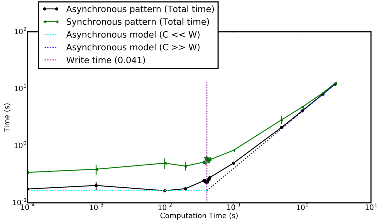
$$T \overset{\text{if C « W}}{\approx} n * W + C$$

$$T \overset{\text{if C » W}}{\approx} n * C + W$$

Simplified model assessment



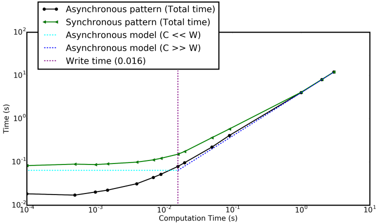
(a) *Intel Core* CPU i7-6700 (Workstation)



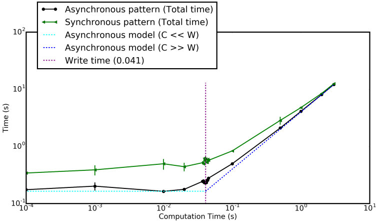
(b) *Intel Xeon* CPU E52680v3 (HPC JURECA)

- Experimental improvement brought by the asynchronous I/O
- Maximum improvement for $C \sim W$
- Potential inaccuracy. Hence, the model 2 (*cf* report)

Simplified model assessment



(a) *Intel Core* CPU i7-6700 (Workstation)



(b) *Intel Xeon* CPU E52680v3 (HPC JURECA)

- Experimental improvement brought by the asynchronous I/O
- Maximum improvement for $C \sim W$
- Potential inaccuracy. Hence, the model 2 (*cf* report)

Table of Contents

Basis of the custom solution

The POSIX-based asynchronous I/O

Modeling the gain and the parameters of the solution

Improving the *Cube re-mapper*

Basic asynchronous implementation

Custom enhancements

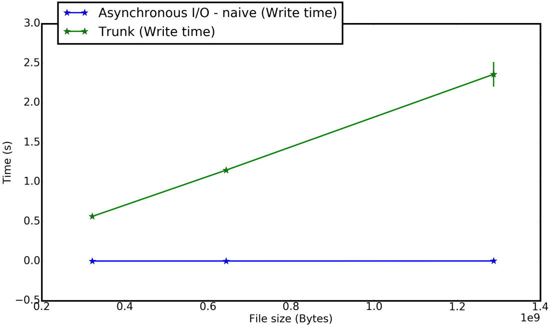
Conclusion and future work

Experimental set-up

Evaluation of the custom implementations of the *Cube re-mapper* using:

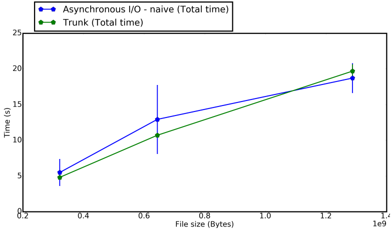
- Real-life input file (**NAS parallel benchmark**):
  - 65535 threads, 0.38 GiB
  - 131071 threads, 0.62 GiB
  - 262143 threads, 1.28 GiB
- Real metrics (ex: CPU time, MPI communication) from HPC executions
- Realistic ratio $\frac{C}{W} \equiv 2$

Basic asynchronous I/O implementation

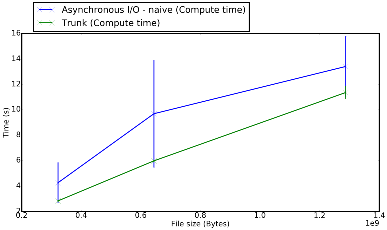


- Significant improvement in the *"write"* time
- Are we done?

Basic asynchronous I/O implementation



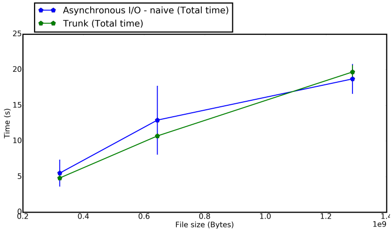
(a) *"Total"* time



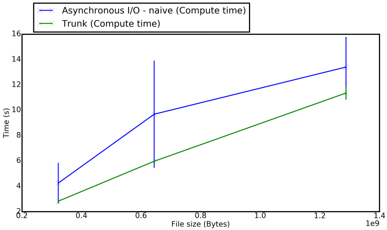
(b) *"Compute"* time

- Performance loss (due to *"Compute"*)
- Uncertainty increase (due to *"Compute"*)

Basic asynchronous I/O implementation



(a) *"Total"* time



(b) *"Compute"* time

- Performance loss (due to *"Compute"*)
- Uncertainty increase (due to *"Compute"*)

Thread-scheduling issue

Threads belong to the same process:

- Mostly scheduled on **same CPU core** (for cache proximity)
- *"compute"* thread **execution is delayed**

Solution: to pin threads on different CPU-cores

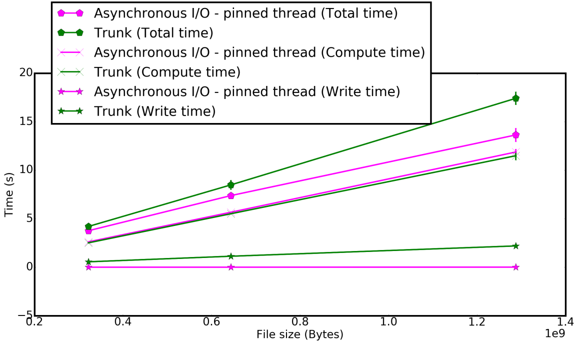
Thread-scheduling issue

Threads belong to the same process:

- Mostly scheduled on **same CPU core** (for cache proximity)
- *"compute"* thread **execution is delayed**

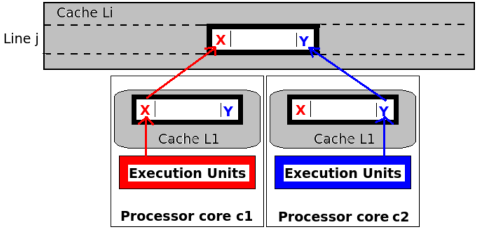
Solution: to pin threads on different CPU-cores

Thread scheduling solution: pin threads



- Lighten interferences with the *"Compute"* thread
- Reduce the *"Compute"* time

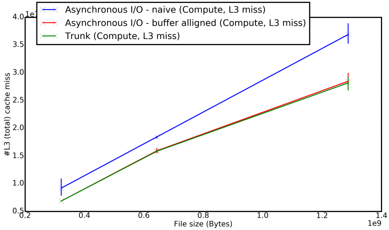
False-sharing issue



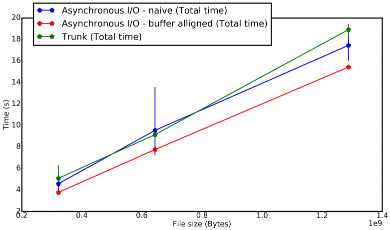
Consequence

- Back and forth invalidation at each address access
- High occurrence frequency $\Rightarrow$ significant impact

Lighten the impact of false-sharing



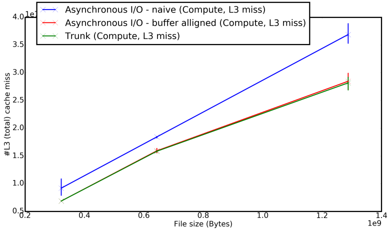
(a) L3 (total) cache miss



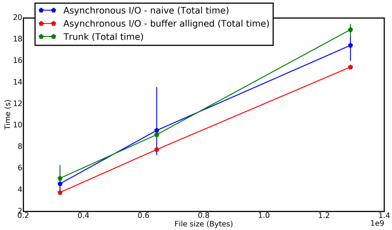
(b) *Cube re-mapper "Total"* time

- Align buffer address to cache line
- Reduce cache-miss rate

Lighten the impact of false-sharing



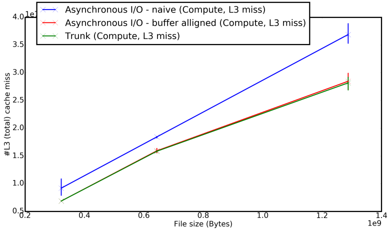
(a) L3 (total) cache miss



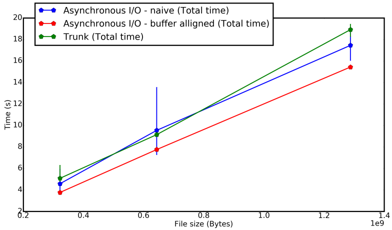
(b) *Cube re-mapper "Total"* time

- Align buffer address to cache line
- Reduce cache-miss rate

Lighten the impact of false-sharing



(a) L3 (total) cache miss



(b) *Cube re-mapper "Total"* time

- Align buffer address to cache line
- Reduce cache-miss rate

Improve dynamic memory allocation usage

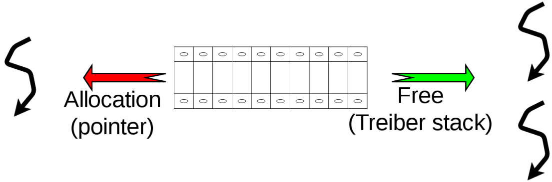


Figure: Free dynamic memory pool

Custom memory allocator

- Heap managed at user-level
- Independent *"allocation"* (Compute) and *"free"* (Write)
- Reduce contention on "Free" pool (*"Treiber"* stack)

Improve dynamic memory allocation usage

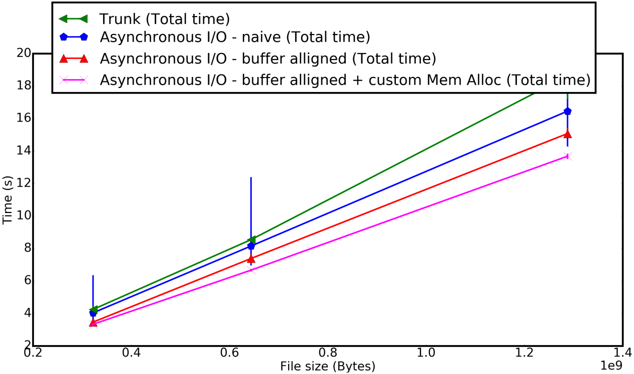


Figure: *Cube re-mapper* full-fledged assessment

Table of Contents

Conclusion

- Studied the theoretical model of asynchronous I/O (AIO) within the *Cube re-mapper* pattern.
- Implemented a prototype version (candidate for production) of the *Cube re-mapper*
  - Identified runtime perturbation created by the asynchronous I/O write
  - Proposed and evaluated solutions

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement (between 30 and 60%).

Conclusion

- Studied the theoretical model of asynchronous I/O (AIO) within the *Cube re-mapper* pattern.
- Implemented a prototype version (candidate for production) of the *Cube re-mapper*
  - Identified runtime perturbation created by the asynchronous I/O write
  - Proposed and evaluated solutions

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement (between 30 and 60%).

Conclusion

- Studied the theoretical model of asynchronous I/O (AIO) within the *Cube re-mapper* pattern.
- Implemented a prototype version (candidate for production) of the *Cube re-mapper*
  - Identified runtime perturbation created by the asynchronous I/O write
  - Proposed and evaluated solutions

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement (between 30 and 60%).

Conclusion

- Studied the theoretical model of asynchronous I/O (AIO) within the *Cube re-mapper* pattern.
- Implemented a prototype version (candidate for production) of the *Cube re-mapper*
  - Identified runtime perturbation created by the asynchronous I/O write
  - Proposed and evaluated solutions

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement (between 30 and 60%).

Future work

Full parallelization of the pattern:

- Multiple concurrent *"compute"* threads
- Expected fit with our current solution:
  - Optimal data distribution/synchronization
  - Allows further scalability evaluation of the current solution

Thank you!