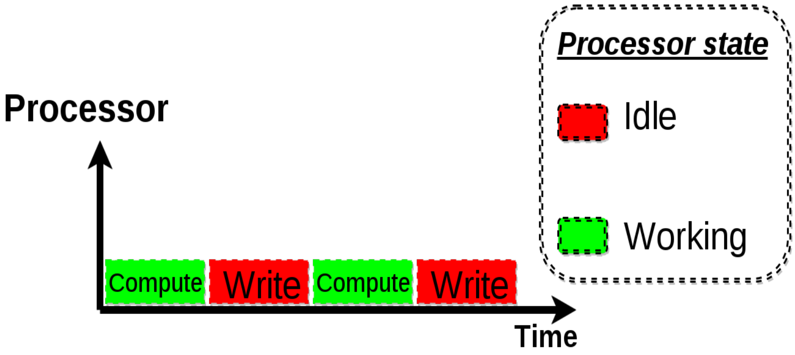
Mitglied der Helmholtz-Gemeinschaft

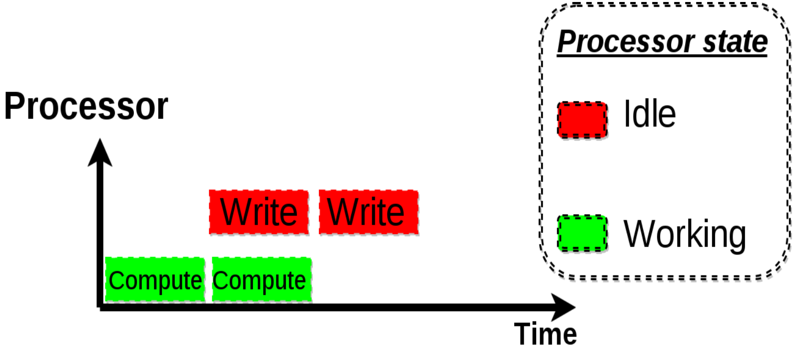
On the Impact of Asynchronous I/O on the *Cube re-mapper* at HPC Scale

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

Motivation: avoid idle write time



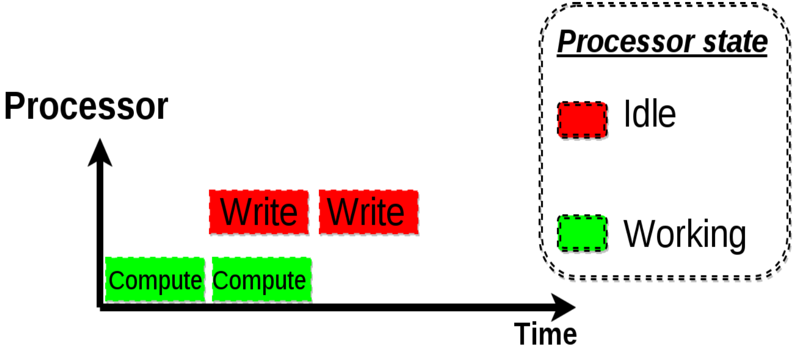
Motivation: avoid idle write time



Target software optimization

The *Cube re-mapper*

Motivation: avoid idle write time



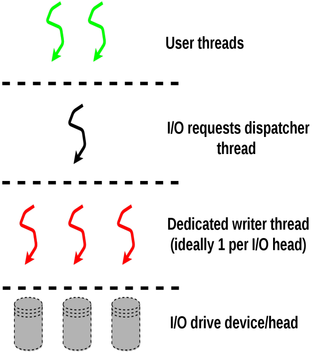
Target software optimization

The *Cube re-mapper*

Table of Contents

Table of Contents

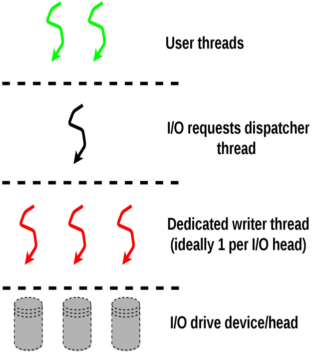
The POSIX-based asynchronous I/O library (AIO.h)



Asynchronous I/O library (AIO.h)

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

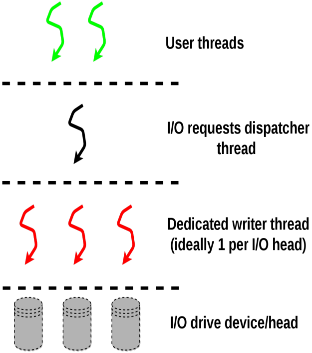
The POSIX-based asynchronous I/O library (AIO.h)



Limitations of the AIO.h

- Memory footprint might explode $\Rightarrow$ RAM swap
- Possible contention on I/O device

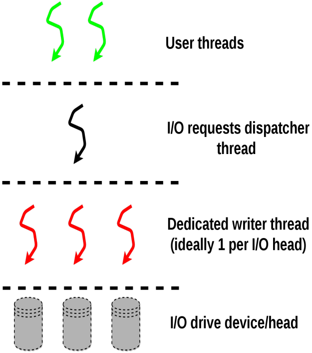
The POSIX-based asynchronous I/O library (AIO.h)



Limitations of the AIO.h

- Memory footprint might explode $\Rightarrow$ RAM swap
- Possible contention on I/O device

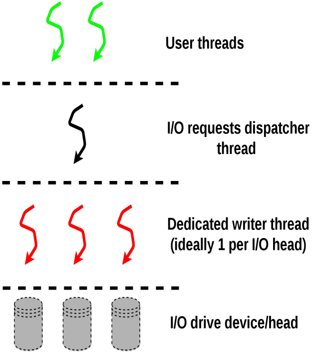
The POSIX-based asynchronous I/O library (AIO.h)



Why this choice?

- Minimize the engineering effort
- Tuned to fit:
  - I/O access pattern
  - Hardware specification

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



Why this choice?

- Minimize the engineering effort
- 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, i, &bufer);
        asynchronous_write(buffer, result);
    }
    wait_asynchronous_write();
}
```

The asynchronous choice

- Reduce processor stall

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

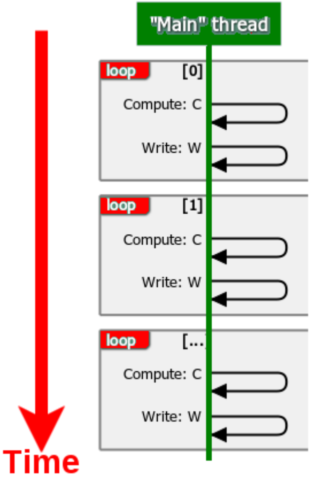
```
void mainCubeRemapper
{
    Cube* inCube = new (
    
    for (int i=0; i<nbMe
    {
        File* result = openFile( "w" );
        compute(inCube, i, &bufer);
        asynchronous_write(buffer, result);
    }
    wait_asynchronous_write();
}
```

The asynchronous choice

- Reduce processor stall
- Benefit from data distribution

Table of Contents

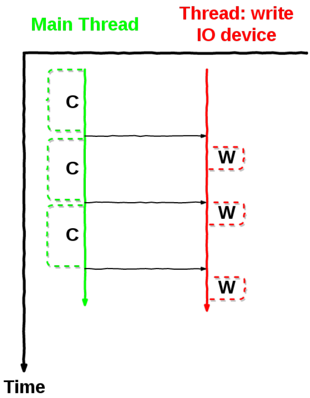
Current synchronous model



Synchronous I/O model

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

Simplified asynchronous I/O model

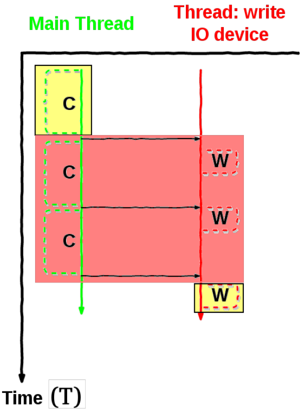


Asynchronous I/O model

Assumptions:

- Constant writing time W of each buffer
- Constant computation time C

Simplified asynchronous I/O model



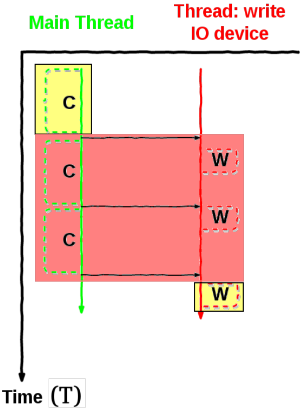
Asynchronous I/O model

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

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

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

Simplified asynchronous I/O model



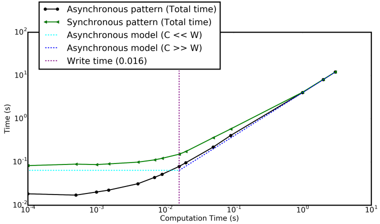
Asynchronous I/O model

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

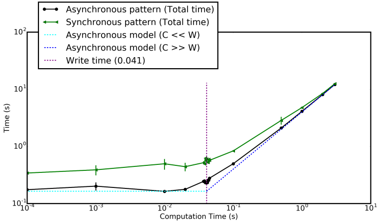
$$T \overset{\text{if C « W}}{\approx} n * W + C$$

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

Simplified model assessment



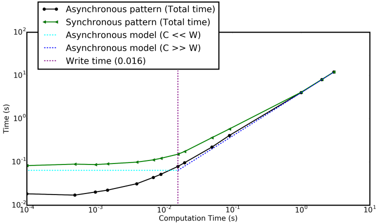
(a) *Intel Core* CPU i7-6700 (Workstation)



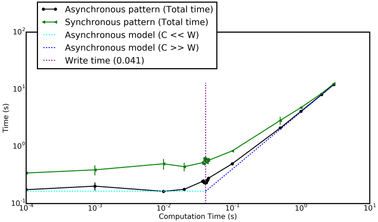
(b) *Intel Xeon* CPU E52680v3 (HPC JURECA)

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

Simplified model assessment



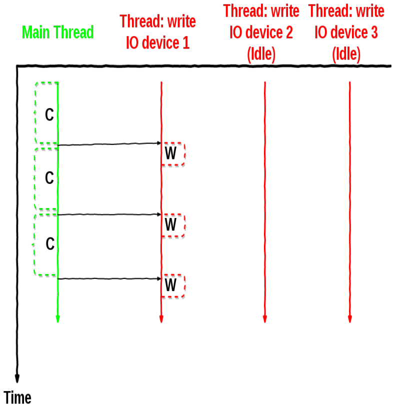
(a) *Intel Core* CPU i7-6700 (Workstation)



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

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

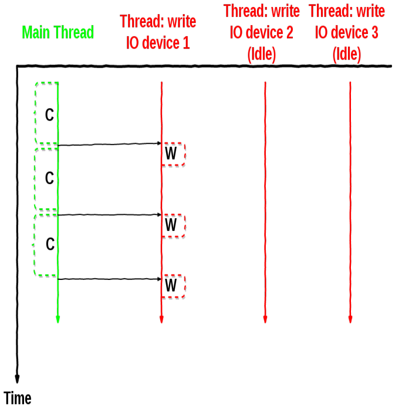
Multiple I/O devices: case $C >> W$



Theoretical model

- $T(N_{io}) \overset{\text{if C » W}}{\approx} n * C + W$
- Useless additional I/O

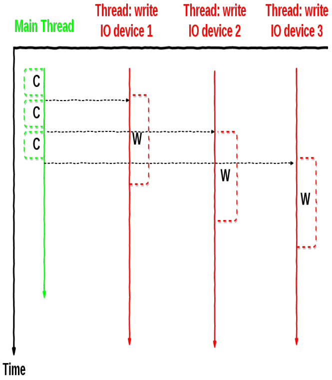
Multiple I/O devices: case $C >> W$



Theoretical model

- $T(N_{io}) \overset{\text{if C » W}}{\approx} n * C + W$
- Useless additional I/O

Multiple I/O devices: case $C << W$

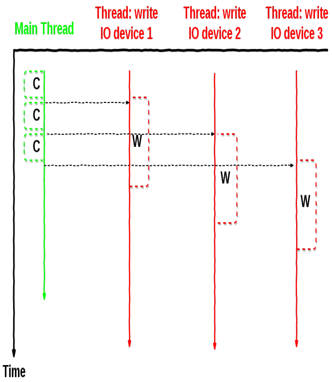


Theoretical model

$$T(N_{io}) \overset{\text{if C « W}}{\approx} C + W + (n-1) * max(\frac{W}{N_{io}}, C)$$

$$T(N_{io}) \overset{\text{if C « }\frac{W}{N_{io}}}{\approx} C + W + (n-1) * \frac{W}{N_{io}}$$

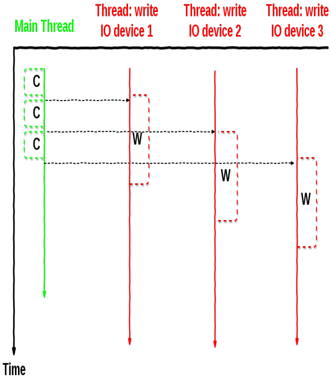
Multiple I/O devices: case $C << W$



Real-life execution

- Relevant additional I/O
- Complicated implementation

Multiple I/O devices: case $C << W$



Real-life execution

- Relevant additional I/O
- Complicated implementation

Table of Contents

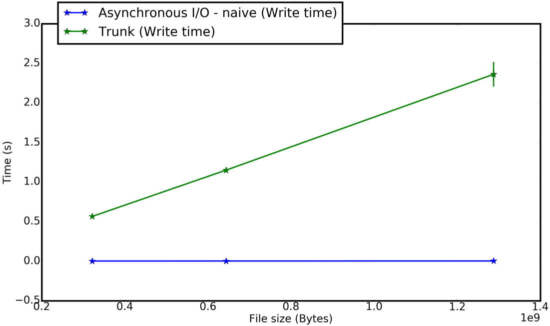
Experimental set-up

Evaluation of custom implementations of *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

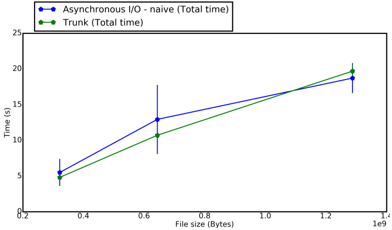
- Real metrics (ex: CPU time, MPI communication) from HPC execution
- Realistic ratio $\frac{C}{W} \equiv 2$

Basic asynchronous I/O implementation

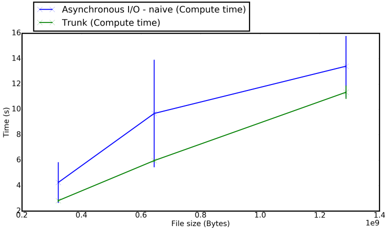


- Significant improvement in the *"write"* time
- Are we done?

Basic asynchronous I/O implementation



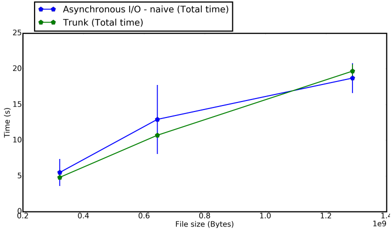
(a) *"Total"* time



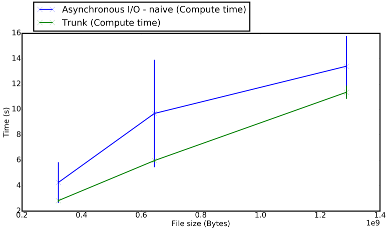
(b) *"Compute"* time

- 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 ⇒

- scheduled mostly on same CPU core (for cache proximity)
- Delay the *"compute"* thread execution

Solution: to pin threads on different CPU-cores

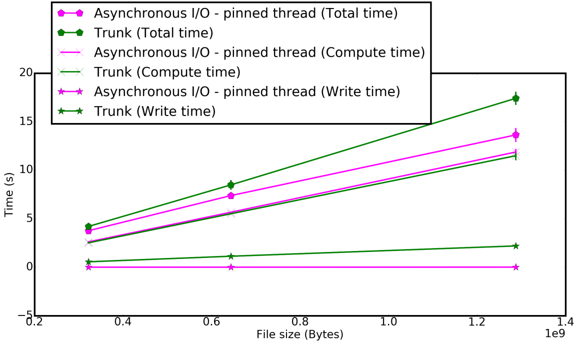
Thread-scheduling issue

Threads belong to the same process ⇒

- scheduled mostly on same CPU core (for cache proximity)
- Delay the *"compute"* thread execution

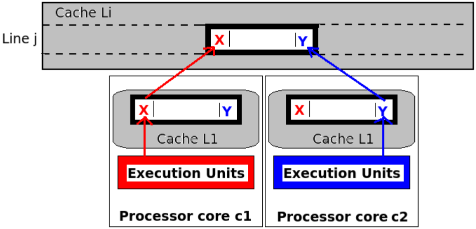
Solution: to pin threads on different CPU-cores

Thread scheduling solution: pin threads



- Lighten interference with *"Compute"* thread
- Reduce *"Compute"* time

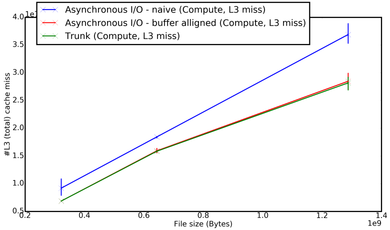
False-sharing issue



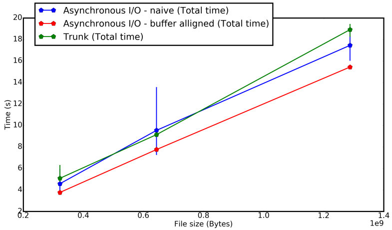
Consequence

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

Lighten the impact of false-sharing



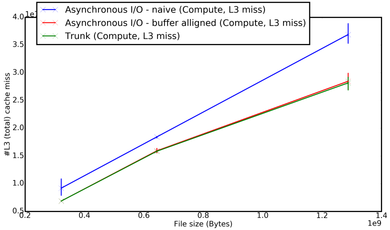
(a) L3 (total) cache miss



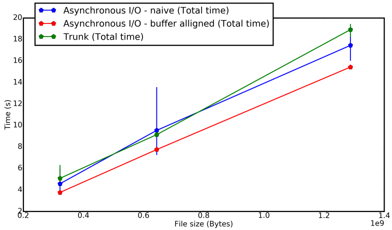
(b) *Cube re-mapper "Total"* time

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

Lighten the impact of false-sharing



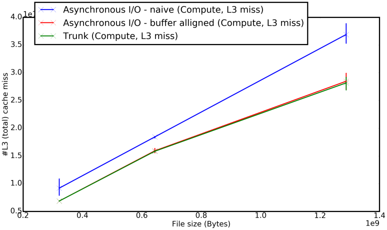
(a) L3 (total) cache miss



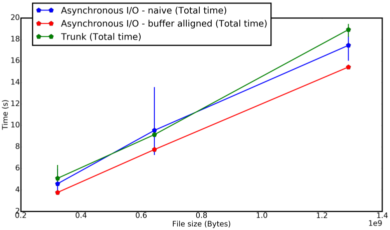
(b) *Cube re-mapper "Total"* time

- 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

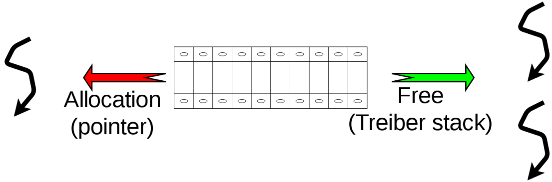


Figure: Free dynamic memory pool

Custom memory allocator

- User-level managed heap
- Independent *"allocation"* (Compute) and *"free"* (Write)
- Reduced contention on "Free" pool (*"Treiber"* stack)

Improve dynamic memory allocation usage

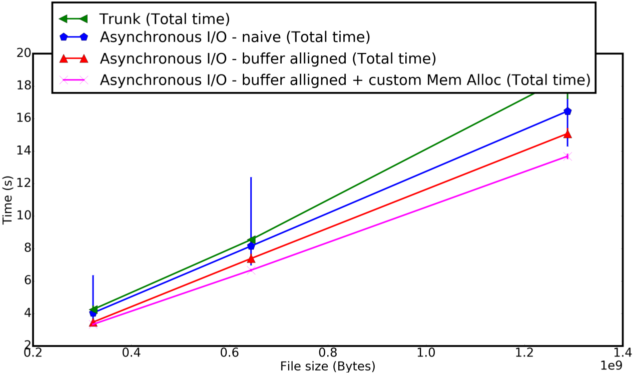


Figure: *Cube re-mapper* full-fledged assessment

Table of Contents

Conclusion

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

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement

Conclusion

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

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement

Conclusion

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

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement

Conclusion

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

Our most enhanced custom implementation of the *Cube re-mapper* allows a significant improvement

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 current solution

Thanks for your attention!