The MPI+CUDA Gaia AVU–GSR Parallel Solver towards next-generation Exascale Infrastructures

V. Cesare[1], U. Becciani[1], A. Vecchiato[2], M. G. Lattanzi[3], F. Pitari[4], M. Raciti[5], G. Tudisco[6], M. Aldinucci[7], B. Bucciarelli[3]

(1) INAF-OACT+ICSC; (2) INAF-OATO+ICSC; (3) INAF-OATO; (4) CINECA; (5) UniCT; (6) INAF-OACT; (7) UniTO+ICSC

INAF USC VIII-Calcolo Critico

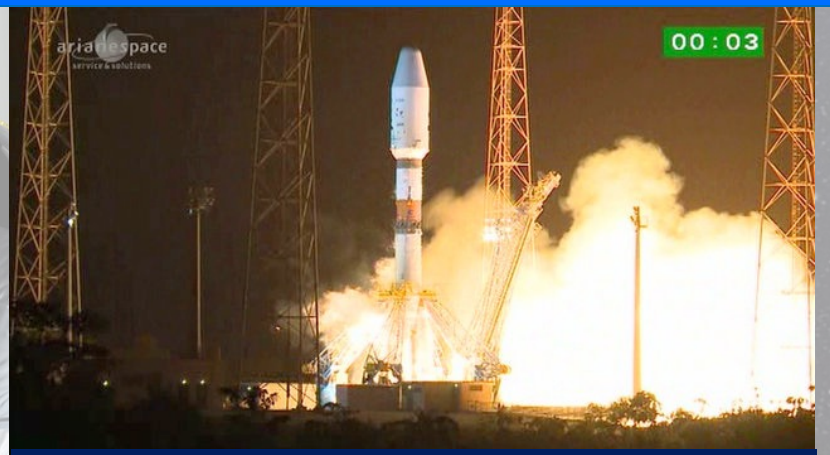
June 15th 2023

Table of contents

1. The ESA Gaia Mission

Gaia AVU-GSR solver target:

Derivation of positions and proper motions of ~10^8 stars (primary stars) in the Milky Way observed with the Gaia satellite, with a [10,100] μas accuracy.



Gaia launch from Guyana Space Center– ESA/CNES/Arianespace

Gaia spacecraft - ESA-D. Ducros (2013)

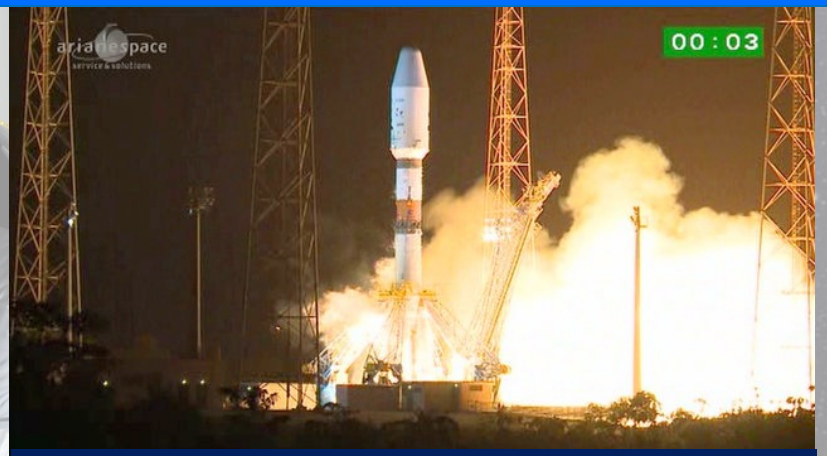
1. The ESA Gaia Mission

Gaia AVU-GSR solver target:

Derivation of positions and proper motions of ~10^8 stars (primary stars) in the Milky Way observed with the Gaia satellite, with a [10,100] μas accuracy.

The Gaia mission:

- **Developed by:** European Space Agency (ESA)
- **Duration:** Dec 19th 2013 – 2018 (extended so far to 2025-2030).
- **Data Release 3:** Published on June 13th 2022
- **Objectives:**
  - Astrometry: map of the positions and the proper motions of the stars in our galaxy
  - Origin and evolution of the Milky Way
  - Test of theories of gravity
- **Website:** https://sci.esa.int/web/gaia



Gaia launch from Guyana Space Center– ESA/CNES/Arianespace

Gaia spacecraft - ESA-D. Ducros (2013)

2. The Gaia AVU-GSR parallel solver

$$\mathbf{A} \times \boldsymbol{x} = \boldsymbol{b}$$

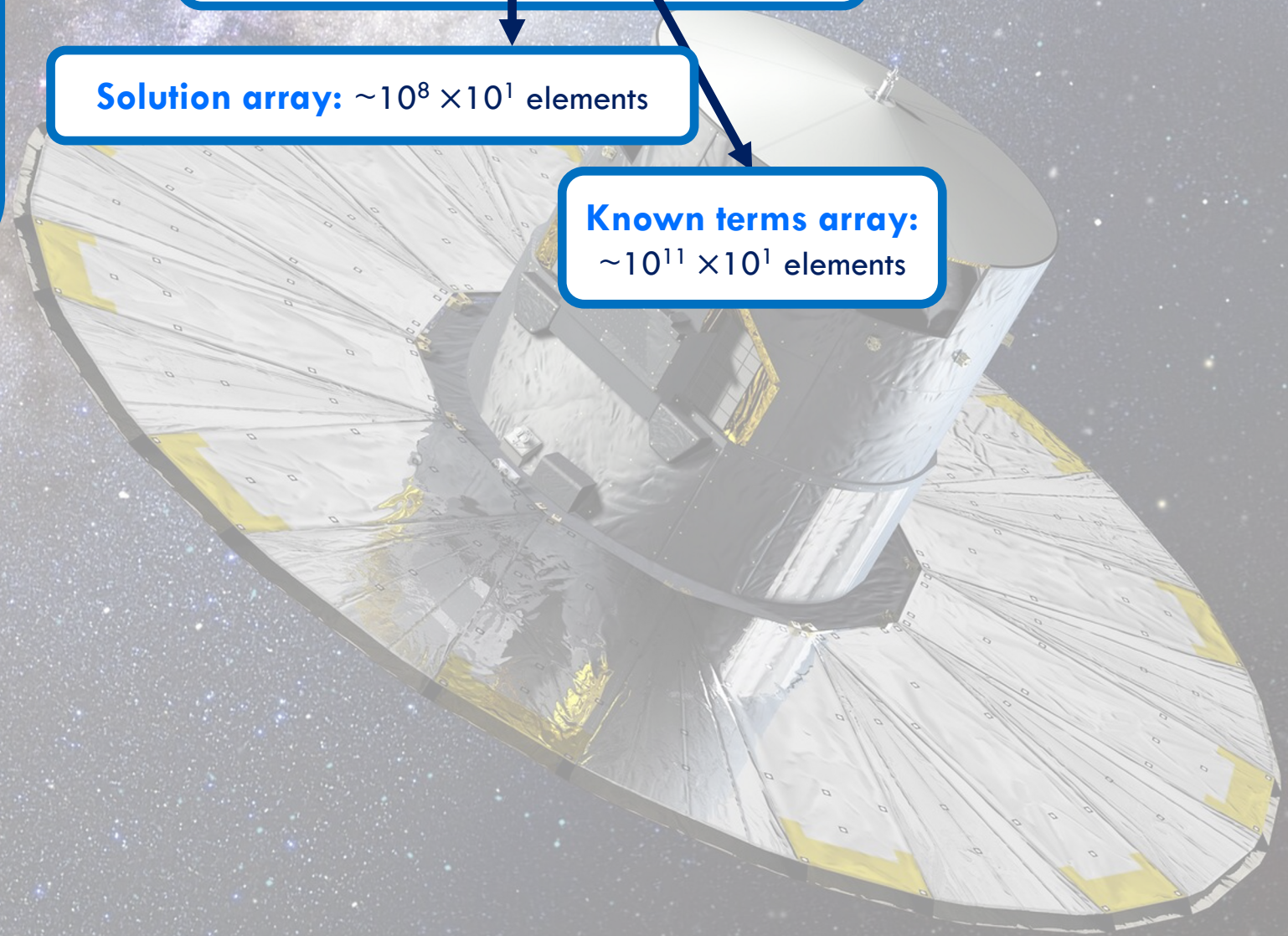
Coefficient matrix:

- Large and sparse ($N_{obs} \times N_{unk} \simeq 10^{11} \times 10^{8}$ elements)
- Computation with a dense matrix $\mathbf{A_d}$ ($\sim 10^{11} \times 10^{1}$ elements)

Solution array: $\sim 10^{8} \times 10^{1}$ elements

Known terms array: $\sim 10^{11} \times 10^{1}$ elements

10-100 TB of memory: **Big Data problem**



Becciani et al. (2014)

Coefficient matrix:

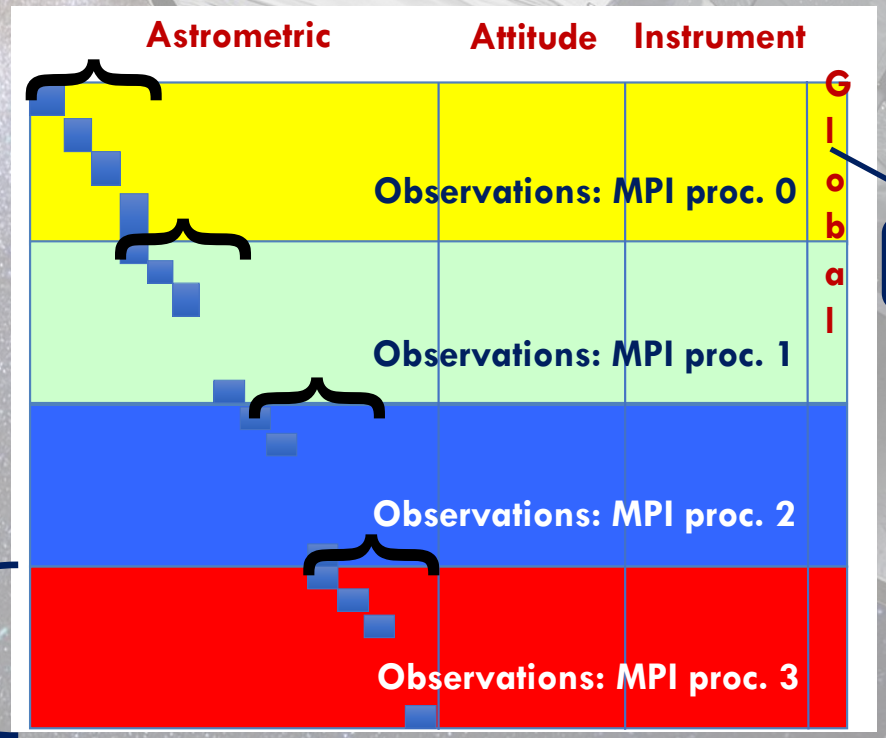
- Large and sparse ($N_{obs} \times N_{unk} \simeq 10^{11} \times 10^{8}$ elements)
- Computation with a dense matrix $\mathbf{A_d}$ ($\sim 10^{11} \times 10^{1}$ elements)

$$\mathbf{A} \times \boldsymbol{x} = \boldsymbol{b}$$

Solution array: $\sim 10^{8} \times 10^{1}$ elements

Known terms array: $\sim 10^{11} \times 10^{1}$ elements

10-100 TB of memory: **Big Data problem**



PPN γ

$N_{obs} \sim 10^{11}$

OpenMP threads

$N_{unk} \sim 10^{8}$

Becciani et al. (2014)

Coefficient matrix:

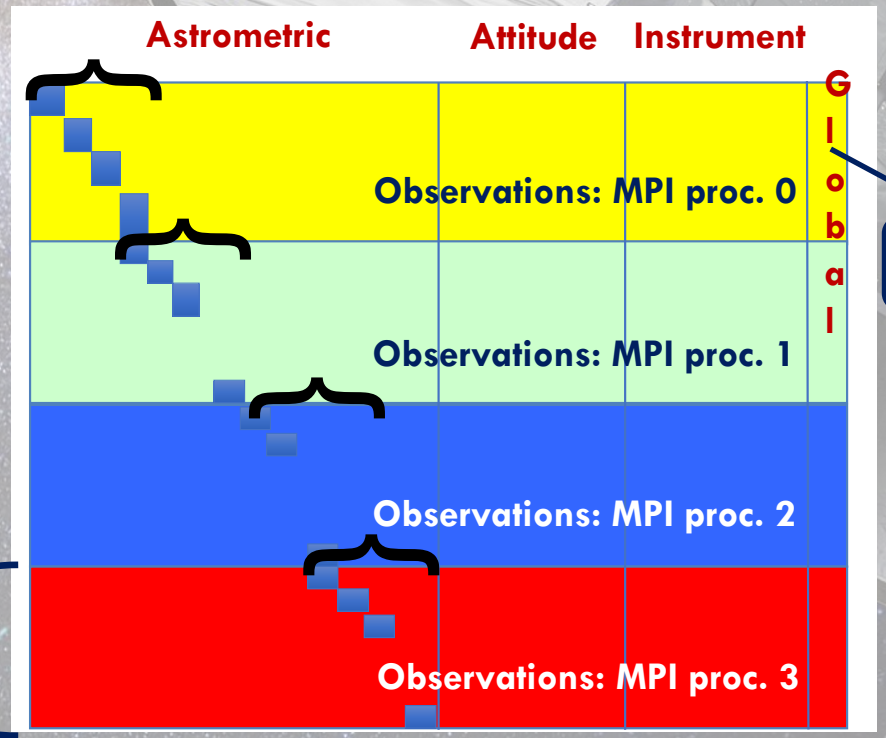
- Large and sparse ($N_{obs} \times N_{unk} \simeq 10^{11} \times 10^{8}$ elements)
- Computation with a dense matrix $\mathbf{A_d}$ ($\sim 10^{11} \times 10^{1}$ elements)

$$\mathbf{A} \times x = b$$

Solution array: $\sim 10^{8} \times 10^{1}$ elements

Known terms array: $\sim 10^{11} \times 10^{1}$ elements

10-100 TB of memory: **Big Data problem**



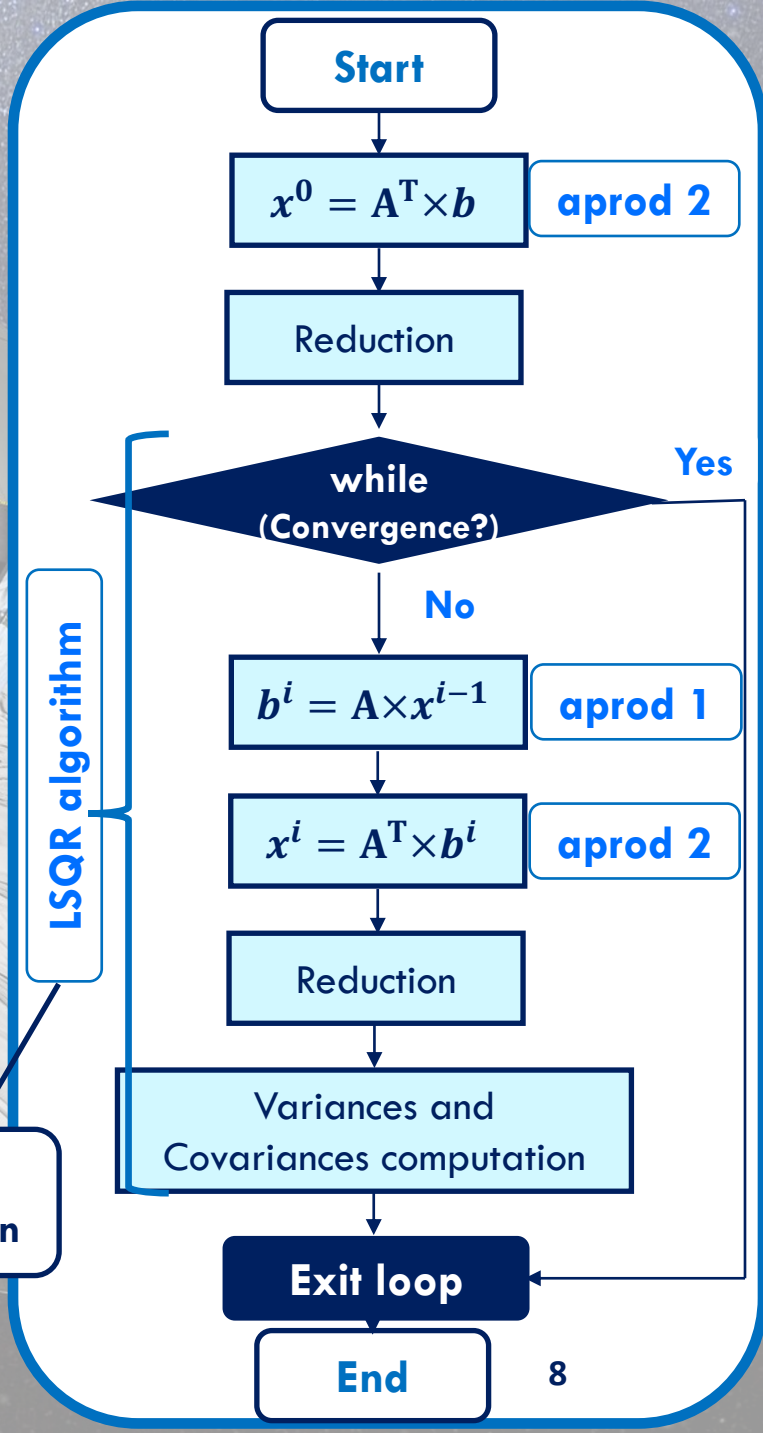
PPN γ

$N_{obs} \sim 10^{11}$

OpenMP threads

$N_{unk} \sim 10^{8}$

> 90% calculation



Becciani et al. (2014)

2.1 Computational problem

- The calculation requires a parallelization over many THIN-like nodes (256-512 GB of memory).
- Application dominated by computation and minimal MPI communications.
- Maximal occupancy of each node resources (between MPI processes and OpenMP threads).
- Mean iteration time: ~4 s
- Typical number of iterations for convergence: ~150000

2.1 Computational problem

- The calculation requires a parallelization over many THIN-like nodes (256-512 GB of memory).
- Application dominated by computation and minimal MPI communications.
- Maximal occupancy of each node resources (between MPI processes and OpenMP threads).
- Mean iteration time: ~4 s
- Typical number of iterations for convergence: ~150000

Total iteration time:

~1 week

Non optimal for the future Gaia Data Releases.

2.1 Computational problem

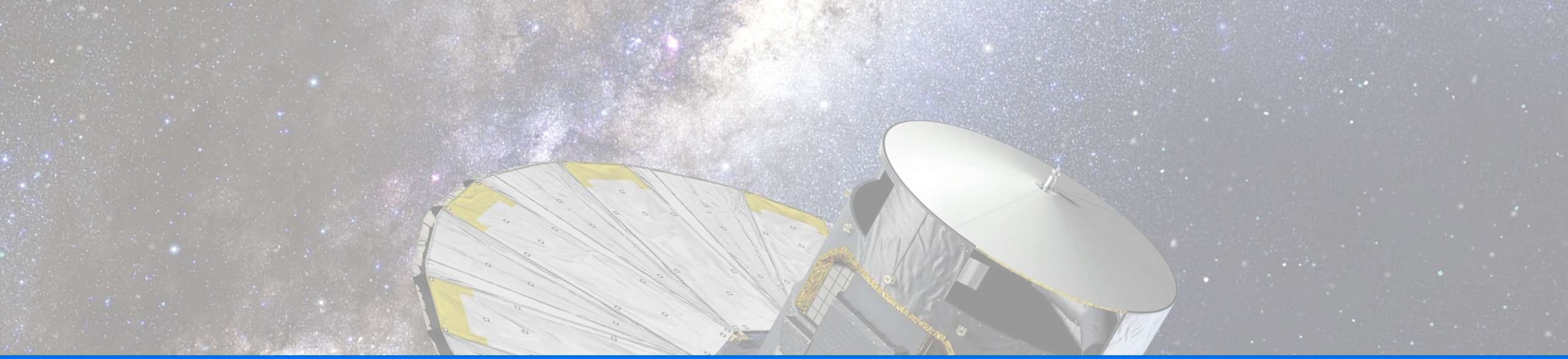
- The calculation requires a parallelization over many THIN-like nodes (256-512 GB of memory).
- Application dominated by computation and minimal MPI communications.
- Maximal occupancy of each node resources (between MPI processes and OpenMP threads).
- Mean iteration time: ~4 s
- Typical number of iterations for convergence: ~150000

Total iteration time:

~1 week

Non optimal for the future Gaia Data Releases.

GPU porting of the code



3. The OpenACC and CUDA porting

References:

Cesare V., et al., accepted for publication in the Publications of the Astronomical Society of the Pacific

Cesare V., et al., 2022b, INAF Technical Reports 164

Cesare, V. et al., 2022c, Astronomy and Computing, 41, 100660

Cesare V., et al., 2022a, INAF Technical Reports 163

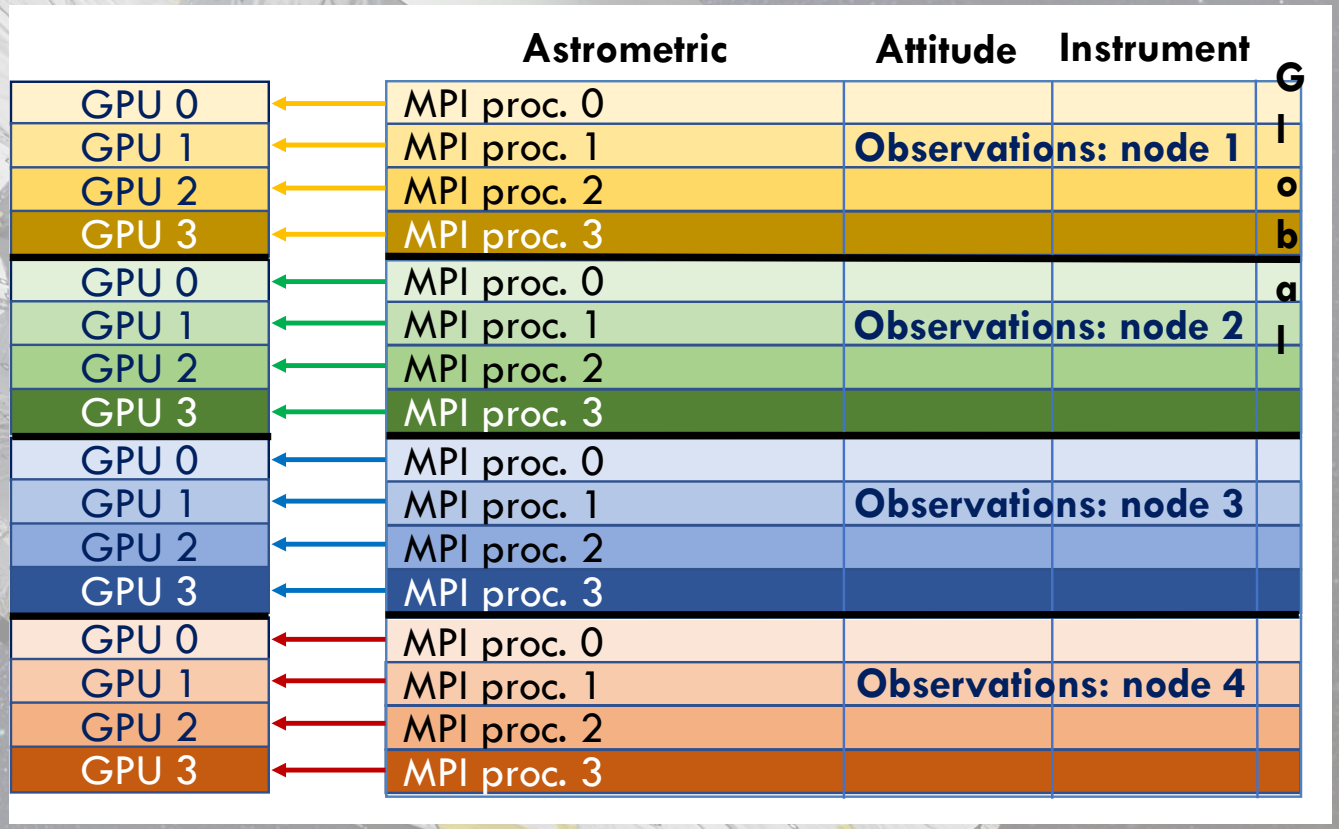
Cesare V., et al., 2021, ASP Conference Series, in Proc. of ADASS XXXI, in press

CUDA

OpenACC

3.1 Multi-GPU computation

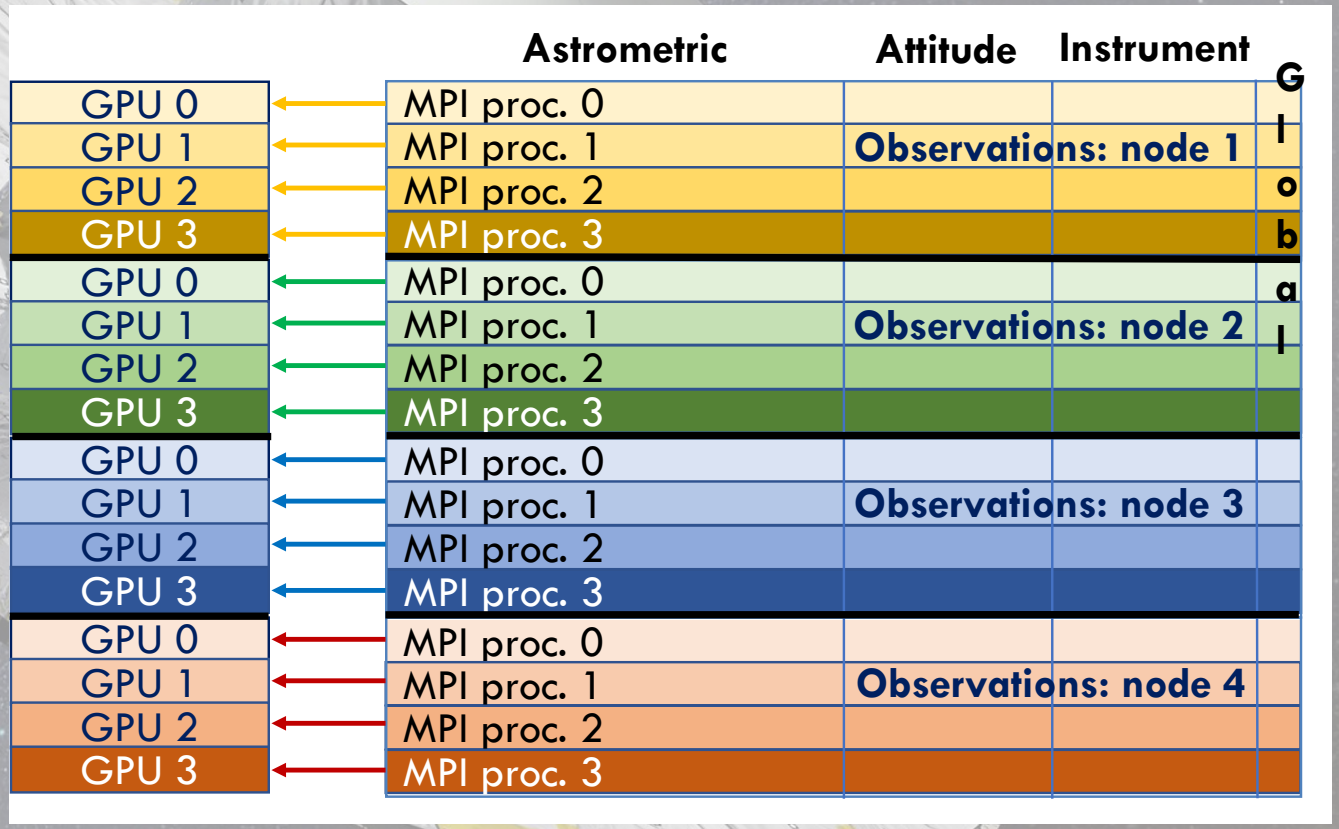
- MPI processes assigned to the GPUs of the node in a round-robin fashion.
- Optimal configuration: number of MPI processes per node = number of GPUs of the node.



Coefficient matrix of the GPU-ported codes parallelized on 4 nodes of a cluster.

3.1 Multi-GPU computation

- MPI processes assigned to the GPUs of the node in a round-robin fashion.
- Optimal configuration: number of MPI processes per node = number of GPUs of the node.
- Tests on CINECA supercomputer **Marconi100**, with **4 NVIDIA Volta V100 GPUs per node with 16 GB of memory each.**



Coefficient matrix of the GPU-ported codes parallelized on 4 nodes of a cluster.

3.2.1 The algorithms – aprod 1

OpenMP

```
int main {
#pragma omp parallel
  #pragma omp for
  for i ← 0 to N[pid] do
    sum = 0.0
    // Astrometric part
    k = i x N_par
    for j ← 0 to N_Astro do
      sum +=
      A_d[k]x[j+offs[i]]
      k++
    ... [similar construct
     x3]
    b[i] += sum
}
```

N. obs. per MPI proc.

N. param. ≠ 0 per obs. = 24

N. Astrometric param. ≠ 0 per obs. ≤ 5

OpenACC

```
int main {
// Astrometric part
#pragma acc parallel
  #pragma acc loop
  for i ← 0 to N[pid] do
    sum = 0.0
    k = i x N_par
    for j ← 0 to N_Astro do
      sum +=
      A_d[k+j]x[j+offs[i]]
    b[i] += sum
  ... [similar construct x3]
}
```

CUDA

```
// Astrometric part
__global__ void
a1_Kernel_Astro
(…, A_d,dev, b_dev, x_dev, …)
i=blockIdx.x*blockDim.x+threadIdx.x
  if (i < N[pid])
    sum = 0.0
    k = i x N_par
    for j ← 0 to N_Astro do
      sum +=
       A_d,dev[k+j]x_dev[j+offs[i]]
    b_dev[i] += sum
... [similar kernels definitions x3]

int main {
a1_Kernel_Astro<<<gridDim,
        blockDim>>>(…,
        A_d,dev, b_dev, x_dev, …)
... [similar kernel calls x 3]
cudaDeviceSyncronyze()
}
```

Kernel

3.2.2 The algorithms – aprod 2

OpenMP

ID of the OpenMP thread

```
int main {
#pragma omp parallel
  tid = omp_get_thread_num()
  for i← Nt[tid][0] to
  Nt[tid][1] do
    // Astrometric part
    k = i x Npar
    for j ← 0 to NAstro do
     x[j+offs[i]]+=Ad[k]b[i]
     k++
    ... [similar construct
     x3]
}
```

OpenACC

```
int main {
#pragma acc parallel
  #pragma acc loop
  for i ← 0 to N[pid] do
    // Astrometric part
    k = i x Npar
    for j ← 0 to NAstro do
     #pragma acc atomic
     x[j+offs[i]]+=Ad[k+j]b[i]
    ... [similar construct
     x3]
}
```

CUDA

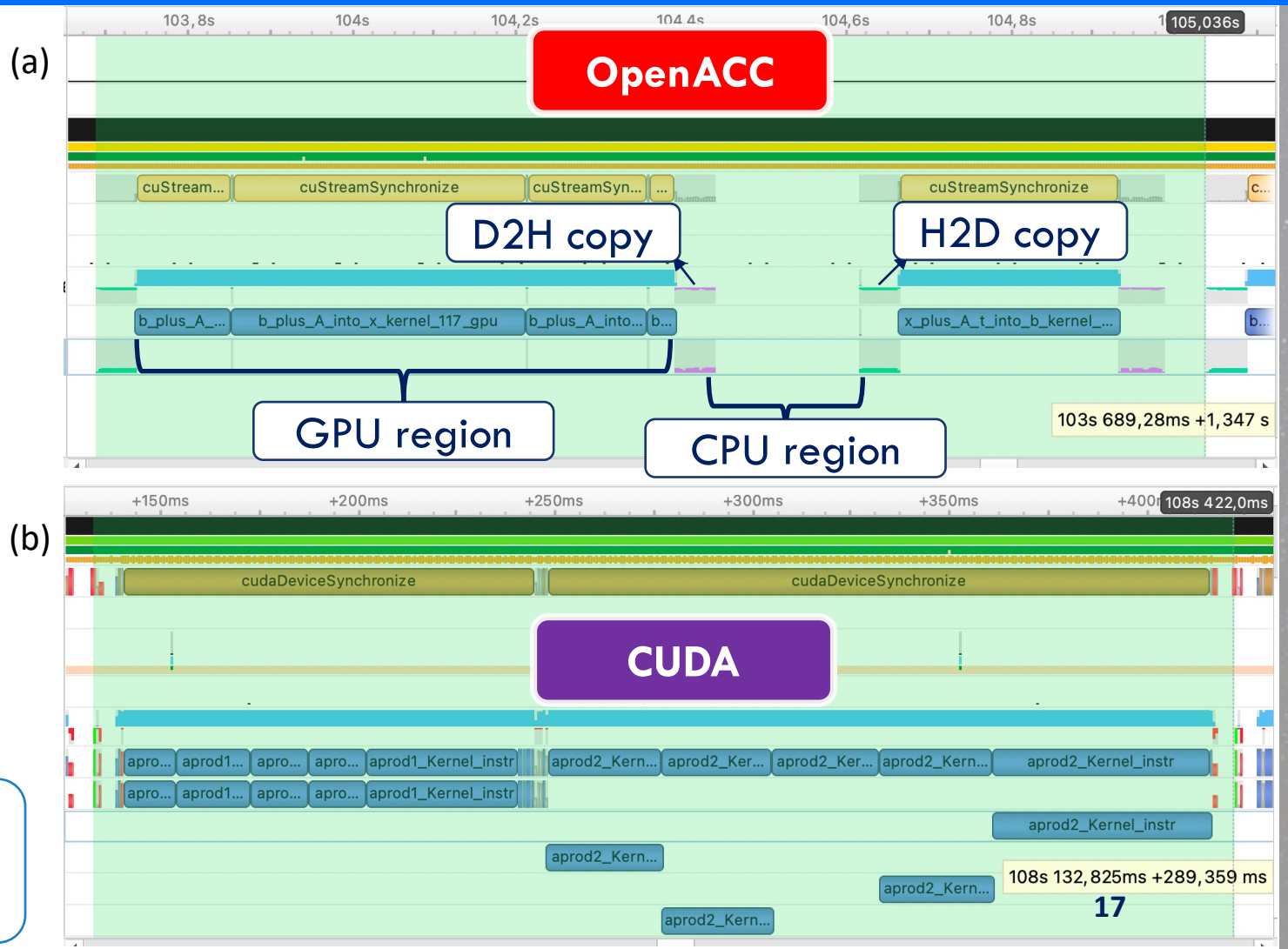
```
__global__ void
a2_Kernel_Astro
(…, Ad,dev,bdev,xdev,…)
i=blockIdx.x*blockDim.x+threadIdx.x
if (i < N[pid])
  k = i x Npar
  for j ← 0 to NAstro do
    atomicAdd(&x[j+offs[i]],
    Ad,dev[k+j]b[i])
... [similar kernels x3]

int main {
a2_Kernel_Astro<<<gridDim,
        blockDim,0,stream1>>>(…,
        Ad,dev,bdev,xdev,…)
... [similar kernel calls x3]
cudaDeviceSyncronyze()
}
```

3.3.1 Performance results – Part 1

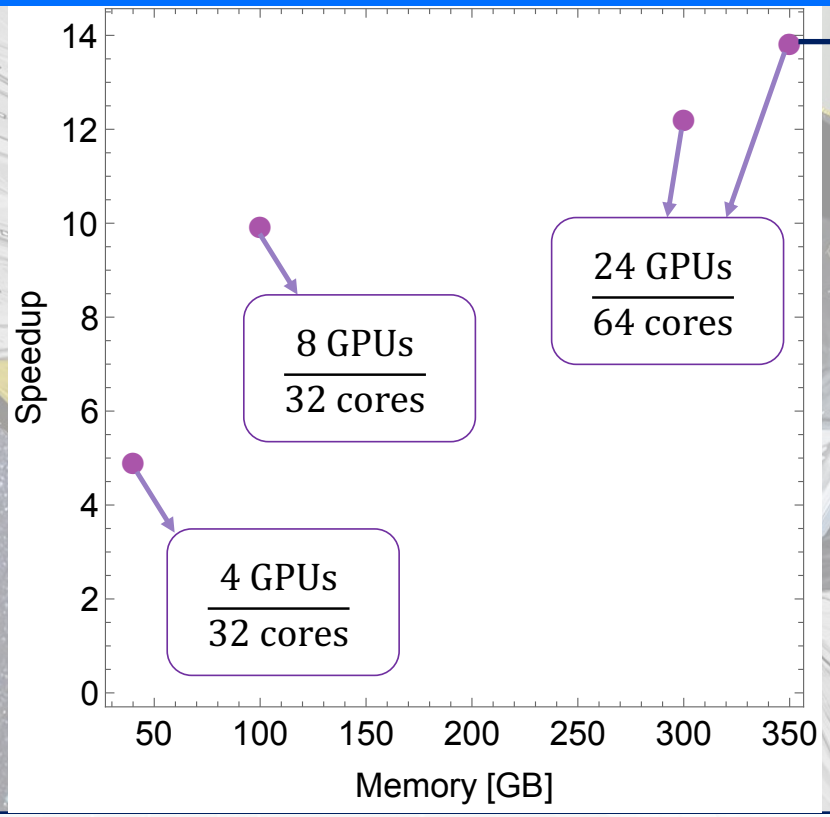
- Calculation time dominated by **GPU computation** and not by **data copies and CPU computation**.
- The iteration time fraction due to GPU computation passed **from ~70% (OpenACC) to >90% (CUDA)**
- The iteration time fraction due to data copies + CPU computation passed **from ~30% (OpenACC) to ~6% (CUDA).**

Compute bound code



NVIDIA Nsight Systems (https://developer.nvidia.com/nsight-systems) profiler output for a 50 GB run parallelized on 4 MPI processes of 1 node of Marconi100.

3.3.2 Performance results – Part 2



SPEEDUP: ~14x

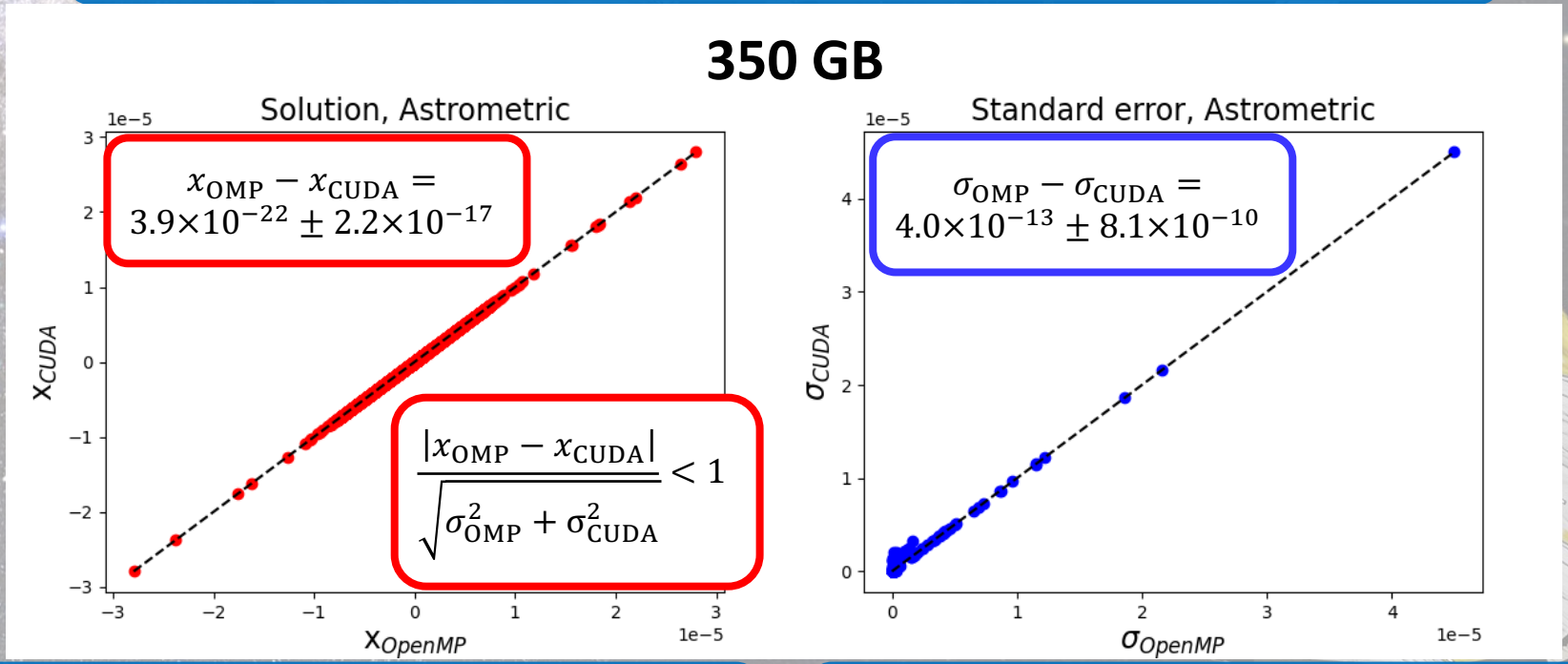
From **~4 s** per iteration (OpenMP) to **~0.3 s** per iteration (CUDA).

- Speedup CUDA-over-OpenMP increasing with a **more efficient utilization of the GPUs** and with the **system size.**
- **Speedup of ~14x** for the largest system! And it is supposed to further increase.

3.4 Numerical stability

Comparison between the solutions and their uncertainties found by the OpenMP and the CUDA codes for a set of different systems.

Example for a system that occupies 350 GB of memory:



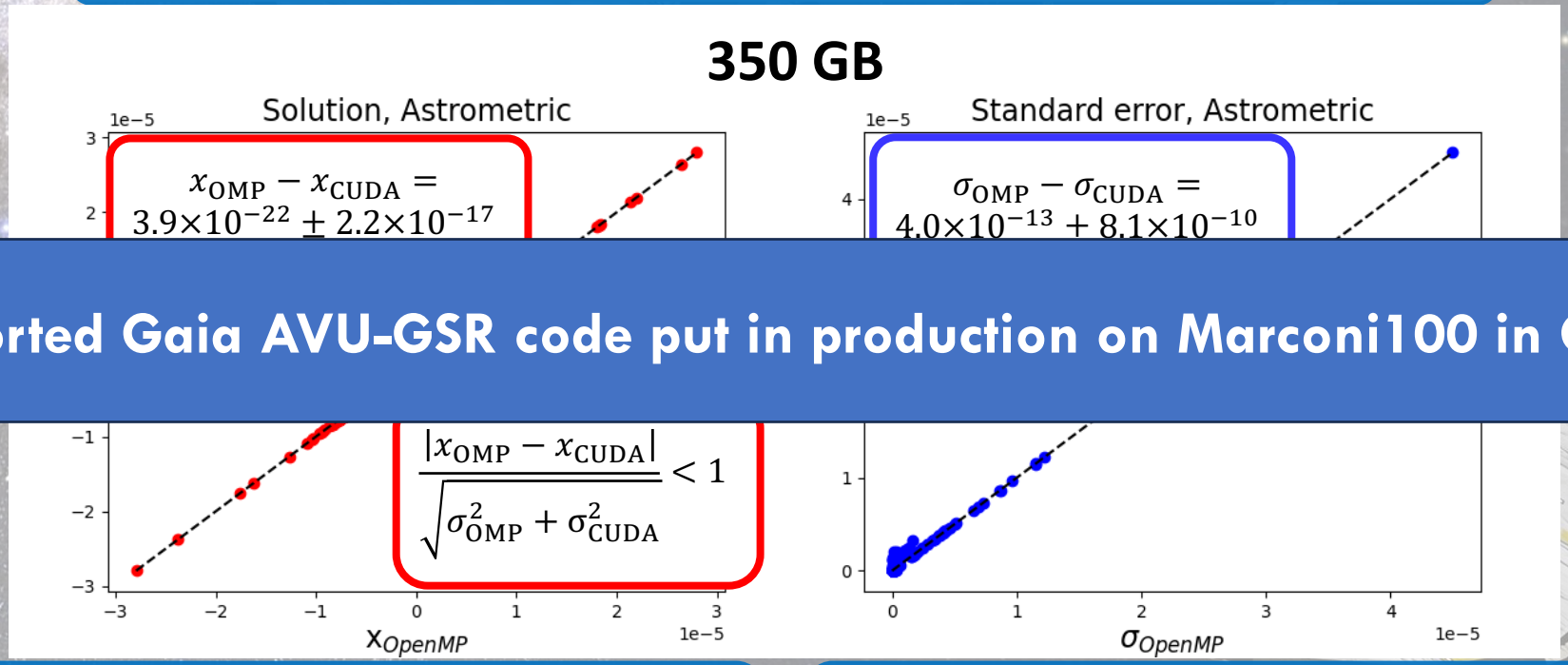
Solutions consistent within 1σ

Differences of the uncertainties consistent with zero

3.5 Numerical stability

Comparison between the solutions and their uncertainties found by the OpenMP and the CUDA codes for a set of different systems.

Example for a system that occupies 350 GB of memory:

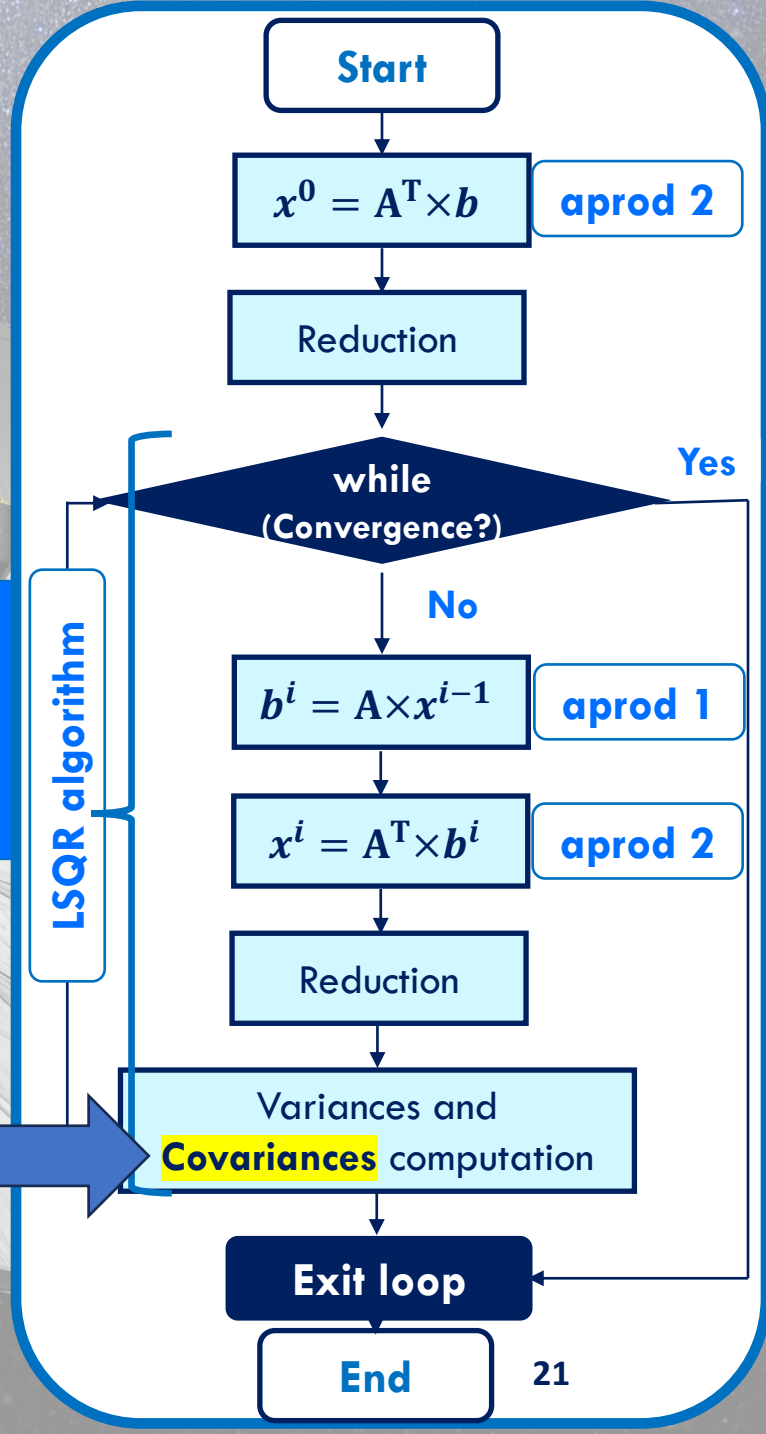


CUDA-ported Gaia AVU-GSR code put in production on Marconi100 in Q2 2022.

Solutions consistent within 1σ

Differences of the uncertainties consistent with zero

4. The covariances computation



4.1 Computational problem

- Covariances calculation in the Gaia AVU-GSR code cannot be faced with standard approaches.
- The total number of covariances is $\sim N_{\mathrm{unk}}^2/2$. With $N_{\mathrm{unk}} \sim 5 * 10^8$ they would occupy ~1EB of memory: **unresolvable problem on existing infrastructures from the memory and storage points of view** → Only compute a subset of total covariances.
- Generate couples of covariances indexes, $\mathtt{index}_1$ and $\mathtt{index}_2$, where each index goes from 0 to $N_{\mathrm{unk}} - 1$.

4.1 Computational problem

- Covariances calculation in the Gaia AVU-GSR code cannot be faced with standard approaches.
- The total number of covariances is $\sim N_{\mathrm{unk}}^2/2$. With $N_{\mathrm{unk}} \sim 5 * 10^8$ they would occupy ~1EB of memory: **unresolvable problem on existing infrastructures from the memory and storage points of view** → Only compute a subset of total covariances.
- Generate couples of covariances indexes, index_1 and index_2, where each index goes from 0 to $N_{\mathrm{unk}} - 1$.

Covariances calculation:

```
for i ← 0 to N_Cov do
  dkprod = factor*x_Glob[index_1]*x_Glob[index_2]
  coVariance[i] += dkprod
```

4.1 Computational problem

- Covariances calculation in the Gaia AVU-GSR code cannot be faced with standard approaches.
- The total number of covariances is $\sim N_{\mathrm{unk}}^2/2$. With $N_{\mathrm{unk}} \sim 5 * 10^8$ they would occupy ~1EB of memory: **unresolvable problem on existing infrastructures from the memory and storage points of view** → Only compute a subset of total covariances.
- Generate couples of covariances indexes, index_1 and index_2, where each index goes from 0 to $N_{\mathrm{unk}} - 1$.

Covariances calculation:

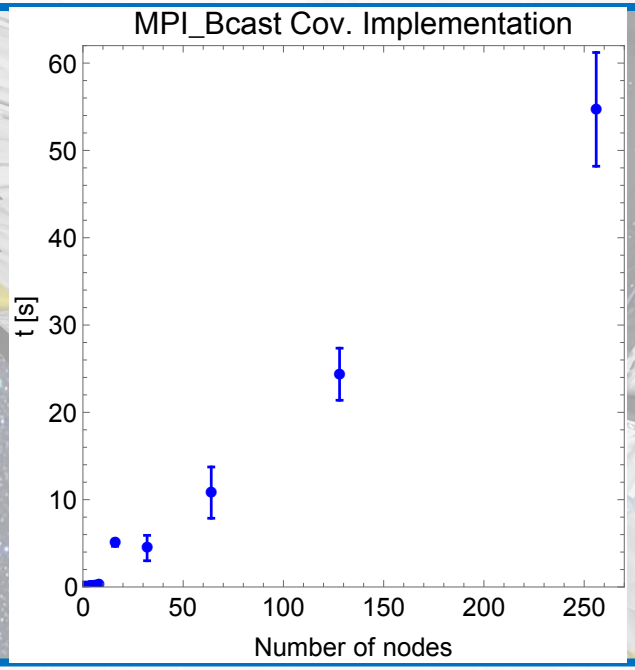
```
for i ← 0 to N_Cov do
  dkprod = factor*x_Glob[index_1]*x_Glob[index_2]
  coVariance[i] += dkprod
```

Problem:

index_1 and index_2 are global indexes, whereas x is a local array → All the MPI processes would have to know the entire array of the unknowns to evaluate $x_{\mathrm{Glob}}[\mathrm{index}_1]$ and $x_{\mathrm{Glob}}[\mathrm{index}_2]$.

4.2 The MPI_Bcast strategy

Each MPI process broadcasts x (~2*10^6 elements ~ 16 MB) to all the other MPI processes at every iteration:



The code already passes from **"compute-bound"** to **"communication-bound"** from 16 nodes onwards. ⇒ Severe loss of performance when the number of nodes, and, thus, of MPI communications, increases (the advantage due to the CUDA porting is completely lost).

4.3 The I/O strategy

Gaia AVU-GSR code, Program 1

Simultaneous start

Covariances code, Program 2

PE = 0 | PE = 1 | ... | PE = nproc - 1

x_{Loc}

itn 0

itn 1

itn 2

...

itn itnCycle

Print

Output directory

~$2*10^6$ elements ~ 16 MB

`itnCycle` = 1000 provides an optimal trade-off between **writing performance** and **storage occupancy** (**~ 7 TB** per cycle at the end of the Gaia mission).

while (All the cycles are read, computed and deleted?)

Yes

No

while (Files present?)

Yes

No

sleep(30)
Check if the files are present in the output directory

`for i ← 0 to itnCycle do`

Reading phase

itn 0

Covariances computation

Delete the files in the output directory for the correspondent cycle

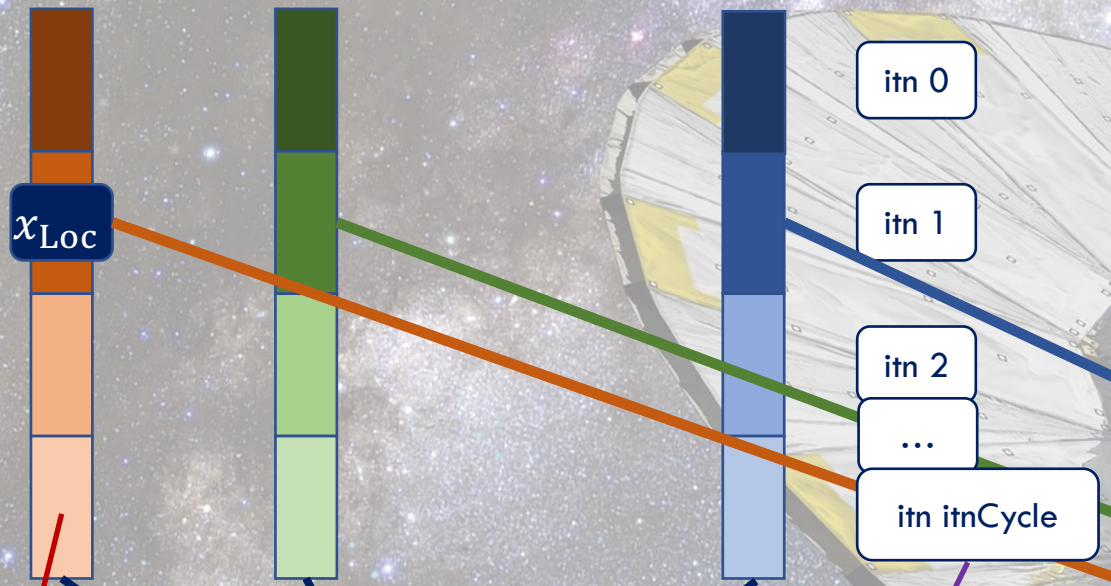
Exit loop

Gaia AVU-GSR code, Program 1

Simultaneous start

Covariances code, Program 2

PE = 0 | PE = 1 | … | PE = nproc - 1

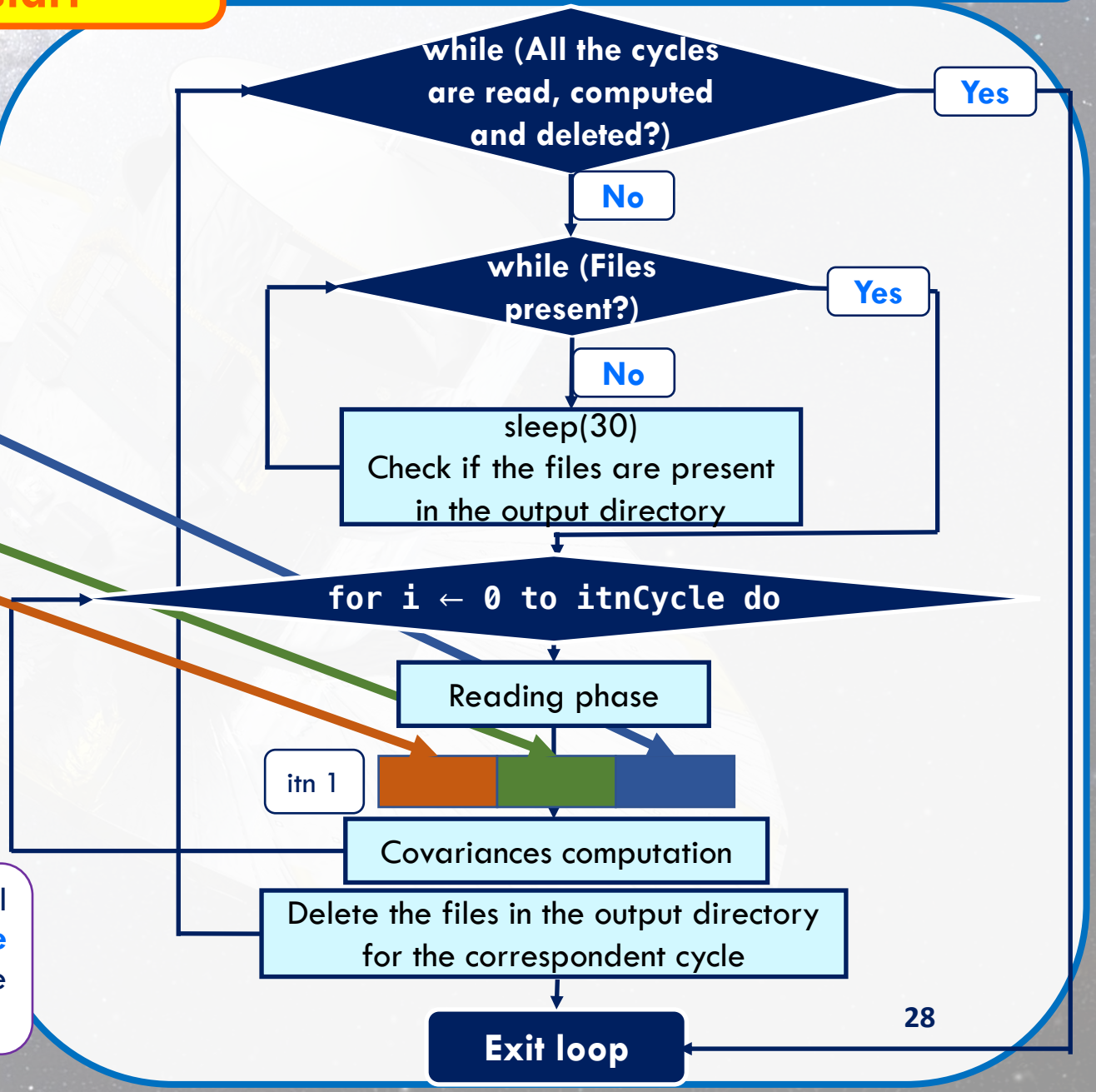


Print

Output directory

~2*10⁶ elements ~ 16 MB

`itnCycle` = 1000 provides an optimal trade-off between **writing performance** and **storage occupancy** (**~ 7 TB** per cycle at the end of the Gaia mission).

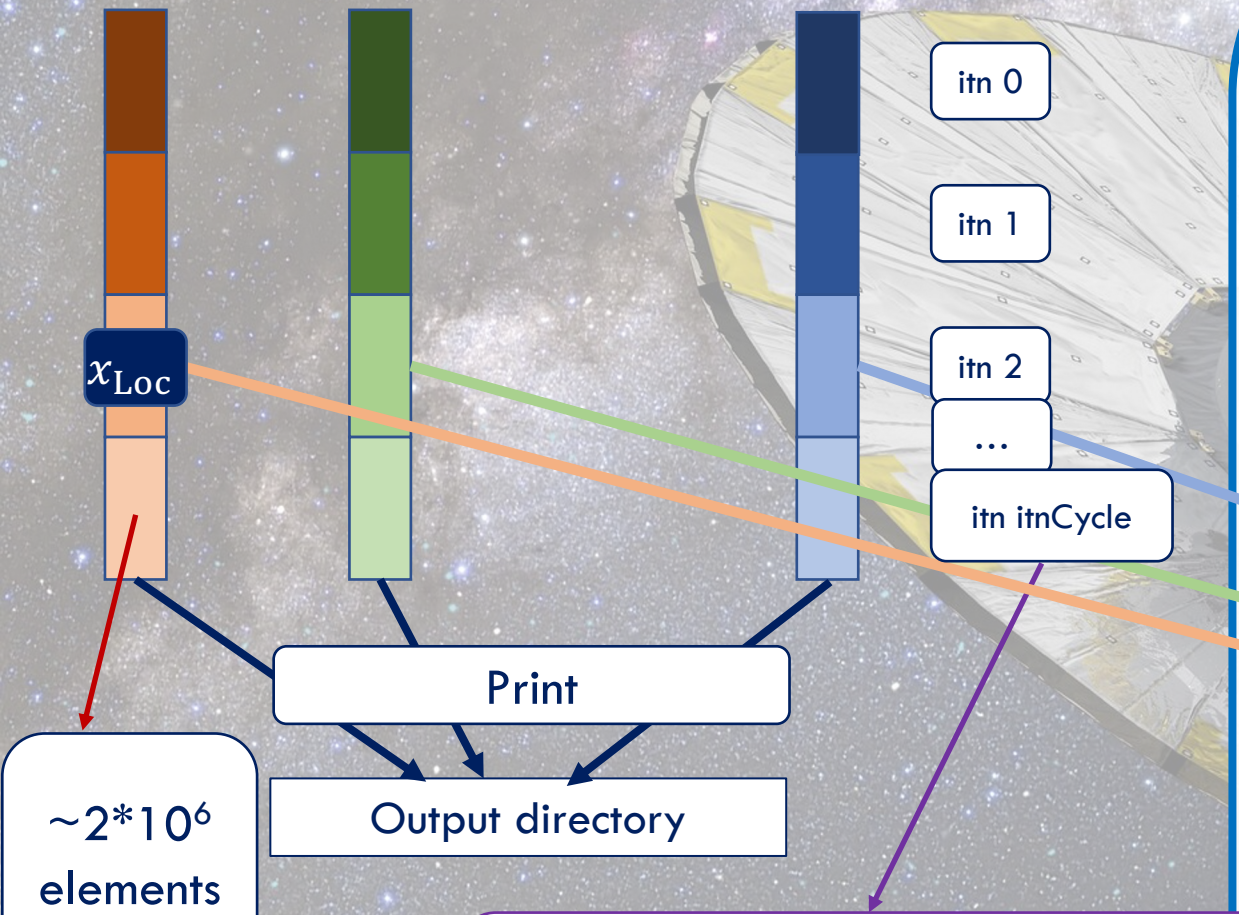


Gaia AVU-GSR code, Program 1

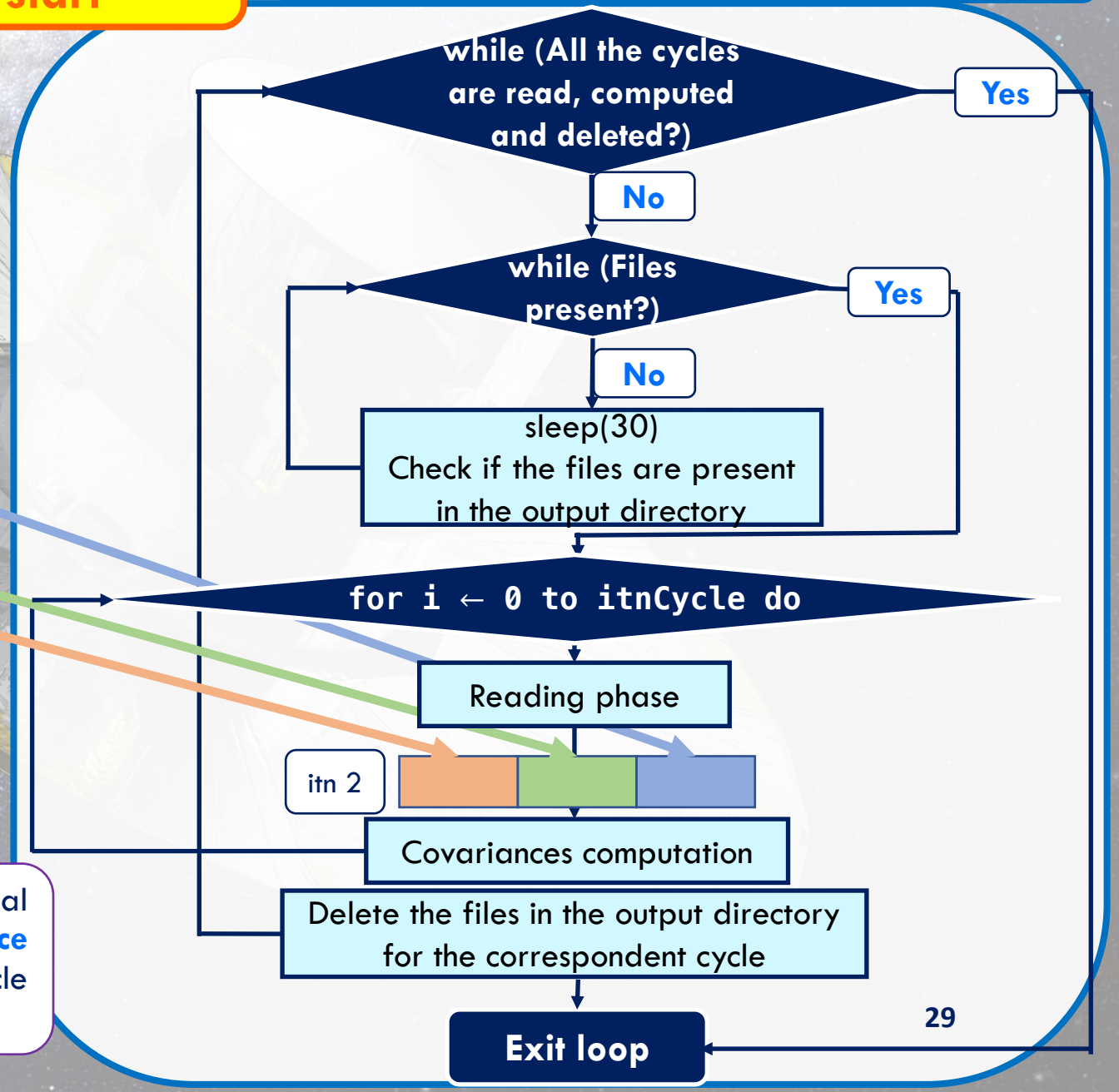
Simultaneous start

Covariances code, Program 2

PE = 0 | PE = 1 | … | PE = nproc - 1



`itnCycle` = 1000 provides an optimal trade-off between **writing performance** and **storage occupancy** (**~ 7 TB** per cycle at the end of the Gaia mission).



Gaia AVU-GSR code, Program 1

Simultaneous start

Covariances code, Program 2

PE = 0 | PE = 1 | ... | PE = nproc - 1

itn 0

itn 1

itn 2

...

itn itnCycle

x_{Loc}

Print

Output directory

~$2*10^6$ elements ~ 16 MB

`itnCycle` = 1000 provides an optimal trade-off between **writing performance** and **storage occupancy** (**~ 7 TB** per cycle at the end of the Gaia mission).

while (All the cycles are read, computed and deleted?)

Yes

No

while (Files present?)

Yes

No

sleep(30)
Check if the files are present in the output directory

`for i ← 0 to itnCycle do`

Reading phase

itn itnCycle

Covariances computation

Delete the files in the output directory for the correspondent cycle

Exit loop

4.3 Preliminary performance results

- To exploit the maximum advantage from this method the time

$$\text{itnCycle} * (t_{\text{Read}} + t_{\text{Cov}})$$

in the Covariances code has to be smaller than

$$\text{itnCycle} * t_{\text{Iter}} + t_{\text{Write}}$$

in the Gaia AVU-GSR code.

4.3 Preliminary performance results

- To exploit the maximum advantage from this method the time

$$\text{itnCycle} * (t_{\text{Read}} + t_{\text{Cov}})$$

in the Covariances code has to be smaller than

$$\text{itnCycle} * t_{\text{Iter}} + t_{\text{Write}}$$

in the Gaia AVU-GSR code. ➡ This allows to delete the files related to a certain cycle before the next cycle of files is generated and so to avoid storage problems.

4.3 Preliminary performance results

❖ To exploit the maximum advantage from this method the time

$$\text{itnCycle} * (t_{\text{Read}} + t_{\text{Cov}})$$

in the Covariances code has to be smaller than

$$\text{itnCycle} * t_{\text{Iter}} + t_{\text{Write}}$$

in the Gaia AVU-GSR code. ➡ This allows to delete the files related to a certain cycle before the next cycle of files is generated and so to avoid storage problems.

❖Fine-tuning of different solutions are being explored to achieve this target.

5. Conclusions and outlooks

- Porting of the Gaia AVU-GSR pipeline on CINECA platform Leonardo for an optimal production in perspective of future Gaia data releases. We expect an even better performance since Leonardo has:
  - 4x GPU memory per node (4 A200 GPUs with 64 GB each per node on Leonardo vs 4 V100 GPUs with 16 GB each per node on Marconi100);
  - Leonardo GPUs with more streaming multiprocessors compared to Marconi100 GPUs. ⇒ More concurrent threads.

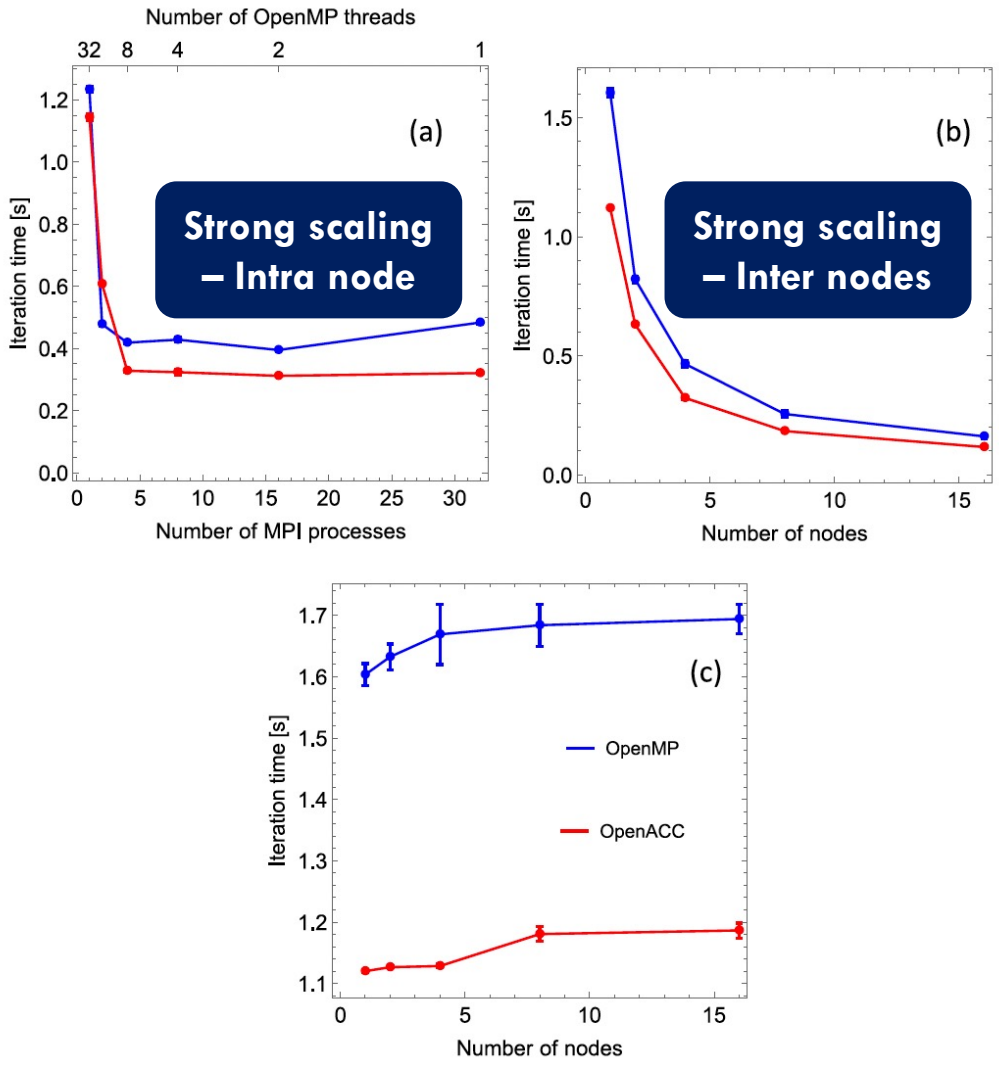
5. Conclusions and outlooks

- Porting of the Gaia AVU-GSR pipeline on CINECA platform Leonardo for an optimal production in perspective of future Gaia data releases. We expect an even better performance since Leonardo has:
  - 4x GPU memory per node (4 A200 GPUs with 64 GB each per node on Leonardo vs 4 V100 GPUs with 16 GB each per node on Marconi100);
  - Leonardo GPUs with more streaming multiprocessors compared to Marconi100 GPUs. ⇒ More concurrent threads.
- Targets of 2 years INAF Mini Grant (in collaboration with Prof. Marco Aldinucci of UniTO):
  - Strong and weak scaling, numerical stability, and Green Computing studies up to system sizes of the final Gaia dataset (10-100 TB) (also targets of CN–Spoke 1–FL5);
  - Tests repeated to compare the CUDA code and a code version rewritten in C++, which allows GPU offloading and a greater code portability.

Thank you for the attention!!! ☺

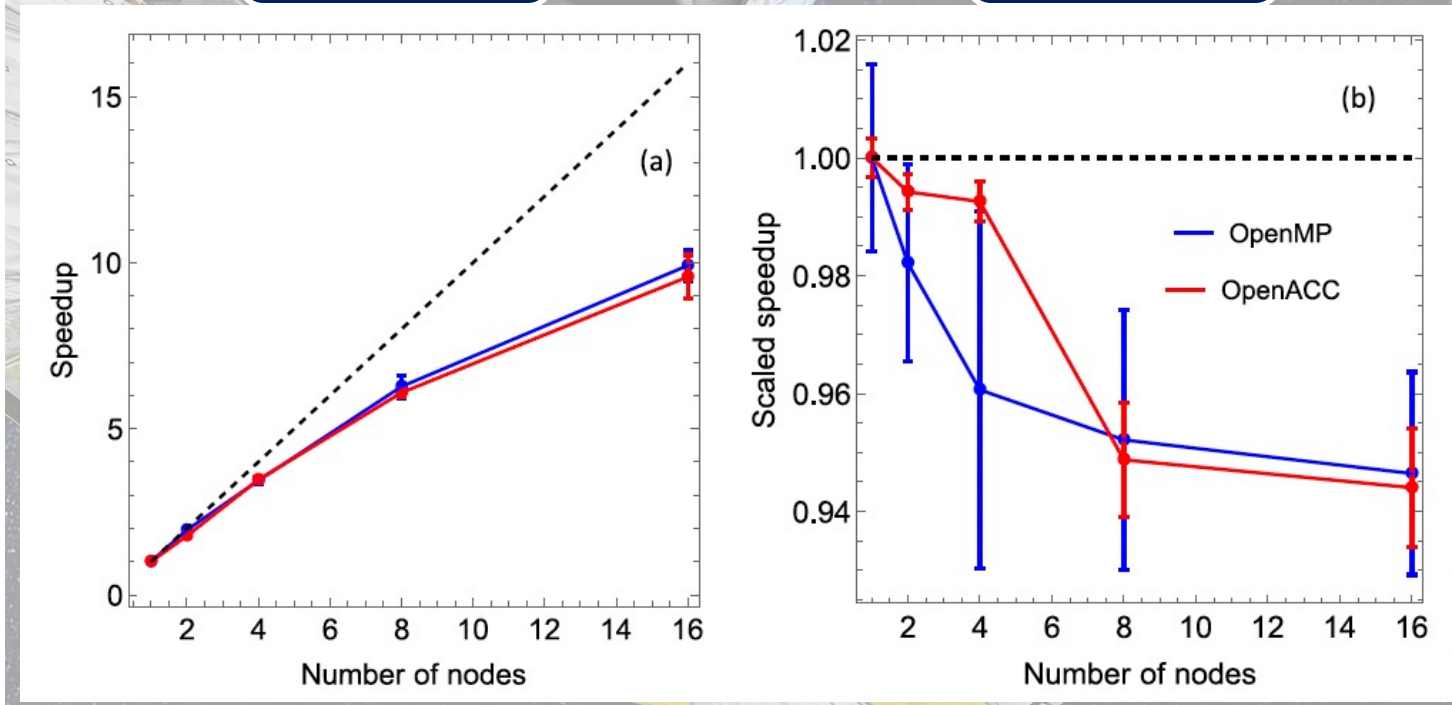
EXTRA SLIDES

A. Strong and weak scaling curves for the OpenMP and the OpenACC AVU-GSR codes



Strong scaling – Inter nodes

Weak scaling – Inter nodes



Cesare et al. (2022c)