Recent developments on NetCDF I/O: towards hiding I/O computations on COSMO

Carlos Osuna (C2SM), Oliver Fuhrer (MeteoSwiss), Neil Stringfellow (CSCS)

HP2C

thanks to Daniel Lüthi (IAC)

Motivation

The 0.5km resolution COSMO is one of the IAC runs in production at CSCS
Taking 1 hour simulation, on ~1850 processors, as an example:

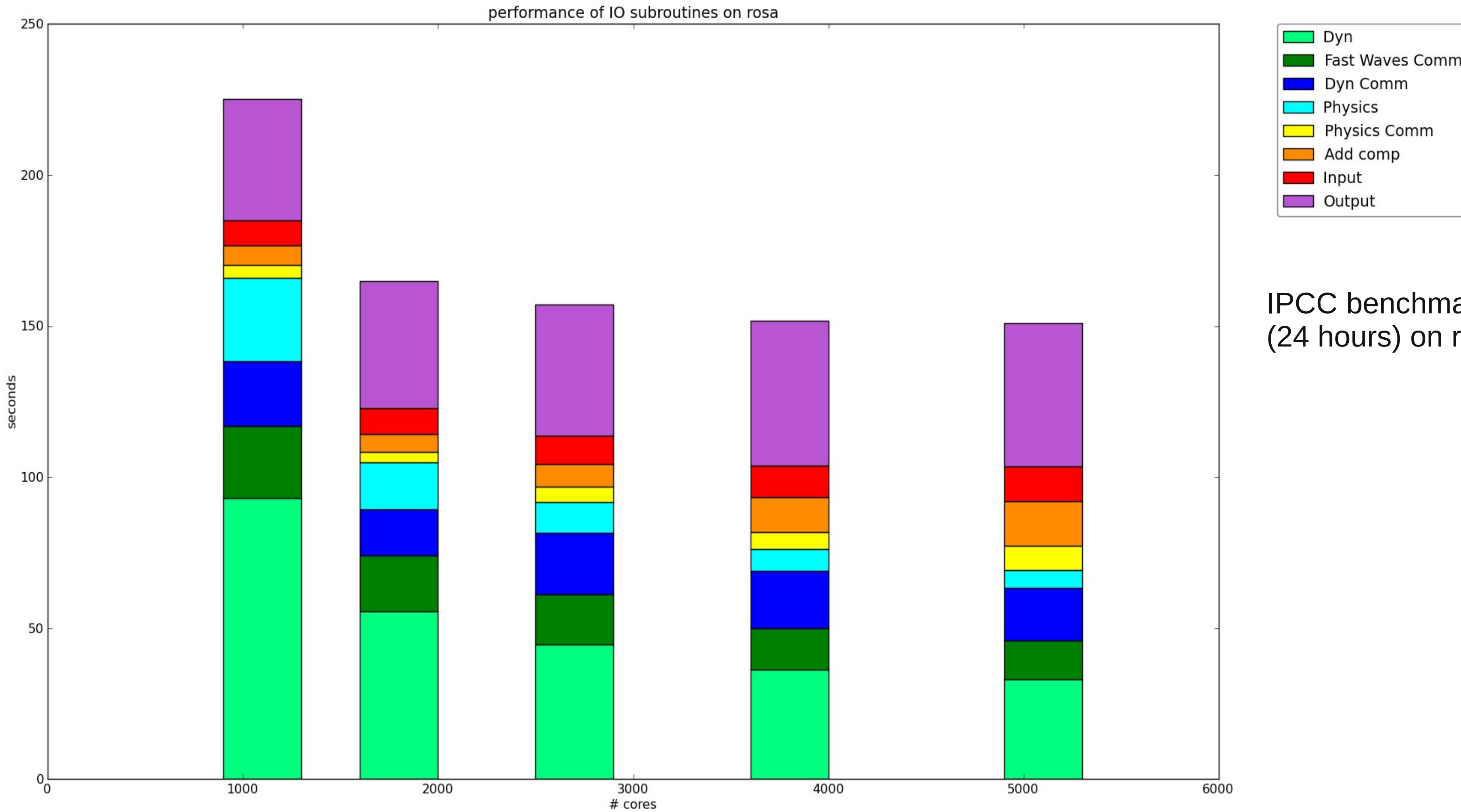
Section	Time (seconds)
Dynamics	897
Physics	189
Output	543
Others	273
Total	1902

We can see that writing output data is ~29% of the total run time.

Could we reduce the output time by increasing the number of processors?

Motivation

Well, in contrast with other sections of the model, the *output* does not scale with the number of processors (its time does not decrease with increasing number of processors).



IPCC benchmark (24 hours) on rosa

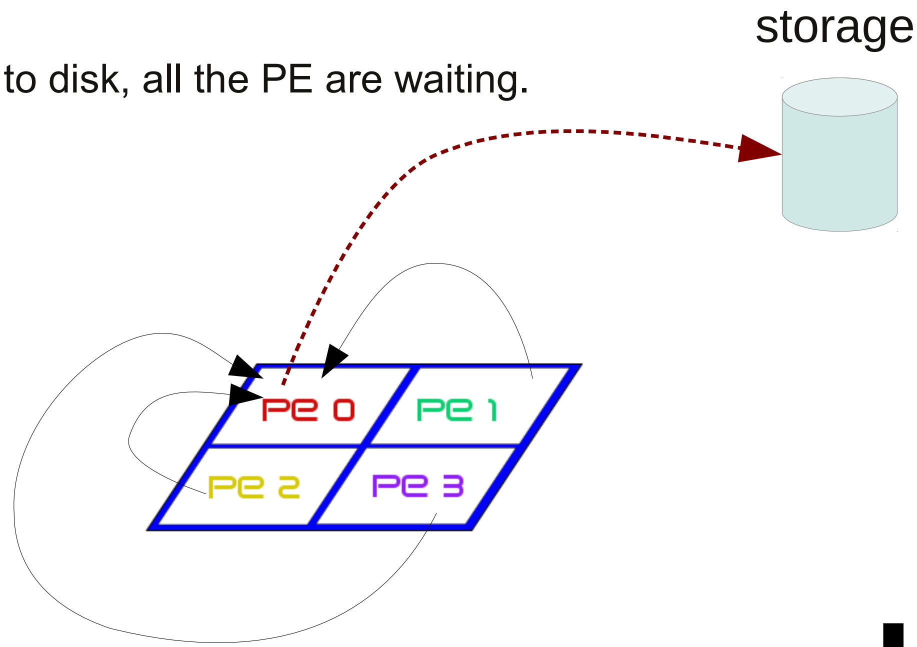
Current I/O strategy

The sequential design of the code that writes output data is the reason the timing of this section does not scale.

Every processor element (PE) with a full record of data will send it to PE 0, which writes data to disk through netcdf.

The process is designed sequentially:

Until PE 0 has delivered all the data requested to disk, all the PE are waiting.

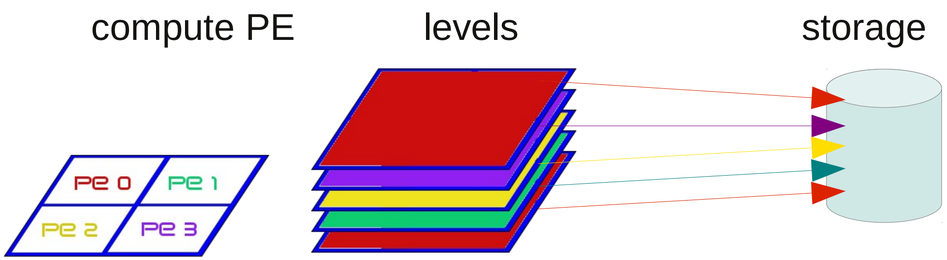


A typical I/O design will specialize processors: compute PE will actually perform only computation, I/O PE will deal with reading/writing data to disk.
Both tasks can overlap in time.

Goal: Improve the I/O performance for the netcdf format.
Ideally, hide the I/O from the compute PE.

Strategies studied:

- **Level to process mapping**
  (we ask every compute PE with a full record to write directly into disk using parallel Netcdf)



Using a parallel API of netcdf, with netCDF-4/HDF5 underlying layer.

But... parallel netcdf has some limitations in performance

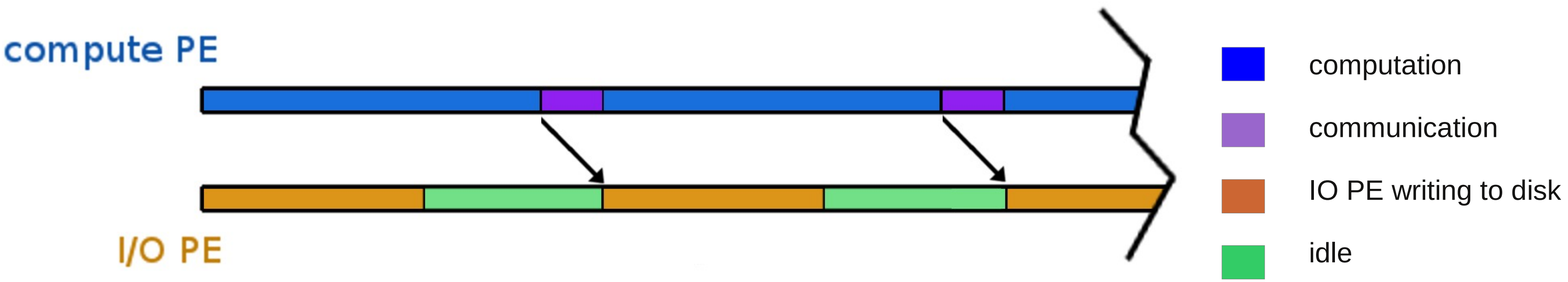
After quite some tuning, only ~300 MB/s bandwidth (non scalable with # processors) was achieved.

The bandwidth obtained makes this strategy not very satisfactory.

Goal: Improve the I/O performance for the netcdf format.
Ideally, hide the I/O from the compute PE for COSMO netcdf.

Strategies studied:

- **Asynchronous I/O**
some PE are reserved for I/O specific tasks. Compute PE can send data to asyn I/O PE and continue processing next steps.



Use case for measurements & development:

IPCC benchmark, 15 km resolution (441x429x40) , 24 hours,
total output size ~2.8 GB

Results for IPCC

Section	Asyn I/O (seconds)	Orig COSMO (seconds)
Dyn Comp	198.4	197.7
Fast waves Comm	26.4	27.09
Dyn Communication	31.89	33.68
Phys. Comp.	74.9	75.4
Phys. Comm.	10.36	10.38
Add. Comp.	10.48	10.5
Input	4.93	5.25
Output	6.68	132.1
Computations_O		93.99 [17.7 – 128.7]
Write data		34.27 [0 - 111.1]
Gather data		3.85
TOTAL	369.6	498.59

~26% reduction in total time using new asynchronous strategy on a 400 processors run.

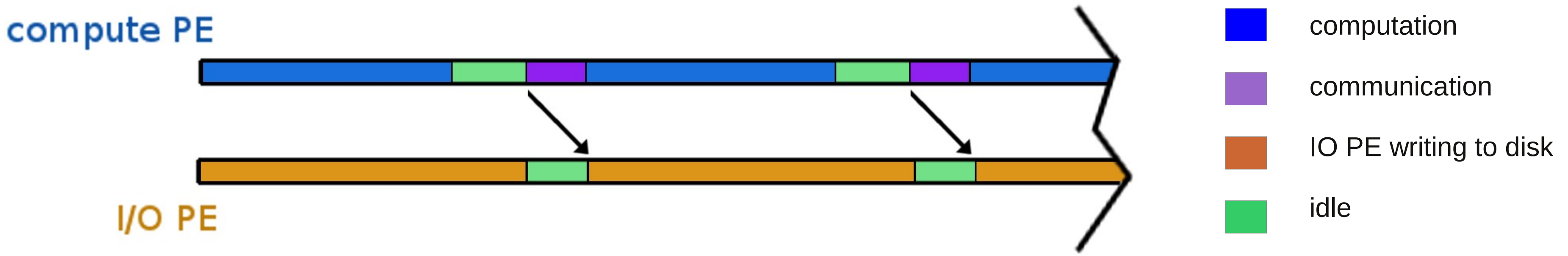
Looks good! but we are not there yet...

there might be bottlenecks if:

- I/O PE takes more time to write a file than compute PE finishing computation, i.e. (computation time) < (data size) / bandwidth

or

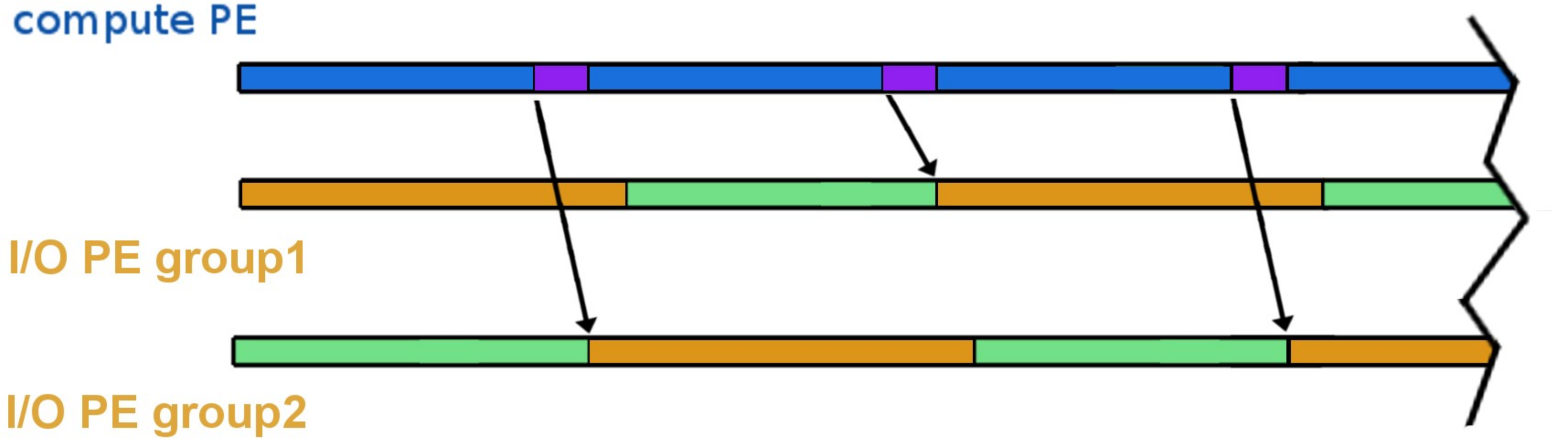
- multiple files are being written for one timestep



Add more resources to I/O??

While I/O PE assigned to I/O are busy writing a NetCDF file, they are blocked, and you cannot process data from next file even if you add more I/O PEs.

We can parallelize writing over several netcdf files by assigning different I/O PEs to different files. By doing this we break the bandwidth limit provided by one file of 300 MB/s



Ideally, adding I/O PEs in this way will scale until we reach the filesystem limit.

Results (using several I/O groups)

Testing the new solution on a more demanding IPCC based example:

- 24 simulated hours,
- 400 processors
- 130 GB total data size,
- 148 files,
- 168771 vertical slices.

Section	Asyn I/O with 3 IO PE (s) per comm & 6 comm	Asyn I/O with 3 IO PE (s) per comm & 1 comm	Orig COSMO(s)
Output	110	882	1103.71
Computations O	6.3		112.83 [49 - 141]
Write data	5.4 [4.3-8.1]		404 [1 – 907]
Gather data	99 [96-100]		584 [182 - 952]
Final Wait	19 [16.7-22]		
Total	440.98	1207	1502

Dedicating 18 processors in 6 I/O groups, total time was reduced by 70%.

But still time of *output* section is not 0

Due to the process of gathering of a full record from the set of subdomains computed in all compute PE.

Section	Asyn I/O with 3 IO PE (s) per comm & 6 comm	Asyn I/O with 3 IO PE (s) per comm & 1 comm	Orig COSMO(s)
Output	110	882	1103.71
Computations O	6.3		112.83 [49 - 141]
Write data	5.4 [4.3-8.1]		404 [1 – 907]
Gather data	99 [96-100]		584 [182 - 952]
Final Wait	19 [16.7-22]		
Total	440.98	1207	1502

Substitute MPI_GATHER by MPI_ALLTOALL communication for gathering, the total output time is reduced to negligible values.

Speedup

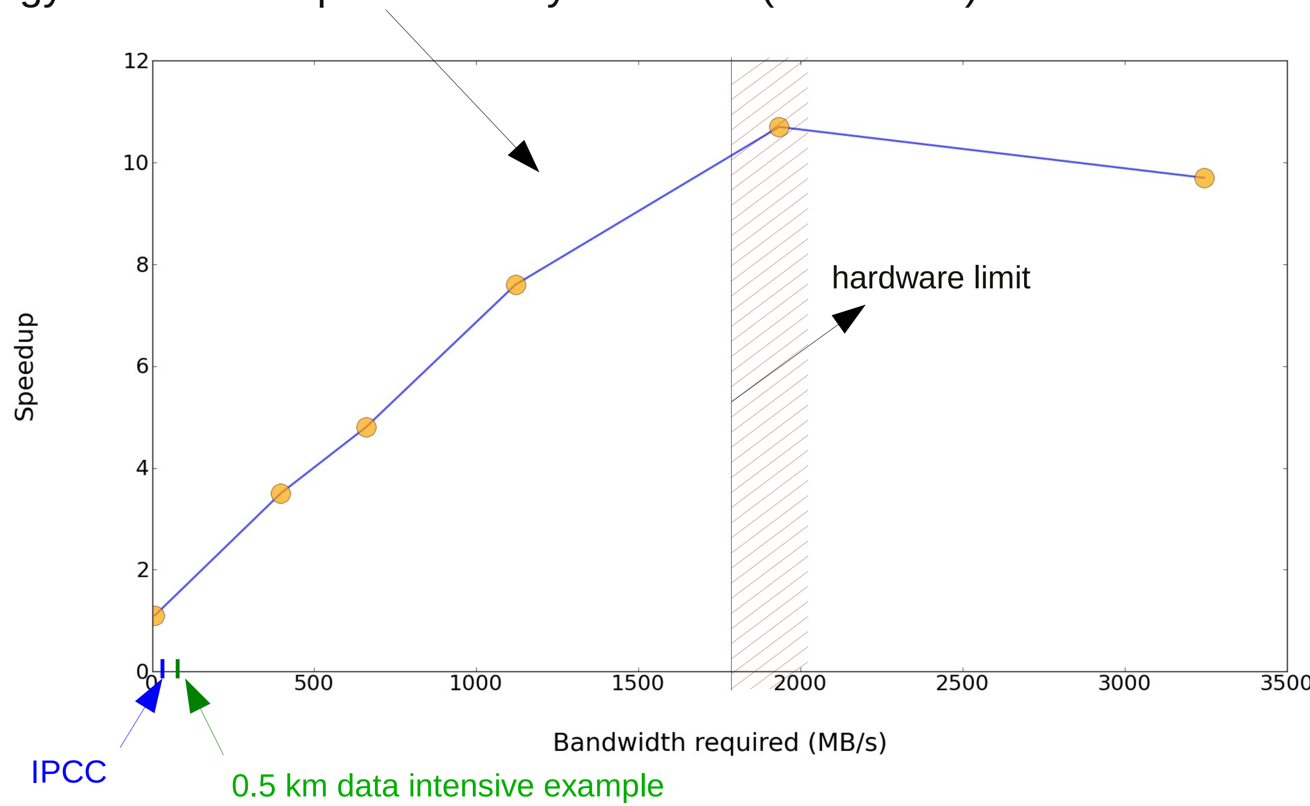
Speedup = (total time sequential I/O cosmo)/
(total time asyncronous strategy)

In general, the speedup will depend on the rate of output data being written.
We can use *bandwidth required* as a measurement of this rate.

bandwidth required = (data size) / (total run time - I/O time)

0.5 Km example: 1 hour simulation, 52 GB. of data, bandwidth required ~40 MB/s.
Total run time is reduced by 25%.

Study of massive data dumping examples (based on IPCC benchmark) shows that the strategy scales well up to the filesystem limit (~1.8 GB/s).



for 400 processors on rosa

Conclusions:

- A new asynchronous strategy for Netcdf I/O was designed in a flexible way to be able to deal with different I/O pattern use cases.
- The number of I/O groups is configurable, and it can be increased when large netcdf files writing operation overlap.
- Together with ALLTOALL gathering operation, the strategy completely hides the IO tasks from the total run time.
- Tested with many different I/O patterns, it scales well until we reach the filesystem limit (~1.6 GB/s).
- The speedup depends on how much time your run spends in I/O (asynchronous strategy removes I/O time):

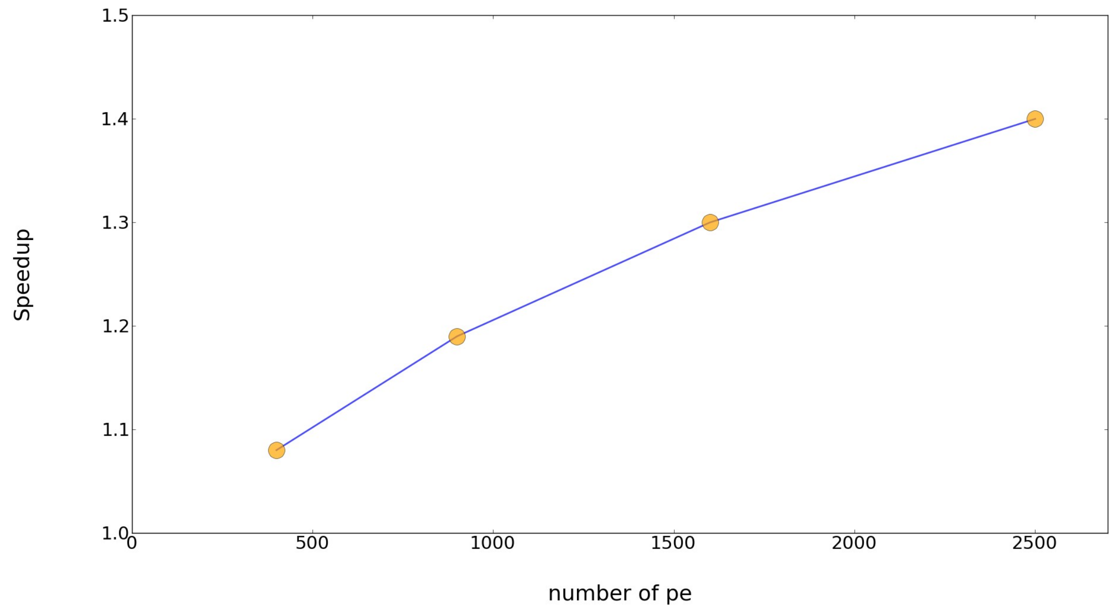
  speedup = (total run time) / (total run time - I/O time)
- Validation tests were carried out to guarantee same results.
- Looking forward to use it in production.

BACKUP SLIDES

Speedup

The *bandwidth required* will depend on the number of processors configured. (Increasing the number of processors will reduce computation time of each time step and increase bandwidth required).

Therefore, the speedup will also depend on the number of processors.



IPCC benchmark on rosa