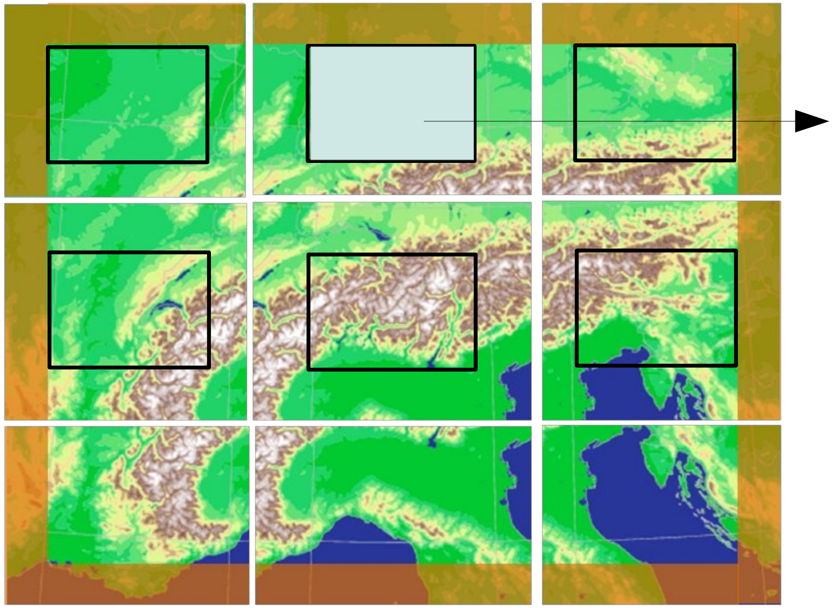
Internode communication (CPU & GPU)

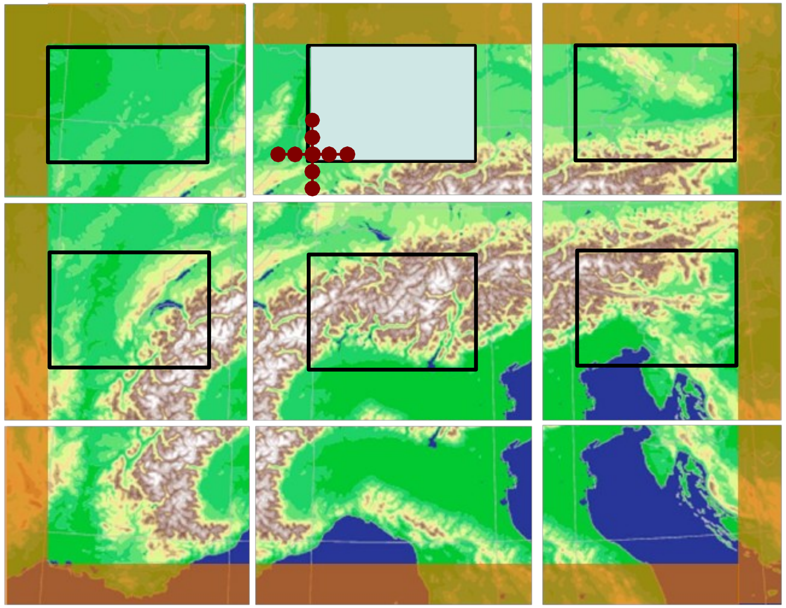
Carlos Osuna (C2SM), Mauro Bianco (CSCS), Ugo Varetto (CSCS),
Tobias Gysi (SCS), Oliver Fuhrer (MeteoSwiss), Peter Messmer (NVIDIA)

Introduction to Inter-Node Communication



1. Stencil A computes field1 at the inner domain

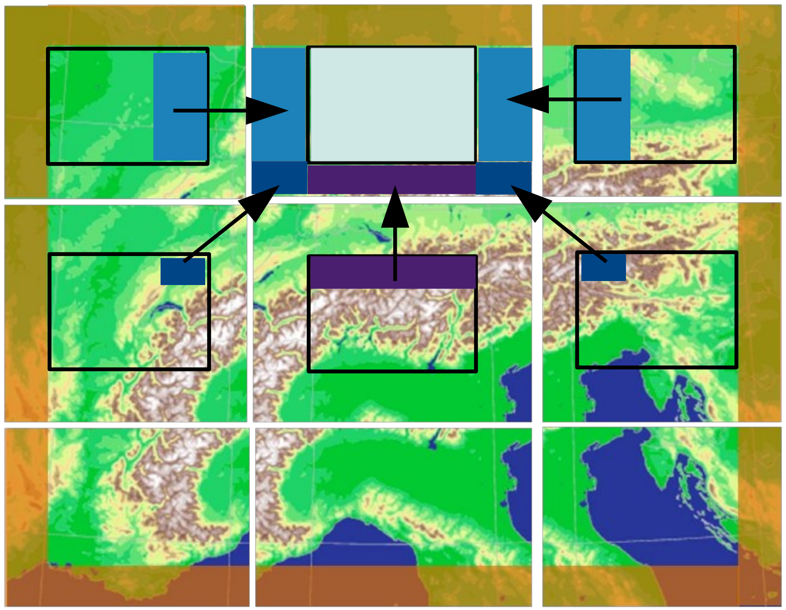
Introduction to Inter-Node Communication



1. Stencil A computes field1 at the inner domain

3. Stencil B requires updated values of field1 at the boundaries.

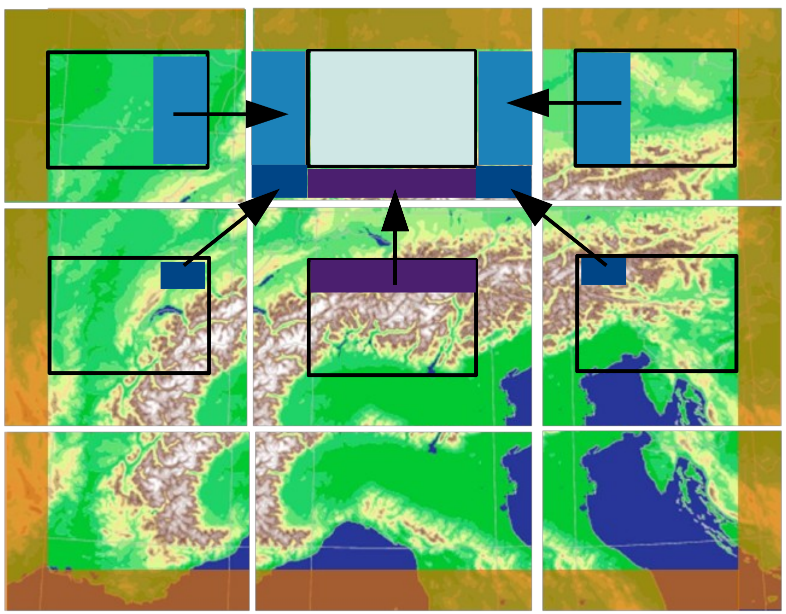
Introduction to Inter-Node Communication



1. Stencil A computes field1 at the inner domain
2. Communicate halos for field1 from neighbour PE's.
3. Stencil B requires updated values at the boundaries.

Introduction to Inter-Node Communication

In COSMO halo exchanges between neighbours are handled by exchg_boundaries subroutine



New features required:

HP2C cosmo project needs a library that can handle inter-GPU communication.

HP2C Cosmo Dycore is completely rewritten in C++, which requires a communication library (available from C++) to deal with halo exhances

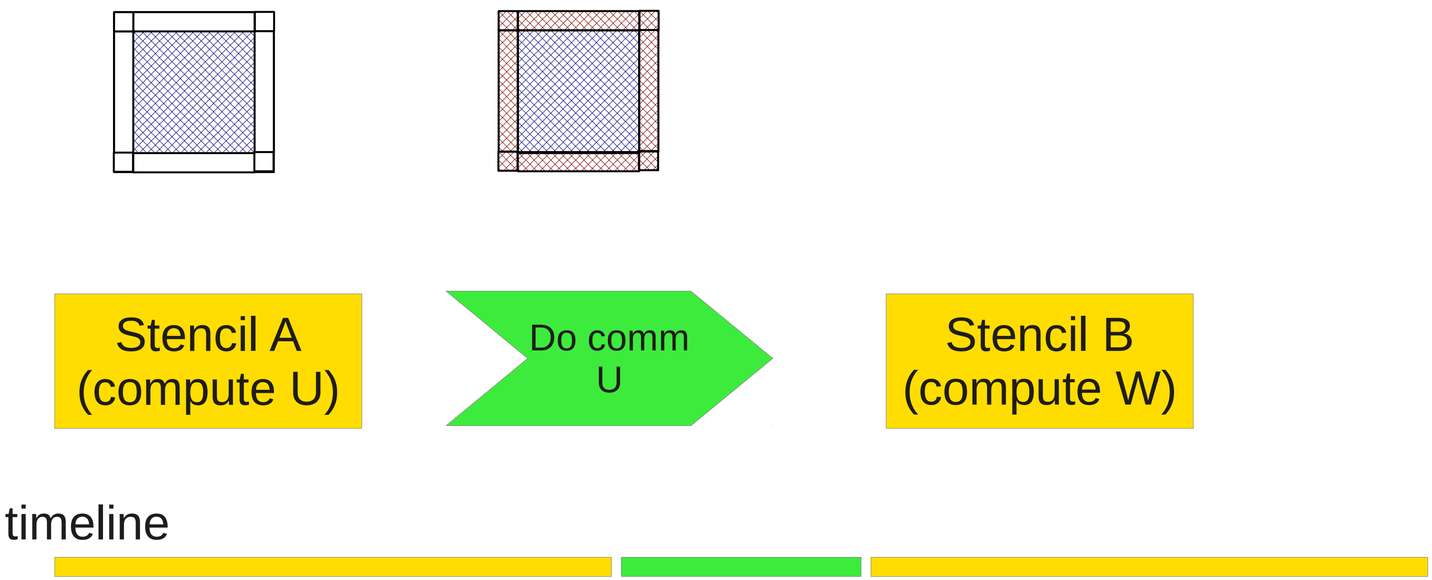
Systematically use asynchronous communication.

The Generic Communication Library (GCL)

- GCL is a library that performs any possible halo exchange pattern of communication.
- Developed at CSCS, in C++03, abstracts the communication layer for halo exchanges.
- Currently uses MPI, it can be adapted to any backend for inter-node communication.
- Features:
  - Interface for asynchronous communication
  - Arbitrary data and grid of processes layouts
  - Handles multiple fields with different halo exchange definitions in a single communication
  - Generic All-to-All
  - CPU and GPU communication (transparent to the user)
  - Several strategies for packing & unpacking (for CPU and GPU)

Asynchronous Communication Using GCL

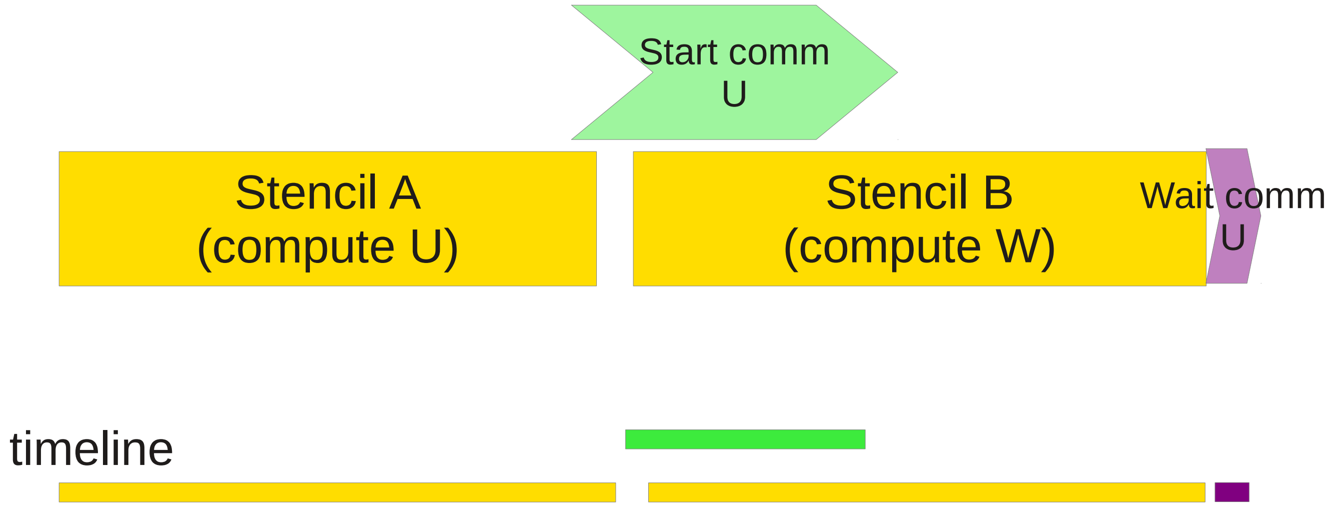
Synchronous communication:



Asynchronous Communication Using GCL

Asynchronous communication:

If Stencil B does not need U field, we can overlap communication with Stencil B computation



Using asynchronous communication with GCL in the Dycore we could reduce communication time by 70%.

```
// Apply qc x advection and start y boundary update
advectionXQC.Apply();
haloUpdateYQC.StartApply();

// apply qv x advection and start y boundary update
advectionXQV. Apply();
haloUpdateYQV.StartApply();

// wait for the boundary update and apply qc y advection
haloUpdateYQC.WaitForApply();
advectionYQC.Apply();

// wait for the boundary update and apply qv y advection
haloUpdateYQV.WaitForApply();
advectionYQV.Apply();
```

Compute field

Immediately
start halo-update

Do other computations
(no wait for exchange)

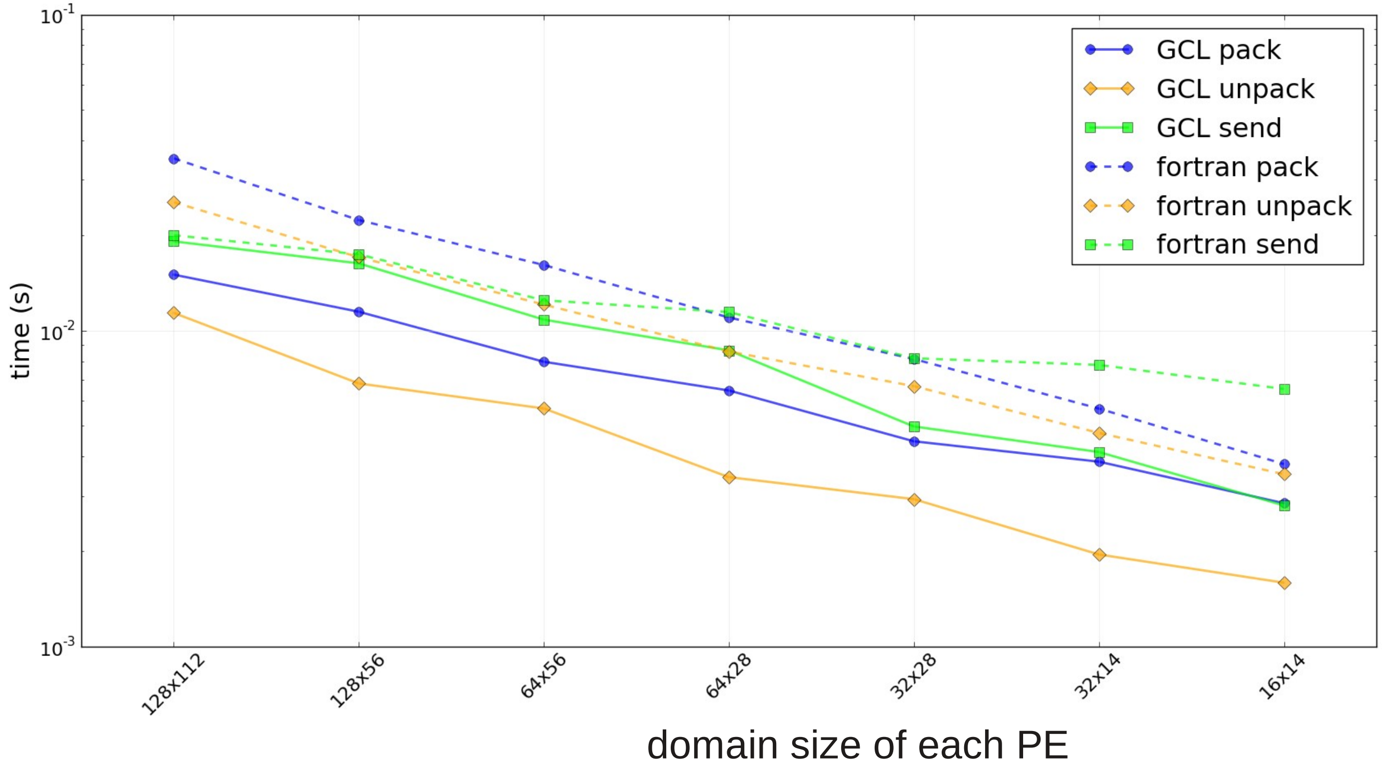
QC needed!!
Wait for exchange

Stencil that uses QC

Communication Performance (CPU):

It took several months to optimize performance (specially in packing & unpacking)

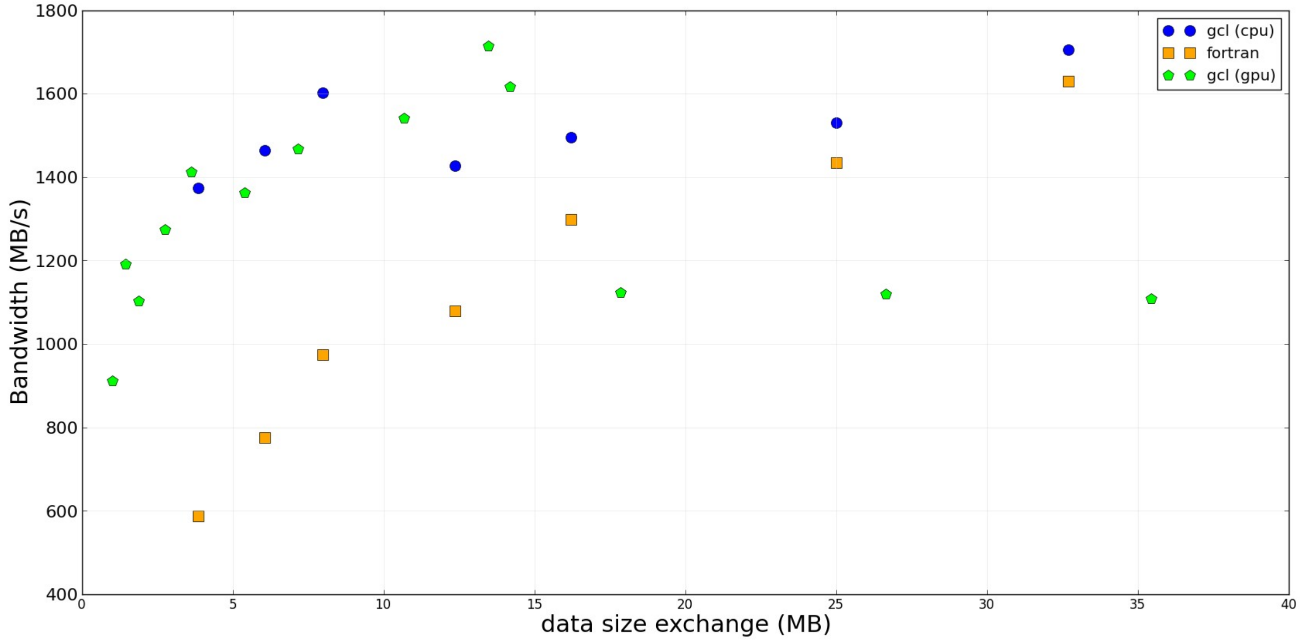
Tests measurements comparing GCL and exch_boundaries subroutine.
3 lines halo exchange of 50 3d fields (60 levels).



grid of 4x4 PE
on XK6 Cray.
Only 1 mpi task per node.
Every task populated with
8 OMP.

Communication Performance:

Bandwidth tests measurements comparing GCL in cpu and gpu mode, and exch_boundaries subroutine.
3 lines halo exchange of 50 3d fields (60 levels)



cpu data:
grid of 4x4 PE on Cray XK6
gpu data:
grid of 2x2 PE on
IBM iDataPlex with FERMI M2090

Only 1 mpi task per node.
Every (CPU) task populated with 8 OMP.

CPU & GPU data extracted from different systems.

Few Notes on GPU communication:

To perform inter-GPU communication,
user can always offload data and perform communication at CPU:

```
cudaMemcpy(buf_cpu, buf_gpu,size, cudaMemcpyDeviceToHost);
MPI_Send(buf_cpu, size,..., MPI_COMM_WORLD)
```

Sender

Receiver

```
MPI_Recv(buf_cpu, size, ···, MPI_COMM_WORLD);
cudaMemcpy(buf_gpu, buf_cpu, size, cudaMemcpyHostToDevice);
```

But this solution offers poor performance.

GCL solution to inter-GPU communication:

Mvapich2/1.8 supports GPU to GPU communication:

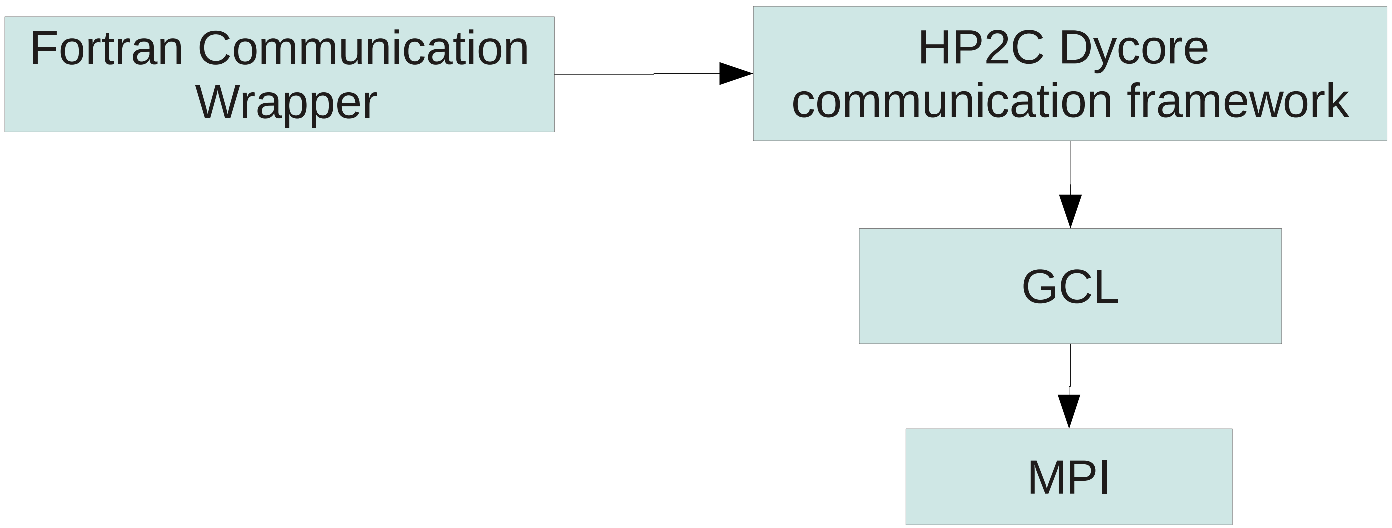
```
MPI_Send( buf_gpu, size, …, MPI_COMM_WORLD)
```

Fortran communcation with GCL:

There exist a Fortran wrapper to the communication framework in C++ that uses GCL.

This provides:

- Reuse C++ code that setup halo exchanges in GCL & minimize code.
- Asynchronous interface that can overlap communication and computation in fortran.



Summary

- Positive experience using GCL to handle communications in C++ Dycore.
- No adaptation of user code needed to use it for GPU.
- For CPU is integrated into the HP2C Dycore, and default communication handler since several months.
  GPU is functional, work in progress tuning performance for packing & unpacking.
- Fortran interface to Communication Framework & GCL is implemented and tested.
- Using asynchronous communication reduces communication time by ~70%.
- Good performance numbers for CPU
- Next:
  - Continue testing and tuning performance.
  - Replace exchg_boundaries() with Fortran wrappers for parts of the code which will run on GPU.