COSMO in HP2C

Carlos Osuna (C2SM)

HP2C

What are we doing in COSMO HP2C?

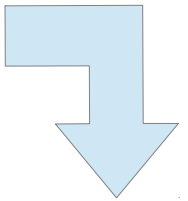
- porting COSMO to run efficiently on x86 & GPU.

The COSMO dynamics is rewritten in a DSEL (C++)

```
DO k = 1, ke
  DO j = jstart, jend
    DO i = istart, iend
      avg(i,j,k) = (data(i+1,j,k) + data(i-1,j,k) + data(i,j+1,k) + data(i,j-1,k))/4.0
    ENDDO
  ENDDO
ENDDO
```

We use a DSEL to specify the equation,

```
static void Do(Context ctx, FullDomain)
  {
    ctx[avg::Center()] = ( ctx[data::At(iplus1)] + ctx[data::At(iminus1)] +
          ctx[data::At(jplus1)] + cx[data::At(jminus1)])/4.0;
  }
```



A hidden backend produces efficient platform dependet code for loops, data structures, blocking strategies, etc. (GPU, x86, ...)

By providing multiple backends, the same user code can run efficiently in different architectures.

GPU

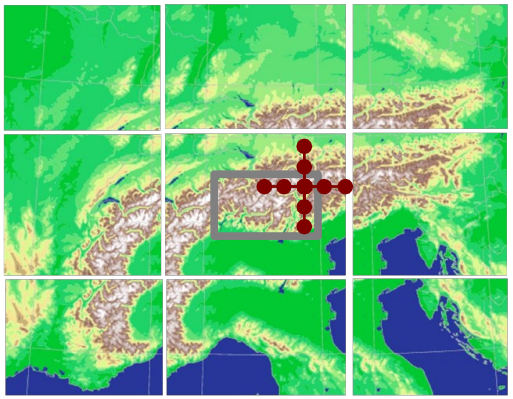
Cray

```
static void Do(Context ctx, FullDomain)
  {
    ctx[avg::Center()] = (
      ctx[data::At(iplus1)] +
      ctx[data::At(iminus1)] +
      ctx[data::At(jplus1)] +
      Cx[data::At(jminus1)]) / 4.0;
  }
```

IBM BlueGene

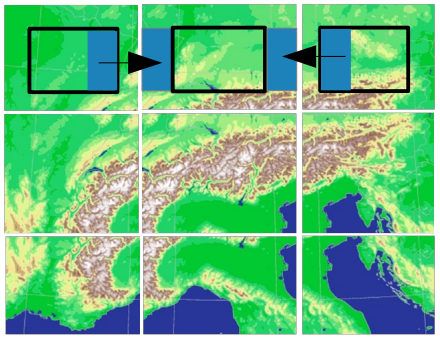
NEC

Porting boundary conditions to HP2C Dycore

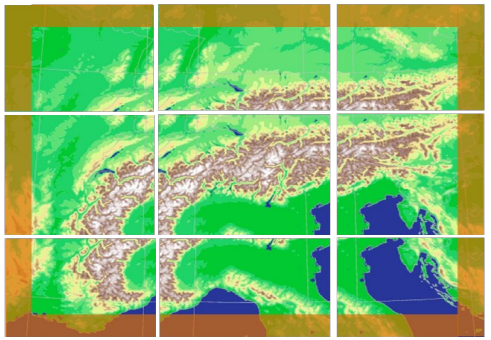


Before running a stencil that requires data a the halos, boundaries must be updated.

From neighbour PE → halo exchange



From boundary files
(for PE at global border) → boundary conditions



Problem 1: Boundary conditions code hard coded.

Boundary conditions are hard coded all around COSMO, every time a boundary needs to be updated after a stencil computation.

```
IF( my_cart_neigh(3) == -1) THEN
  DO k=1, ke
    DO j=1, je
      DO i=iend+1, ie
        u(i,j,k,nnew) = z1 * u_bd(i,j,k,nbd1) + u_bd(i,j,k,nbd2)
      ENDDO
    ENDDO
  ENDDO
ENDIF
```

Code replication :(

Different treatment for u, v or mass point.

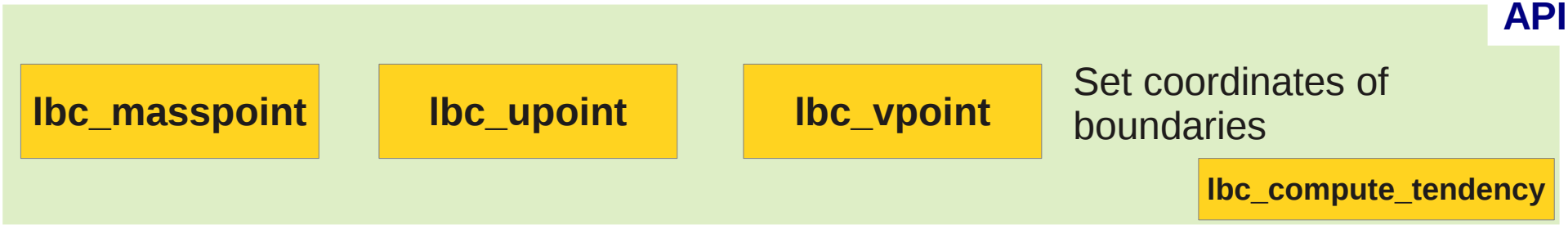
Not well defined functionality for BC :(

We can not hard code in a DSEL :(

Problem 1: How to solve it

Implement BC with subroutines in a module.
Well defined functionality.

```
CALL lbc_masspoint( BCType_ZeroGradient, qr(:,:,:,n_qx), doNS=doNS, doEW=doEW )
```



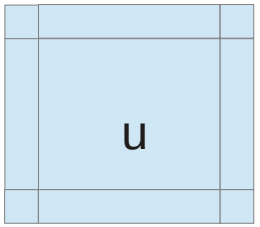
Implementation is provided in a lower layer, but providing an API fixes functionality.

Problem 2: consistency

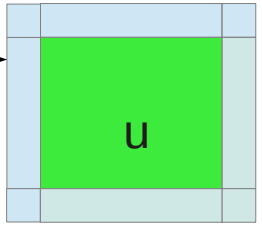
BC code is not tied to the Stencil that requires boundaries.

Example:

initialize_boundaries



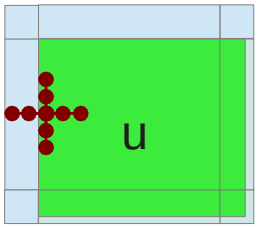
Stencil A



exchg_boundaries(nlines=2)

Because Stencil A does not write at the boundaries, we dont need BC before StencilB

Stencil B
(uses 2 points at the boundary)

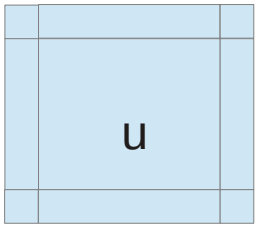


Problem 2: consistency

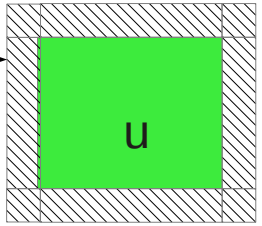
BC code is not tied to the Stencil that requires boundaries.

But

initialize_boundaries

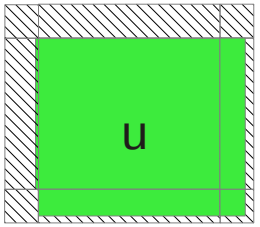


Stencil A.1

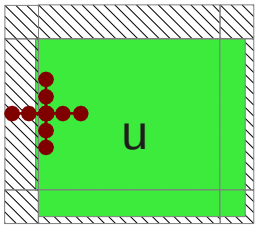


If Stencil A.1 writes at the boundary,
Stencil B uses wrong boundary data.

exchg_boundaries(nlines=2)



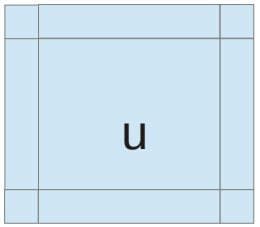
Stencil B
(uses 2 points at the boundary)



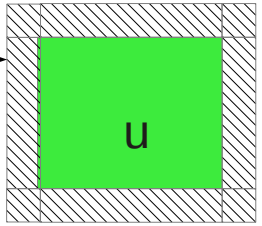
Problem 2: Solution

Always place a BC call before the Stencil that requires boundary data.

initialize_boundaries

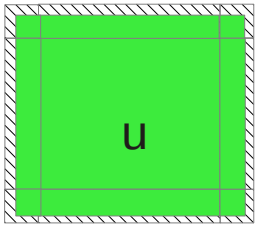


Stencil A.1



If Stencil A.1 writes at the boundary,
Stencil B uses wrong boundary data.

exchg_boundaries(nlines=2)
lbc_upoint() (nlines=2)



Stencil B
(uses 2 points at the boundary)