RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

Automatic Generation of Distributed and Asynchronous HPC Dataflows

Michael Lienhardt,

ONERA, Université Paris-Saclay
France

April 17th 2026

APM @ Etaps

Context

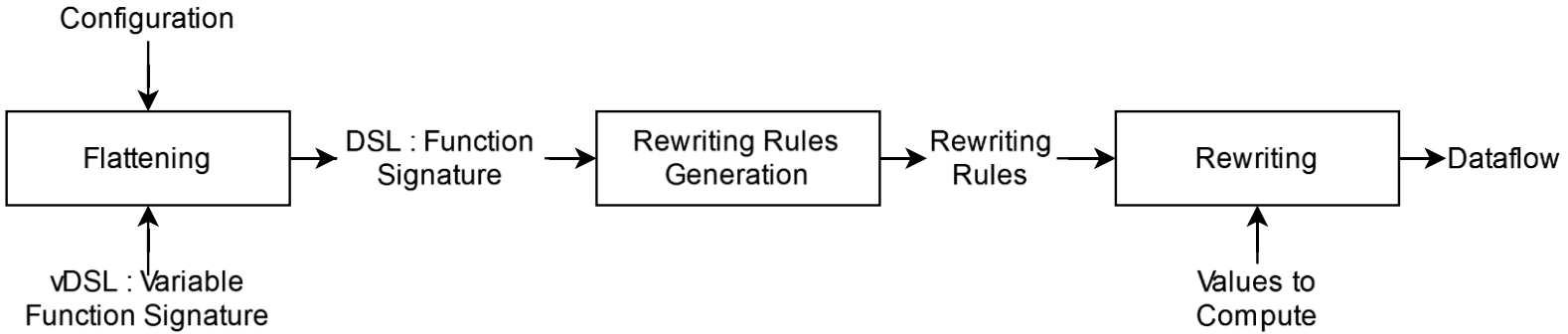
- Computational Fluid Dynamics:
  - Approximates the Navier-stokes equations of fluid dynamics
  - Many methods (balance between *precision*, *speed*, *stability*)
- The computation itself is highly variable
  - Many options (e.g., elsA → ~2000 options)
  - The mesh gives the physical elements to consider
  - The user may have some value of interest (e.g., pressure)
  - Distributed hardware means distributed computation that needs ***consistency***

(including inter-node communications)

Context

- Existing work:
  - managing this variability is a non-distributed context

    Lienhardt, M., ter Beek, M. H., and Damiani, F. *Product lines of dataflows*. J. Syst. Softw. 210 (2024), 111928.
  - Core Idea: use a DSL and term rewriting to produce a dataflow

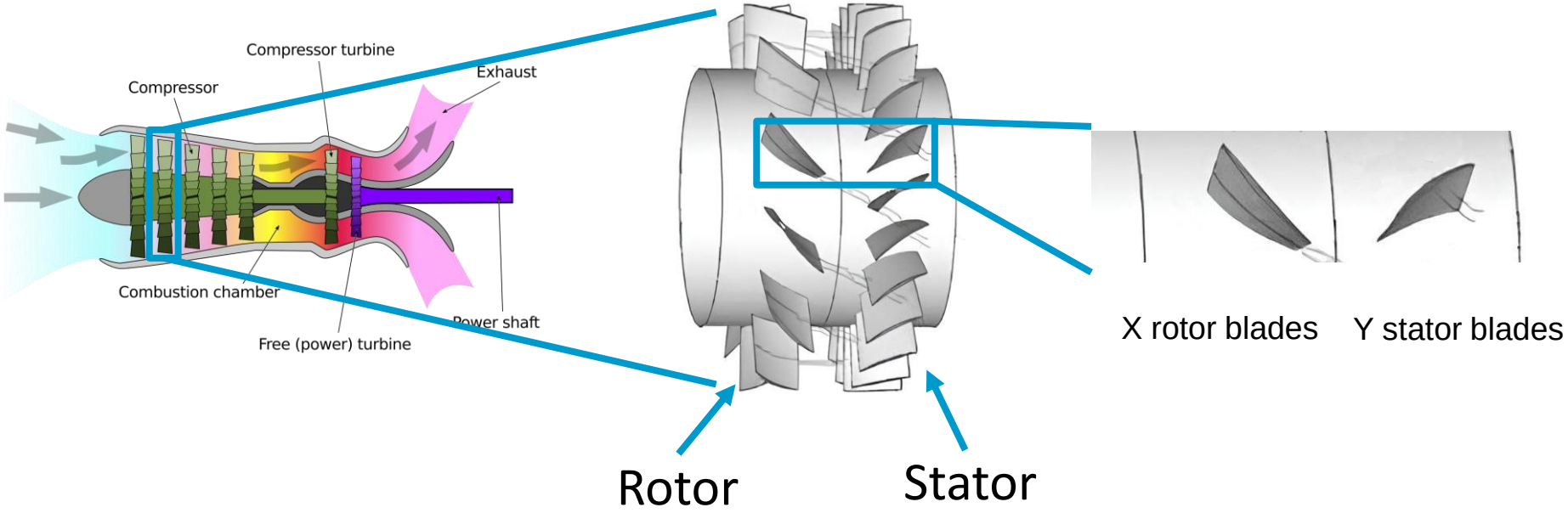


This work

- Build upon previous work to
  - Manage distributed computation
  - Better capture meshes (necessary for distributed computation)

Motivating Example (1/4)

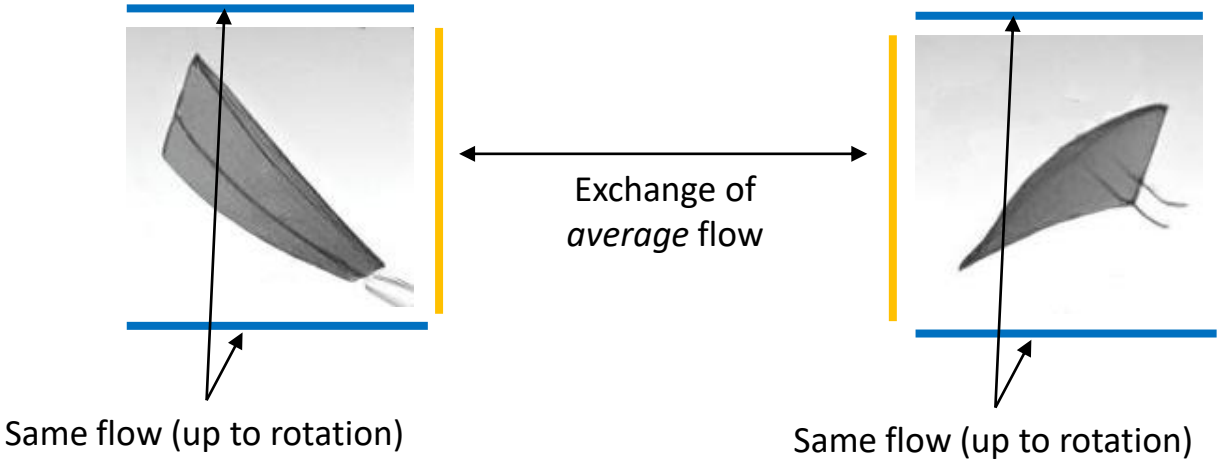
Simulate interaction rotor/stator



Motivating Example (2/4)

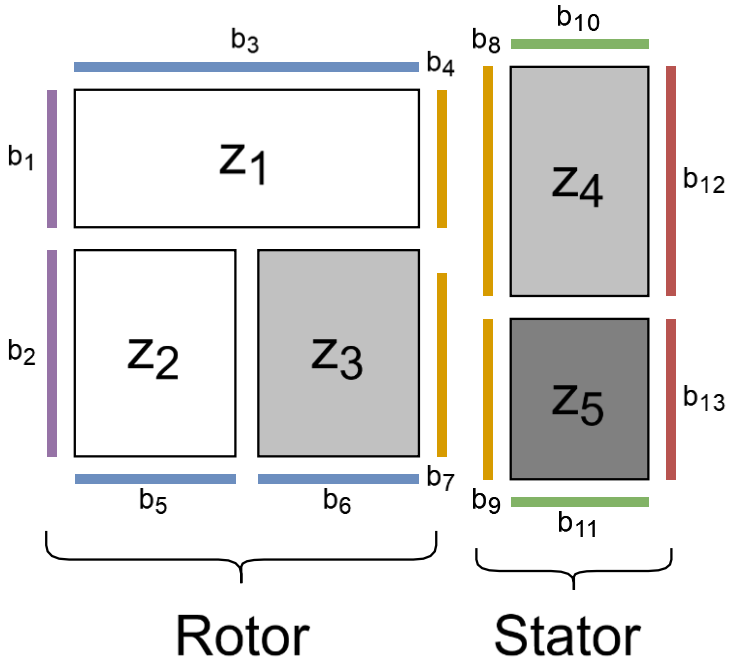
Simulate interaction rotor/stator

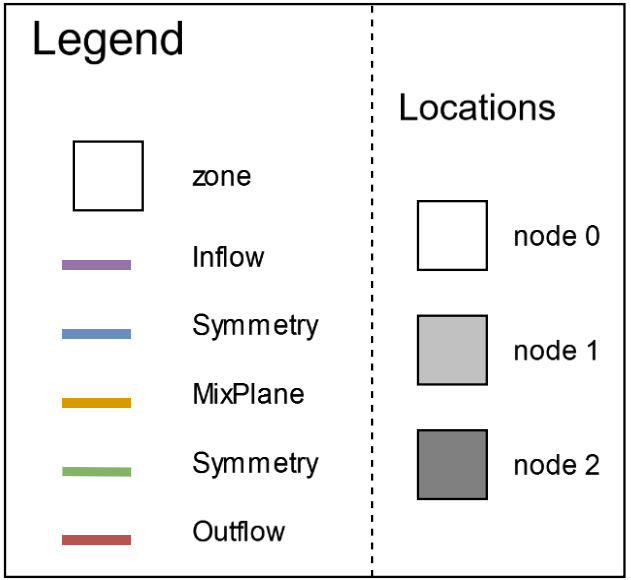
- Instead of simulating the whole wheel, simulate one channel
  - ➢ Less precise simulation, but large speed gain



Motivating Example (3/4)

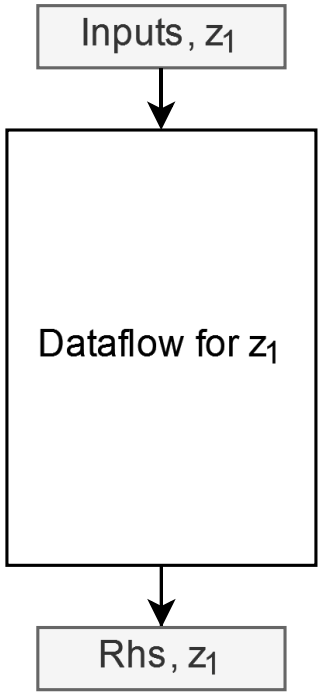
Corresponding Mesh Structure, with example distribution





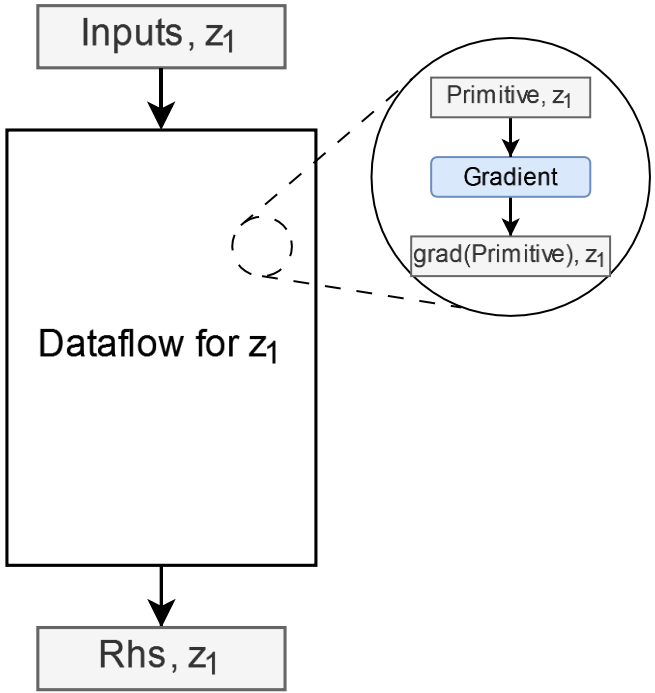
Motivating Example (4/4)

Aspects of the Dataflow for z_1 related to the distributed computation



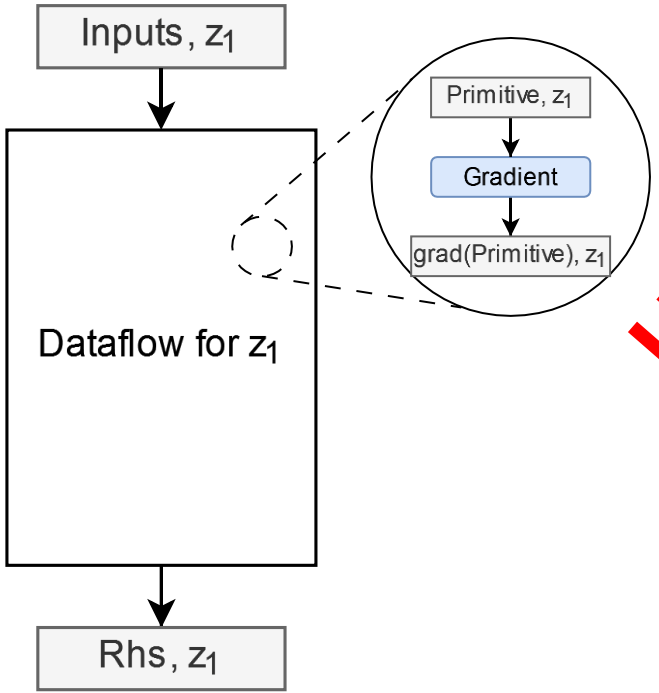
Motivating Example (4/4)

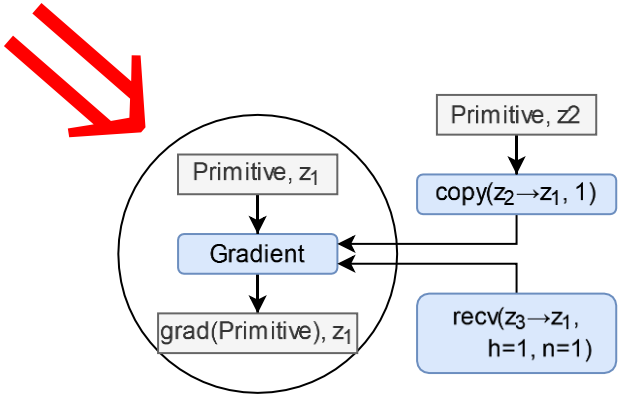
Aspects of the Dataflow for z_1 related to the distributed computation



Motivating Example (4/4)

Aspects of the Dataflow for z_1 related to the distributed computation





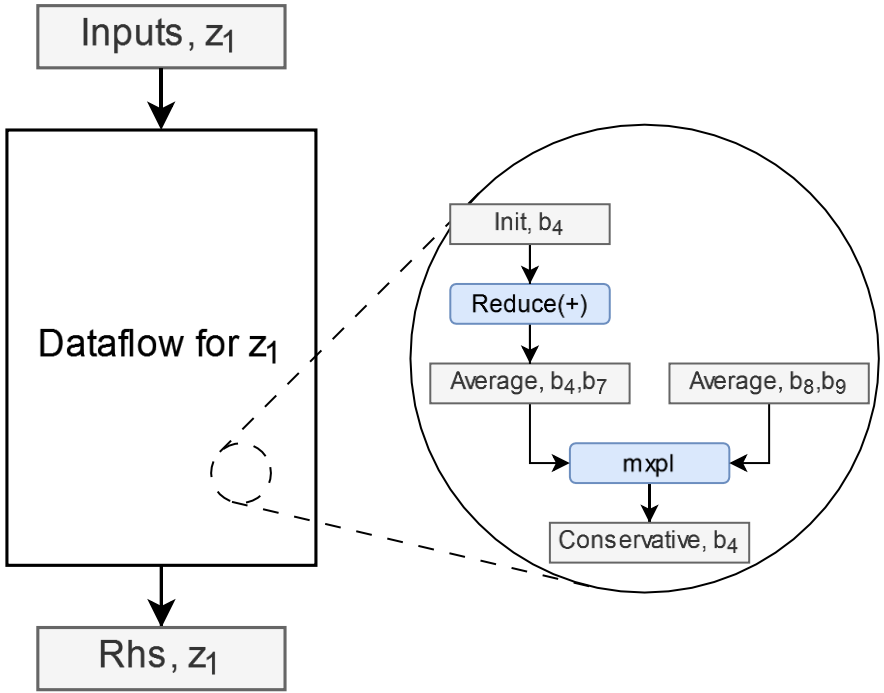
- Stencil Operations
  - Require layer of data from neighboring zones

Ensure that node 1 computes and sends

Primitive, z_3

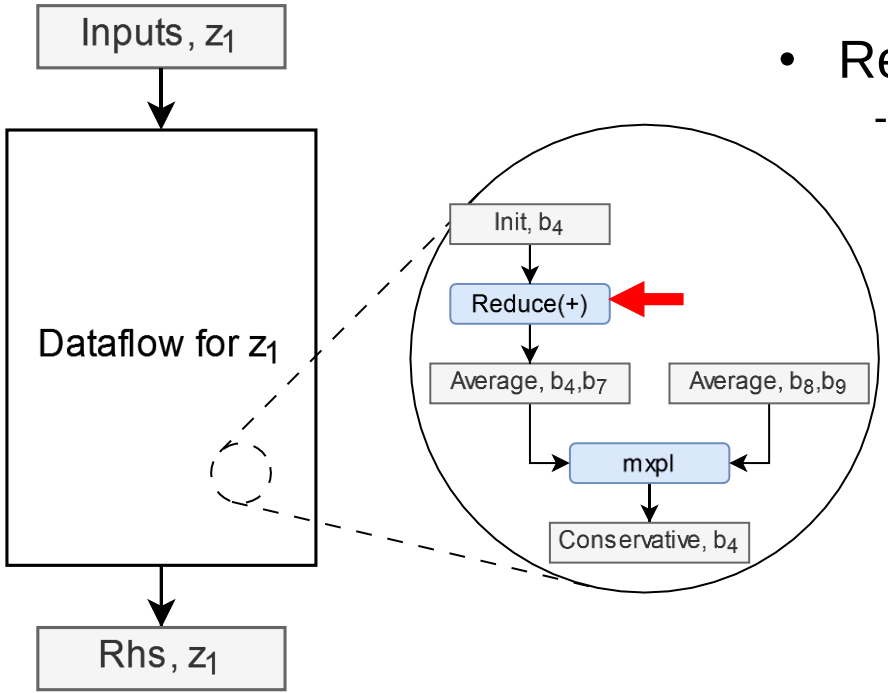
Motivating Example (4/4)

Aspects of the Dataflow for z_1 related to the distributed computation



Motivating Example (4/4)

Aspects of the Dataflow for z_1 related to the distributed computation

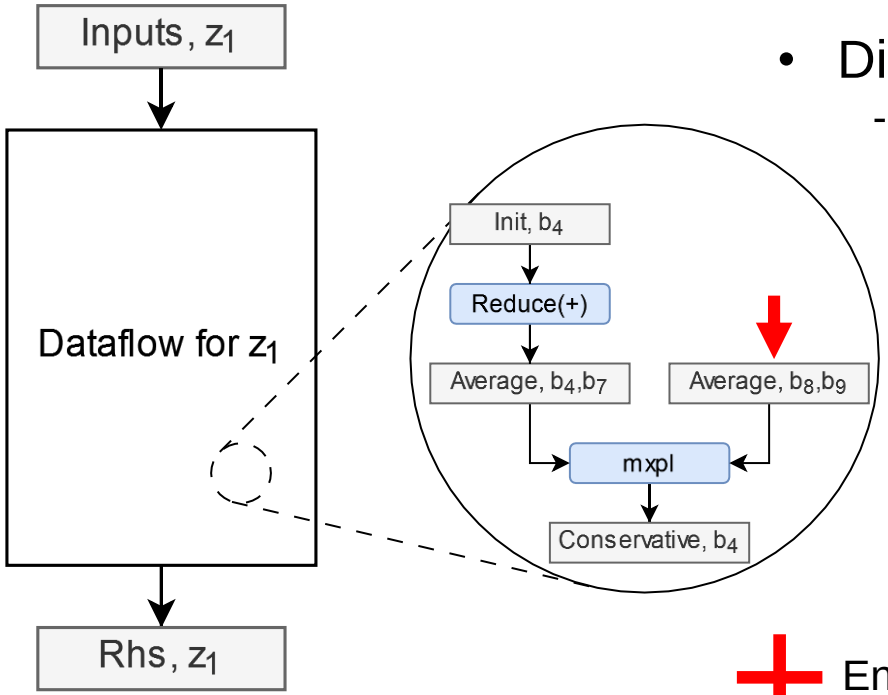


- Reduce Operations
  - Global Communications

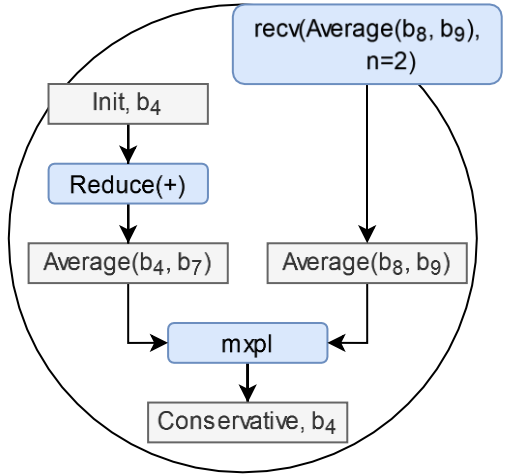
Ensure that nodes 1 and 2 perform the same operation

Motivating Example (4/4)

Aspects of the Dataflow for z_1 related to the distributed computation



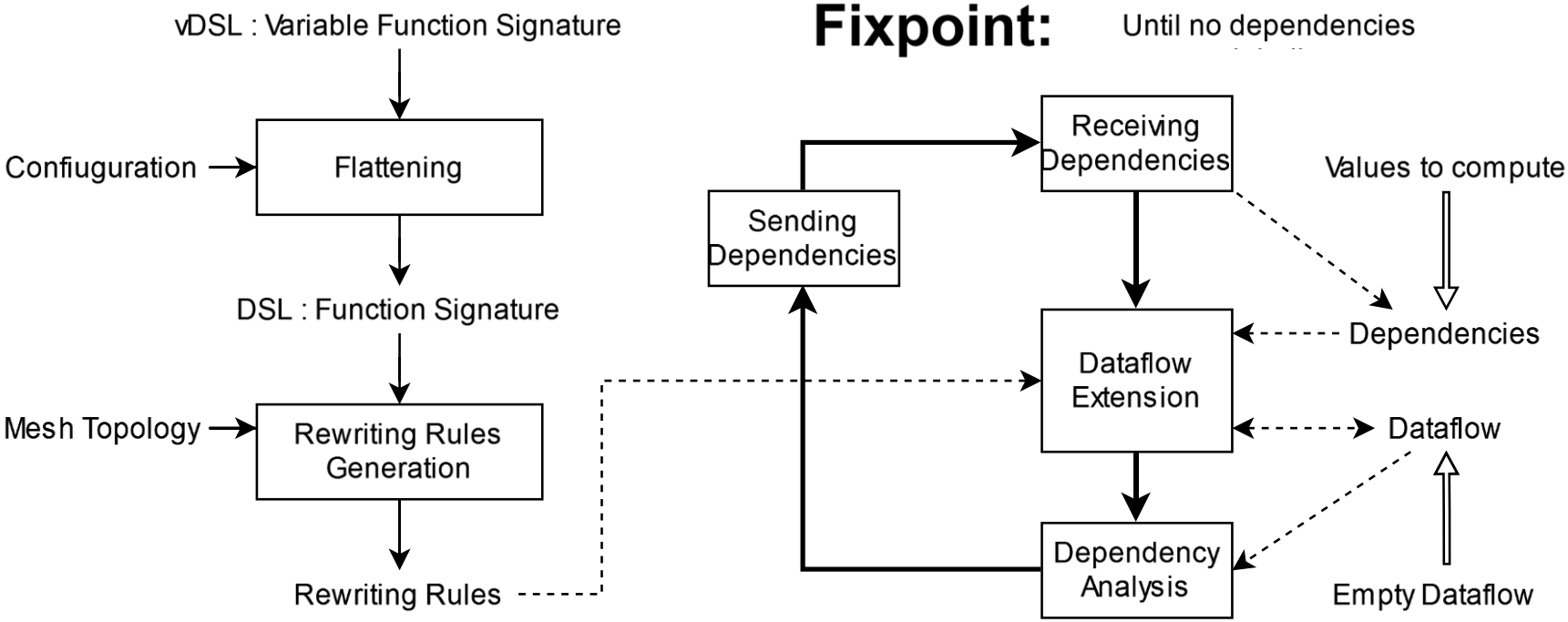
- Distant Data
  - needs data that cannot be computed locally



+ Ensure that node 2 computes and sends Average(b_8, b_9)

Our Approach (1/3)

Updated Pipeline



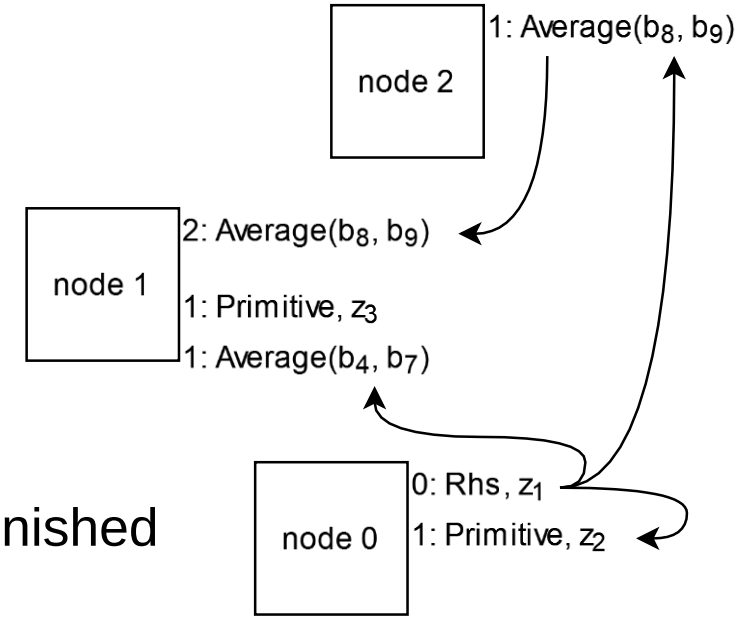
Our Approach (2/3)

Counting dependencies

- Initial dependency = root of a tree
  - each node counts the number of unsolved leaves:
  - step 0: node 0 → 1; node 1 → 0; node 0 → 0
  - step 1: node 0 → 4; node 1 → 0; node 0 → 0
  - step 2: node 0 → 1; node 1 → 0; node 0 → 0
  - step 3: node 0 → 0; node 1 → 0; node 0 → 0

    ⇒ **fixpoint terminates**

- **Note:** the tree is non-deterministic
- Need to collect the fact that all nodes have finished

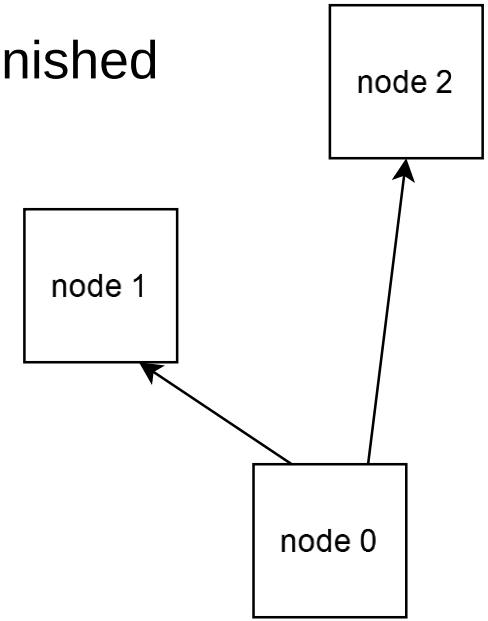


Dependency tree rooted on Rhs, z_1

Our Approach (2/3)

Counting dependencies

- Need to collect the fact that all nodes have finished
  - Define a binary tree structure on the nodes
  - Node *n* has children 2*n*+1 and 2*n*+2
  - When node *n* → 0 and children finished
    ⇒ node n finished
  - When node 0 finished ⇒ fixpoint terminates



- Example:
  - step 0: node 0 → 1; node 1 → 0; node 0 → 0
    ⇒ node 1 and node 2 have finished
  - …
  - step 3: node 0 → 0; node 1 → 0; node 0 → 0
    ⇒ node 0 has finished **⇒ fixpoint terminates**

Our Approach (3/3)

Collecting dependencies

- All locations (zones, sets of BCs, etc) have an *owner*
  - Responsible to ensure that data on that location is computed
- Dependencies: obtained from tasks/data added to the dataflow
- 3 triggers of dependencies:

1. root data `value, L` where *owner*(L_i) ≠ self

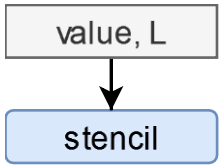
➢ send request to compute `value, L` to its owner

Our Approach (3/3)

Collecting dependencies

- All locations (zones, sets of BCs, etc) have an *owner*
  - Responsible to ensure that data on that location is computed
- Dependencies: obtained from tasks/data added to the dataflow
- 3 triggers of dependencies:

2. stencil operation



➢ send request to compute value, L_1, …, value, L_n to their owners,

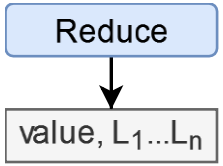
where $L_1 \ldots L_n$ = neighbors(L)

Our Approach (3/3)

Collecting dependencies

- All locations (zones, sets of BCs, etc) have an *owner*
  - Responsible to ensure that data on that location is computed
- Dependencies: obtained from tasks/data added to the dataflow
- 3 triggers of dependencies:

3. reduce operation



- if self = *owner*(L_1…L_n), send request to compute value, L_1...L_n to all *owner*(L_i)
- else, send request to compute value, L_1...L_n to *owner*(L_1…L_n)

Conclusion

This Work:

- Approach to automatically manage a highly variable computation on distributed hardware
- Includes a precise management of meshes

The mechanism to actually add the communication tasks in the dataflow will be presented in the paper

Future Work:

- Heterogeneous hardware
  - Functions might not be implemented on some hardward
  - Mesh distribution must be adapted to hardware characteristics
- Communication Optimization
  - By sending larger layers, it is possible to reduce the number of stencil-triggered communications

Thank You

Questions?