ONERA

THE FRENCH AEROSPACE LAB

Distributor2 Documentation

Release 3.7

/ELSA/MU-12016/V3.7

Oct 19, 2023

CONTENTS

CHAPTER

ONE

PREAMBLE

This module provides functions to distribute blocks on a given number of processors. At the end of the process, each block will have a number corresponding to the processor it must be affected to for a balanced computation, depending on given criterias. This module doesn't perform splitting (see the Transform module for that).

This module is part of Cassiopee, a free open-source pre- and post-processor for CFD simulations.

For use with the array interface, you have to import Distributor2 module:

```
import Distributor2 as D2
```

For use with the pyTree interface:

```
import Distributor2.PyTree as D2
```

CHAPTER

TWO

LIST OF FUNCTIONS

– Automatic load balance

Distributor2.distribute(arrays, NProc[, ...])	Distribute zones over NProc processors.
Distributor2.PyTree.distribute(t, NProc[, ...])	Distribute a pyTree over processors.

– Various operations

Distributor2.PyTree.addProcNode(t, proc)	Add a "proc" node to zones with the given proc value.
Distributor2.PyTree.getProc(t)	Return the value of proc node.
Distributor2.PyTree.getProcDict(t[, ...])	Return the proc of a zone in a dictionary proc['zoneName'].
Distributor2.PyTree.getProcList(t[, NProc, sort])	Return the list of zones for each proc.
Distributor2.PyTree.copyDistribution(a, b)	Copy distribution of a in b.
Distributor2.PyTree.printProcStats(t[, ...])	Print stats dictionary.
Distributor2.Mpi.redispatch(t[, graph])	Redistribute tree from graph.

CHAPTER

THREE

CONTENTS

Distributor2.**distribute**(*A*, *NProc*, *prescribed=None*, *perfo=None*, *weight=None*, *com=None*, *algorithm='graph'*, *mode='nodes'*, *nghost=0*)

Distribute automatically the blocks amongst NProc processors.

- prescribed is a list of blocks that are forced to be on a given processor.

For instance, prescribed[2] = 0 means that block 2 MUST be affected to processor 0.

- perfo is a tuple or a tuple list for each processor.

Each tuple describes the relative weight of solver CPU time regarding the communication speed and latence (solverWeight, latenceWeight, comSpeedWeight).

- weight is a list of weight for each block indicating the relative cost for solving each block.
- com is a ixj matrix describing the volume of points exchanged between bloc i and bloc j.
- algorithm can be chosen in: 'gradient', 'genetic', 'fast', 'graph'
- mode='node', 'cells': optimize distribution of block considering node (cells) numbers.
- nghost: take into account ghost cells (only for structured grids)

Parameters

- **a** (`[array, list of arrays]`) – Input data
- **N** (`int`) – number of processors
- **prescribed** (`list of ints`) – list of prescribed blocks
- **perfo** (`list of tuples`) – list of performance for each processor
- **weight** (`list of ints`) – list of weight for each block
- **algorithm** (`string`) – ['gradient', 'genetic', 'fast', 'graph']

- **nghost** (int) – number of ghost cells present in the mesh

The function output is a stats dictionary. stat['distrib'] is a vector describing the attributed processor for each block, stats['meanPtsPerProc'] is the mean number of points per proc, stats['varMin'] is the minimum variation of number of points, stats['varMax'] is the maximum variation of number of points, stats['varRMS'] is the mean variation of number of points, stats['nptsCom'] is the number of points exchanged between processors for communication, stats['comRatio'] is the ratio between the number of points exchanged between processors in this configuration divided by the total number of matching/overlap boundary points, stats['adaptation'] is the value of the optimized function.

Example of use:

- Distribute arrays (array):

```
# - distribute (array) -
import Generator as G
import Distributor2 as D2
import numpy
from Converter.Internal import E_NpyInt

# Distribution sans communication entre blocs
N = 11
arrays = []
for i in range(N):
    a = G.cart( (0,0,0), (1,1,1), (10+i, 10, 10) )
    arrays.append(a)
out = D2.distribute(arrays, NProc=5); print(out)

# Distribution avec des perfos differentes pour chaque proc
out = D2.distribute(arrays, NProc=3, perfo=[(1,0,0), (1.2,0,0), (0.2,0,
↪0)]); print(out)

# Distribution avec forcage du bloc 0 sur le proc 1, du bloc 2 sur le␣
↪proc 3
# -1 signifie que le bloc est a equilibrer
prescribed = [-1 for x in range(N)]
prescribed[0] = 1; prescribed[2] = 3
out = D2.distribute(arrays, NProc=5, prescribed=prescribed); print(out)

# Distribution avec communications entre blocs, perfos identique pour␣
↪tous
# les procs
volCom = numpy.zeros((N, N), dtype=E_NpyInt)
```

(continues on next page)

(continued from previous page)

```
volCom[0,1] = 100; # Le bloc 0 echange 100 pts avec le bloc 1
out = D2.distribute(arrays, NProc=5, com=volCom, perfo=(1,0.,0.1));␣
↪print(out)

# Distribution avec des solveurs differents pour les blocs (le solveur␣
↪est 2
# fois plus couteux pour les bloc 2 et 4)
out = D2.distribute(arrays, weight=[1,2,1,2,1,1,1,1,1,1,1], NProc=3);␣
↪print(out)
```

Distributor2.PyTree.**distribute**(*A*, *NProc*, *prescribed=None*, *perfo=None*, *weight=None*, *useCom='match'*, *algorithm='graph'*, *mode='nodes'*, *nghost=0*)

Distribute automatically the blocks amongst NProc processors.

With the pyTree interface, the user-defined node .Solver#Param/proc is updated with the attributed processor number.

If useCom=0, only the grid number of points is taken into account. If useCom='match', only match connectivities are taken into account. if useCom='overlap', only overlap connectivities are taken into account. if useCom='bbox', overlap between zone bbox is taken into account. if useCom='ID', ID (interpolation or match) and IBCD (IBM points) are taken into account. If useCom='all', matching and overlap communications are taken into account.

When using distributed trees, prescribed must be a dictionary containing the zones names as key, and the prescribed proc as value. weight is also a dictionary where the keys are the zone names and the weight as the value. It is not mandatory to assign a weight to all the zones of the pyTree. Default value is assumed to be 1, only different weight values can be assigned to zones. t can be either a skeleton or a loaded skeleton pyTree for useCom=0 or useCom='match', but must be a loaded skeleton tree only for the other settings.

Parameters

- **a** (`[pyTree, base, zone, list of zones]`) – Input data
- **N** (`int`) – number of processors
- **prescribed** (`dictionary`) – dictionary of prescribed block (optional)
- **perfo** (`list of tuples`) – list of perfo for each processor (optional)
- **weight** (`dictionary`) – dictionary of weights for block (optional)

- **useCom** (['0', 'all', 'match', 'overlap', 'bbox', 'ID']) – tell what to use to measure communication volumes
- **algorithm** (string) – ['gradient', 'genetic', 'fast', 'graph']
- **nghost** (int) – number of ghost cells present in the mesh

Example of use:

- Distribute zones (pyTree):

```
# - distribute (pyTree) -
import Generator.PyTree as G
import Distributor2.PyTree as D2
import Converter.PyTree as C
import Connector.PyTree as X

N = 11
t = C.newPyTree(['Base'])
pos = 0
for i in range(N):
    a = G.cart((pos,0,0), (1,1,1), (10+i, 10, 10))
    pos += 10 + i - 1
    t[2][1][2].append(a)
t = X.connectMatch(t)

# Distribute on 3 processors
t, stats = D2.distribute(t, 3)
C.convertPyTree2File(t, 'out.cgns')
```

3.1 Various operations

Distributor2.PyTree.**addProcNode**(*a*, *NProc*)

Add a ".Solver#Param/proc" node to all zones of a with given value. Exists also as in place version (_addProcNode) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **NProc** (int) – proc to be set

Returns

reference copy of a

Return type

identical to input

Example of use:

- Add a proc node (pyTree):

```
# - addProcNode (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Generator.PyTree as G
import Distributor2.PyTree as D2

a = G.cart((0,0,0), (1,1,1), (10,10,10))
D2._addProcNode(a, 12)
Internal.printTree(a)
#>> ['cart',array(shape=(3, 3),dtype='int32',order='F'),[3 sons],'Zone_t
↪']
#>>   |_['ZoneType',array('b'Structured'',dtype='|S1'),[0 son],'ZoneType_
↪t']
#>>   | ...
#>>   |_['.Solver#Param',None,[1 son],'UserDefinedData_t']
#>>       |_['proc',array([12],dtype='int32'),[0 son],'DataArray_t']
C.convertPyTree2File(a, 'out.cgns')
```

Distributor2.PyTree.**getProc**(*a*)

Return the proc value of a zone or a list of zones.

Parameters
a ([pyTree, base, zone, list of zones]) – input data

Returns
the affected proc of zone

Return type
int or list of ints (for multiple zones)

Example of use:

- Get proc of a zone (pyTree):

```
# - getProc (pyTree) -
import Generator.PyTree as G
import Distributor2.PyTree as D2

a = G.cart((0,0,0), (1,1,1), (10,10,10))
a = D2.addProcNode(a, 12)
proc = D2.getProc(a); print(proc)
#>> 12
```

Distributor2.PyTree.**getProcDict**(*a*, *prefixByBase=False*)

Return a dictionary where procDict['zoneName'] is the no of proc affected to zone 'zoneName'.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **prefixByBase** (boolean) – if true, add base prefix to zone name

Returns
the dictionary of zone/proc.

Return type
dictionary

Example of use:

- Return the dictionary of zones/proc (pyTree):

```
# - getProcDict (pyTree) -
import Generator.PyTree as G
import Distributor2.PyTree as D2
import Converter.PyTree as C
import Connector.PyTree as X

N = 11
t = C.newPyTree(['Base'])
pos = 0
for i in range(N):
    a = G.cart((pos,0,0), (1,1,1), (10+i, 10, 10))
    pos += 10 + i - 1
    t[2][1][2].append(a)

t = X.connectMatch(t)
t, stats = D2.distribute(t, 3)

proc = D2.getProcDict(t)
zoneNames = C.getZoneNames(t, prefixByBase=False)
for z in zoneNames: print(z, proc[z])

# - or with base prefix -
proc = D2.getProcDict(t, prefixByBase=True)
zoneNames = C.getZoneNames(t, prefixByBase=True)
for z in zoneNames: print(z, proc[z])
```

Distributor2.PyTree.**getProcList**(*a*, *NProc=None*)

Return procList where procList[proc] is a list of zone names attributed to the proc processor.

Parameters
a ([pyTree, base, zone, list of zones]) – input data

Returns
the affected proc of zone

Return type
int or list of ints

Example of use:

- Return the list of zones affected to a proc (pyTree):

```
# - getProcList (pyTree) -
import Generator.PyTree as G
import Distributor2.PyTree as D2
import Converter.PyTree as C
import Connector.PyTree as X

N = 11
t = C.newPyTree(['Base'])
pos = 0
for i in range(N):
    a = G.cart((pos,0,0), (1,1,1), (10+i, 10, 10))
    pos += 10 + i - 1
    t[2][1][2].append(a)

t = X.connectMatch(t)
t, stats = D2.distribute(t, 3)

procList = D2.getProcList(t)
print(procList)
```

Distributor2.PyTree.**copyDistribution**(*a*, *b*)

Copy the distribution from b to a matching zones by their name. Exists also as in place version (_copyDistribution) that modifies a and returns None.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **b** ([pyTree, base, zone, list of zones]) – original data

Returns
modifie reference copy of a

Return type
same as input data

Example of use:

- Copy proc distribution from one tree to another (pyTree):

```
# - copyDistribution (pyTree) -
import Converter.PyTree as C
import Distributor2.PyTree as D2
import Generator.PyTree as G

# Case
N = 11
t = C.newPyTree(['Base'])
pos = 0
for i in range(N):
    a = G.cart((pos,0,0), (1,1,1), (10+i, 10, 10))
    a[0] = 'cart%d'%i
    pos += 10 + i - 1
    D2._addProcNode(a, i)
    t[2][1][2].append(a)

t2 = C.newPyTree(['Base'])
for i in range(N):
    a = G.cart((pos,0,0), (1,1,1), (10+i, 10, 10))
    a[0] = 'cart%d'%i
    pos += 10 + i - 1
    t2[2][1][2].append(a)
t2 = D2.copyDistribution(t2, t)
C.convertPyTree2File(t2, 'out.cgns')
```

Distributor2.Mpi.**redispatch**(*a*)

Redispatch a tree where a new distribution is defined in the node 'proc'.

Parameters
a ([pyTree, base, zone, list of zones]) – input data

Returns
modifie reference copy of a

Return type
same as input data

Example of use:

- Redispatch a tree (pyTree):

```
# - redispatch (pyTree) -
import Converter.PyTree as C
import Distributor2.PyTree as D2
import Distributor2.Mpi as D2mpi
import Converter.Mpi as Cmpi
import Connector.PyTree as X
import Converter.Internal as Internal
import Generator.PyTree as G

# Case
N = 11
t = C.newPyTree(['Base'])
pos = 0
for i in range(N):
    a = G.cart((pos,0,0), (1,1,1), (10+i, 10, 10))
    pos += 10 + i - 1
    t[2][1][2].append(a)
t = X.connectMatch(t)
if Cmpi.rank == 0: C.convertPyTree2File(t, 'in.cgns')
Cmpi.barrier()

# lecture du squelette
a = Cmpi.convertFile2SkeletonTree('in.cgns')

# equilibrage 1
(a, dic) = D2.distribute(a, NProc=Cmpi.size, algorithm='fast', useCom=0)

# load des zones locales dans le squelette
a = Cmpi.readZones(a, 'in.cgns', rank=Cmpi.rank)

# equilibrage 2 (a partir d'un squelette charge)
(a, dic) = D2.distribute(a, NProc=Cmpi.size, algorithm='gradient1',
                         useCom='match')

Cmpi._convert2PartialTree(a)
D2mpi._redispatch(a)

# force toutes les zones sur 0
zones = Internal.getNodesFromType(a, 'Zone_t')
for z in zones:
```

(continues on next page)

(continued from previous page)

```
    nodes = Internal.getNodesFromName(z, 'proc')
    Internal.setValue(nodes[0], 0)

D2mpi._redispatch(a)

# Reconstruit l'arbre complet a l'ecriture
Cmpi.convertPyTree2File(a, 'out.cgns')
```

Distributor2.PyTree.**printProcStats**(*a*, *stats=None*, *NProc=None*)

Print statistics for each processor: number of points and list of zones names.

Parameters

- **a** ([pyTree, base, zone, list of zones]) – input data
- **stats** (Python dictionary) – dictionary obtained from Distributor2.distribute
- **NProc** (integer) – number of processors

Returns

None

Example of use:

- Print procs statistics after distribution of a tree (pyTree):

```
# - printProcStats (pyTree) -
import Generator.PyTree as G
import Distributor2.PyTree as D2
import Converter.PyTree as C
import Connector.PyTree as X

N = 11
t = C.newPyTree(['Base'])
pos = 0
for i in range(N):
    a = G.cart((pos,0,0), (1,1,1), (10+i, 10, 10))
    pos += 10 + i - 1
    t[2][1][2].append(a)
t = X.connectMatch(t)

# Distribute on 3 processors
t, stats = D2.distribute(t, 3)
```

(continues on next page)

(continued from previous page)

```
# With stats and NProc
D2.printProcStats(t, stats, NProc=3)
# NProc is guessed from stats
D2.printProcStats(t, stats)
# All are guessed from t
D2.printProcStats(t)

C.convertPyTree2File(t, 'out.cgns')
```

Note: new in version 2.7.

CHAPTER

FOUR

INDEX

- genindex
- modindex
- search