# Task-based parallelization of a CFD code over a Runtime System

## FLUSEPA : A Navier-Stokes solver with bodies in relative motion

JM. Couteyen,J. Roman,P. Brenner

May 20, 2015

# Outlines

1. Introduction
2. Parallelization : OpenMP/MPI Approach
3. Parallelization : design of a task-based version

# Outlines

1. Introduction
   - Aerodynamics and CFD at Airbus DS
   - Flusepa
2. Parallelization : OpenMP/MPI Approach
3. Parallelization : design of a task-based version

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# FLUSEPA

## About

- Navier-Stokes code developed for more than 20 years
- Development choices driven by unsteady simulations (Stage Separations, Takeoff Blast wave...)

## Characteristics

- Finite Volume
- Unstructured meshes
- Changing topology (bodies in relative motion)
- Geometric intersection (CHIMERA-like)
- Temporal adaptive time integration

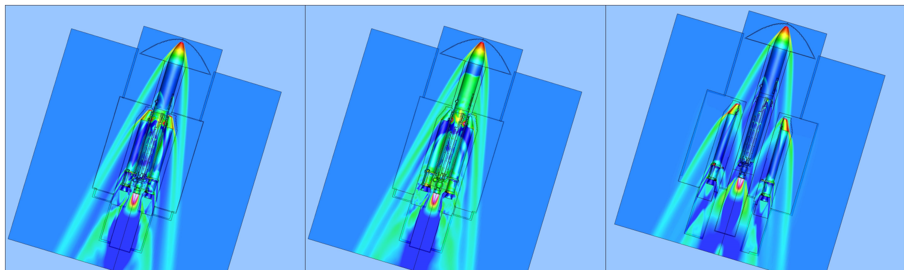
# General scheme

**Algorithm 1** General iteration

1: Aerodynamic Solver
2: Current_kinematic+=Kinematic Computation
3: **if** Important motion since last intersection **then**
4:     Body Displacement(Current_kinematic)
5:     Intersection Computation
6:     Current_kinematic=0
7: **end if**

# Separation computation

## A5 booster separation



AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Outlines

1 Introduction

2 Parallelization : OpenMP/MPI Approach
- Temporal Adaptive Solver
- Response to the limits

3 Parallelization : design of a task-based version

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Parallelization issue : Temporal Adaptive Solver

- Explicit solver (considered competitive for unsteady phenomenon)

## Temporal adaptive solver principles

- Compute each cell near its maximum physical time step
- Interpolation for boundaries between temporal levels

## Difficulties

- Load Balancing
- Temporal zones evolve during computation
- A lot of synchronizations implied by current algorithm

# Temporal Adaptive

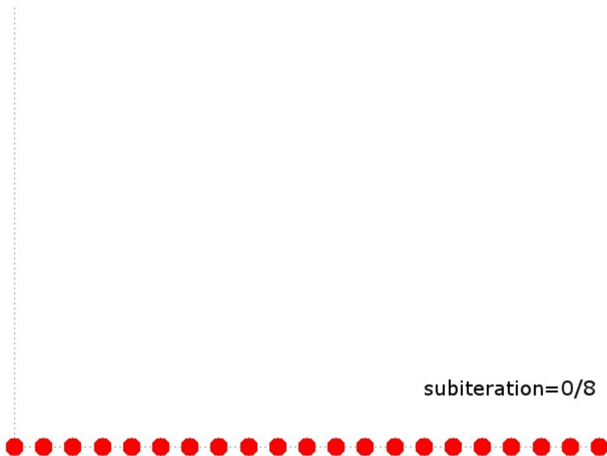
**Algorithm 2** Temporal Adaptive in Flusepa, one general iteration of the aerodynamic solver

1: Timestep computation for every cell
2: Classification of every cell inside a temporal class.
3: *Temporal adapative loop:*
4: **for** subiteration=1 **to** $2^\theta + 1$ **do**
5: $\tau = 0$
6: **for** $tmp = 1$ **to** $\theta + 1$ **do**
7: **if** $(mod(subiteration - 1, 2^{tmp}) == 0)$ **then**
8: $\tau = tmp$
9: **end if**
10: **end for**
11: *Predictor (0 to* $\tau$*)*
12: **if** $\tau \neq \theta$ **then**
13: Intensive repositionning of $\tau + 1$ temporal class
14: **end if**
15: **for** $\tau' = \tau$ **to** 0 **do**
16: *Corrector*
17: **end for**
18: **end for**

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

[2,2,2,1,1,1,1,0,1,1,1,1,2,2,2,2,2,2,2,2,2]



[2,2,2,1,1,1,1,0,1,1,1,1,2,2,2,2,2,2,2,2,2]

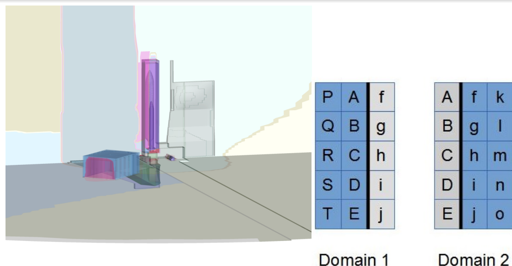
AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Aerodynamic solver parallelization

## Domain Decomposition

Partitioning the mesh in several parts, each given to a different process




## Ghost Cells

A cell belong to one and only one domain, but at the borders cells are replicated (Border faces are duplicated)

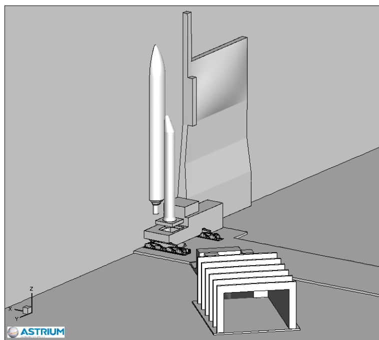
# Aerodynamic solver parallelization

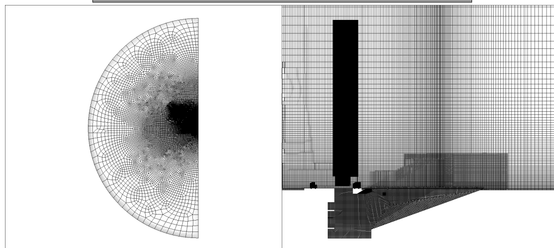
## Asynchronous Point-to-Point Communications

```
computation_part1()
foreach neighbour:
    Isend(border_cells)
    Irecv(ghost_cells)
computation_part2()
foreach neighbour:
    Wait(border_cells) #Isend
    Wait(ghost_cells) #Irecv
computation_part3()
```

- Computations overlapping communications
- Communication only when sharing a border of the given temporal level with neighbours

# Take-off blast wave

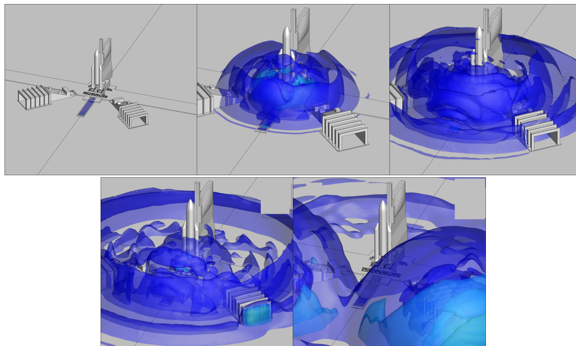


AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

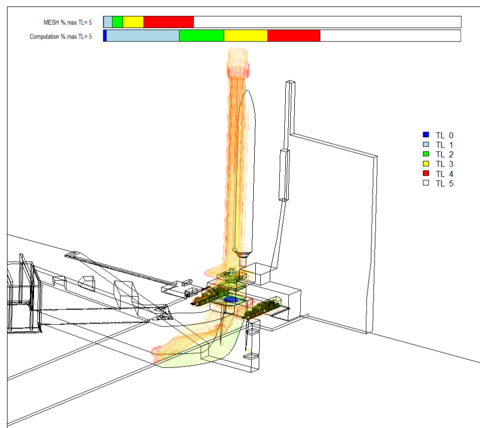
# Take-off blast wave



AIRBUS DEFENCE & SPACE

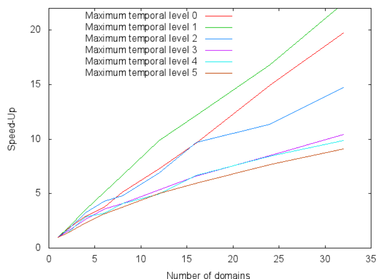
Inria INVENTEURS DU MONDE NUMÉRIQUE

# Perf Analysis

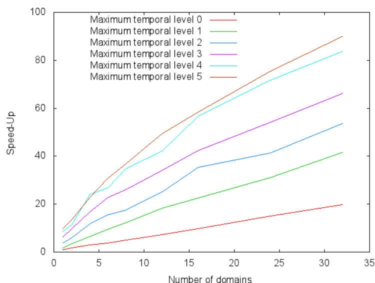



Location of cells according to their temporal level (5 temporal levels)

# Even with low scalability, Temporal Adaptive is competitive

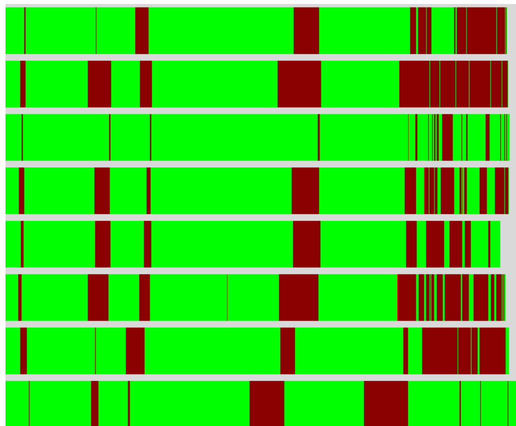



Scalability with 1 domain as reference for each curve




Speed-up with global time step for one domain as reference

# Aerodynamic solver : limits



- 8 nodes, OpenMP used inside each node.
- Current parallelization implies too much synchronization.

# Response to the limits

### Aerodynamic solver

- Synchronization problem : the current algorithm implies a lot of synchronizations in distributed memory while actual dependencies are local
- Working on subdomains in shared memory could lead to a more flexible way of overlapping computation and communication

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Outlines

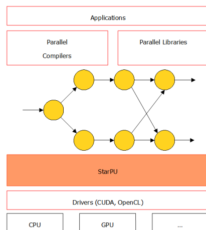

1. Introduction
2. Parallelization : OpenMP/MPI Approach
3. Parallelization : design of a task-based version
   - Parallelization using a Runtime (StarPU)


AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Runtime : Overview

## Layer between the application and the hardware




- Toward *portability of performance*.
- Exploit heterogeneous architectures.
- Exploit different paradigm : tasks, OpenMP, MPI
- Do dynamically what can not be done statically
  - Load balance
  - React to hardware feedback
  - Self-organization, autotuning

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Runtime : Overview

## Scheduling *tasks*

- Tasks are scheduled on processing units (CPU, GPU...), with respect to dependencies.
- Dynamic scheduling : tasks are scheduled at runtime.
- Scheduling strategy can be changed and adapted to the problem or the computer.

## Managing Data Transfers

- When computation must be done on a GPU, data must be moved. This is done transparently by the runtime.
- When combined with a scheduling strategy that take decision early, ability to *prefetch* data.

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Runtime : Data handle and Codelet

## Data handle

- Represents the data internally.
- Allows to keep coherency between memory nodes. (via MSI protocol)

## Codelet

- Contains implementation(s) of an operation for different PUs.
- Contains the way data are accessed by the operation. (READ, WRITE, READ-WRITE)

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Runtime : Tasks

## Task

- The application of a **codelet**...
- ... to different **data handles**...
- ... with respect to **dependencies**

## Dependencies

- Possibility to describe them explicitly.
- Compute them at runtime : sequential consistency. The order of task insertion defines the dependencies.
- Mix the two other options.

# Runtime : Implicit Dependencies example

**Sequential consistency**

Any data access will appear as sequentially consistent from the application.

Example :

# Runtime : Implicit Dependencies example

**Sequential consistency**

Any data access will appear as sequentially consistent from the application.

Example :

insert_task(codelet1,A(R))

# Runtime : Implicit Dependencies example

**Sequential consistency**

Any data access will appear as sequentially consistent from the application.

Example :

insert_task(codelet1,A(R))
insert_task(codelet2,A(R))

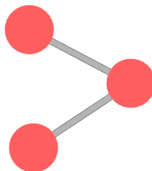
# Runtime : Implicit Dependencies example

**Sequential consistency**

Any data access will appear as sequentially consistent from the application.

Example :

```
insert_task(codelet1,A(R))
insert_task(codelet2,A(R))
insert_task(codelet3,A(RW))
```



AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

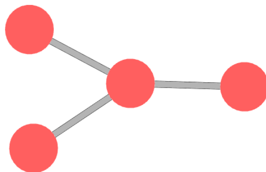
# Runtime : Implicit Dependencies example

**Sequential consistency**

Any data access will appear as sequentially consistent from the application.

Example :

```
insert_task(codelet1,A(R))
insert_task(codelet2,A(R))
insert_task(codelet3,A(RW))
insert_task(codelet2,A(R))
```



AIRBUS DEFENCE & SPACE

Inria

# Task version : implementation

## What kind of operations are performed ?

- Finite Volume is mainly about content of cells and flows at surface between cells
- Some "patterns" in the way the data are accessed
  - Computation on Faces
  - Computation on Cells
  - Computation on Faces using Cell data
  - Computation on Cells using Face data

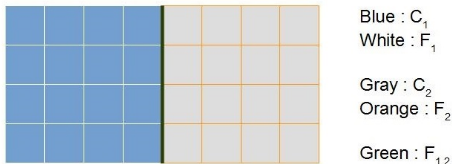
## Realisation of a task version

- Domain decomposition to generate tasks
- Rely on an abstraction (Computation Elements) to access data

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Task version : Computation Elements




## Ci / Fi / Fij

- Create Computation Elements (CE) with a cell component and a face one (Domain Decomposition)
- Retrieving handles through CE
- Topology is explicit (unstructured meshes), so it must be explicit between the computation elements too. Fij component between two computation elements

AIRBUS DEFENCE & SPACE

Inria

# Task version : Inserting the tasks

### Foreach

- Replacing SUBROUTINE CALLS by FOREACH_XX in order to submit tasks.
- Each kind of operation have its foreach (FOREACH_F, FOREACH_CI_FI, FOREACH_CI_CJ_FIJ...)
- Mandatory step of refactoring.

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Task version : FOREACH

```
      IHANDLE(1:30)=[F_WEN, F_XFACE,
&                         F_DROG,F_DRPG,
&                         F_DRXG,F_DRGG,F_DRRG,F_DRSG,F_DREG,
&                         F_DROD,F_DRPD,
&                         F_DRXD,F_DRGD,F_DRRD,F_DRSD,F_DRED,
&                         F_DRUD,F_DRVD,F_DRWD,
&                         F_DRUG,F_DRVG,F_DRWG,
&                         F_VDMS,   F_VDMUXS, F_VDMUYS, F_VDMUZS,
&                         F_VDMXNS, F_VDMENS, F_VDMQRS, F_VDMQSS]

      CALL CE_FOREACH_CI_FI(OPAQ_CE, NUM_CE,
&                              C_LOC(PARAM), OPAQ_RUNTIME,
&                              CO_RIEMAN,
&                              BO_F, 0, C_LOC(IPARAM),
&                              0,  C_LOC(IHANDLE), ! CELLS
&                              30, C_LOC(IHANDLE)) ! FACES

      CALL CE_FOREACH_CI_CJ_FIJ(OPAQ_CE, NUM_CE,
&                                     C_LOC(PARAM), OPAQ_RUNTIME,
&                                     CO_RIEMAN,
&                                     0, 0, C_LOC(IPARAM),
&                                     0,  C_LOC(IHANDLE), ! CELLS
&                                     30, C_LOC(IHANDLE)) ! FACES
```

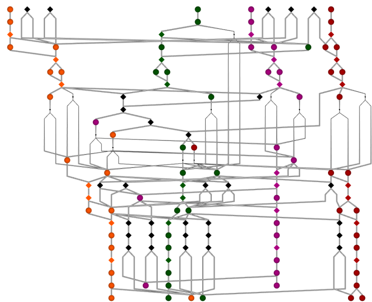
# Task version : FOREACH pseudo-code

```
for (int i=0;i<num_CE;i++)
{
    //retrieve codelet
    current_codelet=global_opaque.codelet[codelet];

    if (CE not concerned by current computation)
        continue;

    //retrieve handles
    for (int k=0;k<num_cells;k++)
        add_parameter(param_array,ce[i].cell[param_cell[k]]);

    for (int k=0;k<num_faces;k++)
        add_parameter(param_array,ce[i].faces[param_faces[k]]);

    //Task insertion
    insert_task(codelet, param_array)
}
```

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Task version : DAG generated by foreach



## DAG with 4 CE

- The colors represent different CE.
- Diamonds are for operations that write on Faces.
- Circles are for operations that write on Cells.

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

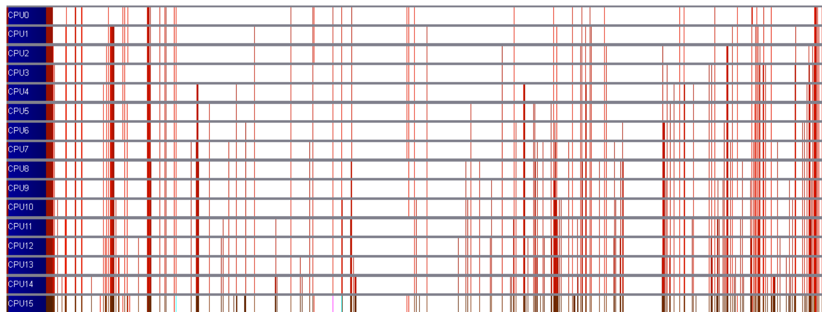
# Task version : Importance of parallel tasks

Without parallel tasks and a few number of CE...

AIRBUS DEFENCE & SPACE

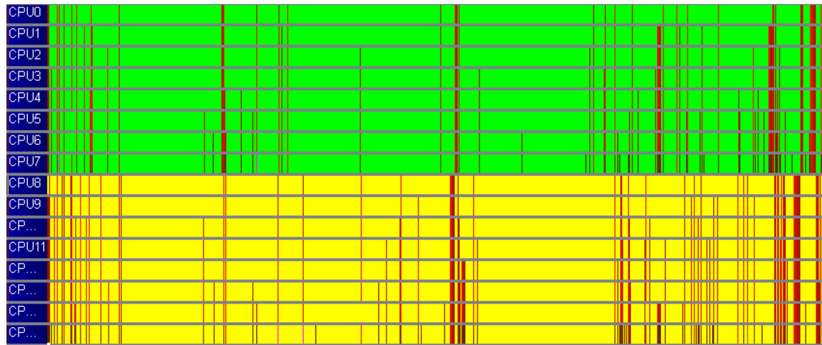
Inria

# Task version : Importance of parallel tasks




Same number of CE and just one parallel worker

# Task version : Importance of parallel tasks




Number of workers can be adjusted... (here one per socket)

# Conclusion

## Task parallelisation over runtime

- Mandatory to think about the way to creates the tasks
- Allow to try different scheduling strategy and use of the computer
- Importance of limiting runtime overhead
  - Packing task is mandatory
- Achieving performance is not easy but once it is done, task parallelisation opens new path.

## Perspectives

- Task parallelisation of the Intersection operation (co-schedule intersection and aerodynamic)
- Pipelining of iterations...

AIRBUS DEFENCE & SPACE

Still a lot of work to do!

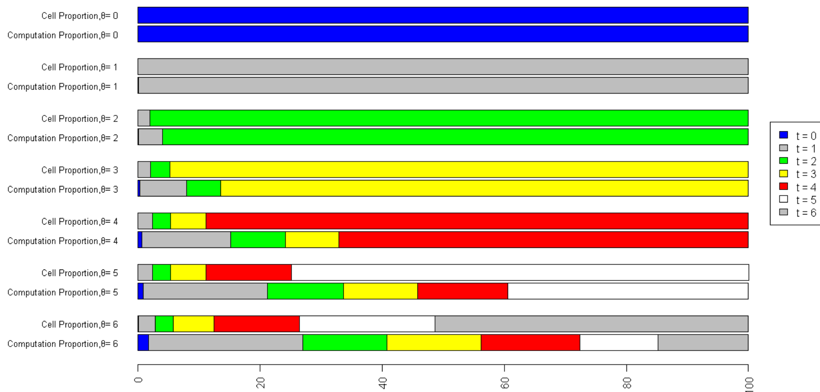
Inria INVENTEURS DU MONDE NUMÉRIQUE

Thank you for your attention.
Any questions?

AIRBUS DEFENCE & SPACE

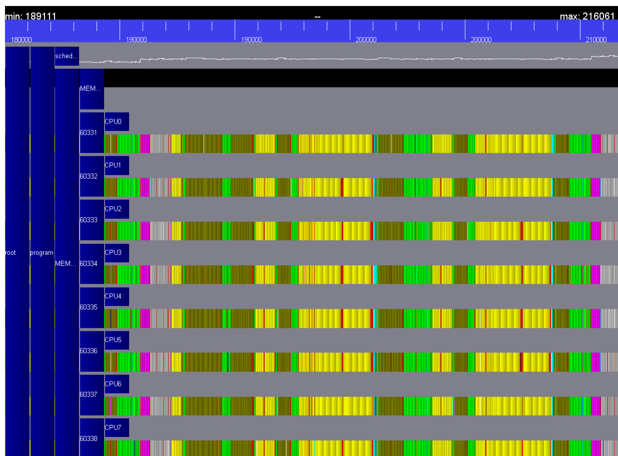
Inria INVENTEURS DU MONDE NUMÉRIQUE

# Perf Analysis




Proportion of cells of each temporal level and the associated computation proportion with different maximum temporal level

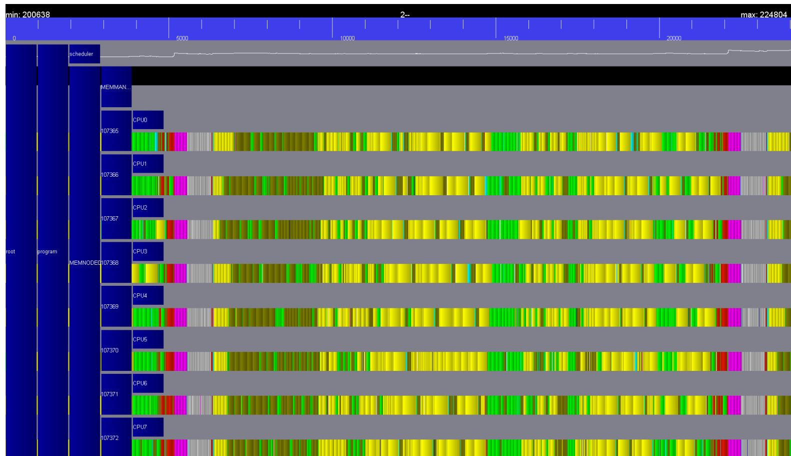
# Trace 'OMP PARALLEL DO'-like




AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

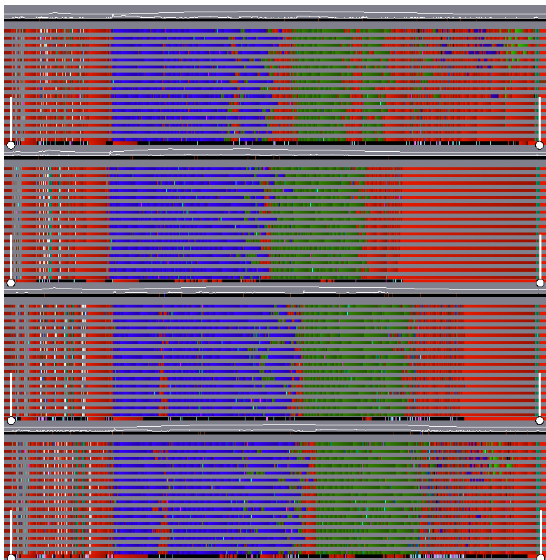
# Trace using tasks spawned by foreach




AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

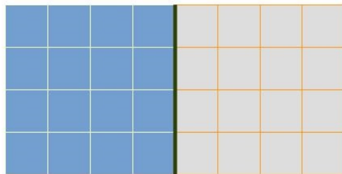
# MPI Trace



AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Task version : Distinguishing inner and border of Ci



Blue : $C_1$
White : $F_1$

Gray : $C_2$
Orange : $F_2$

Green : $F_{1,2}$

GRADIE_B1 :
GRADIE_B1.1
- In : Intens[$C_i$]
- Out : Drxy[$F_i$]

GRADIE_B1.2
- In : Intens[$C_i$] , Intens[$C_j$]
- Out : Drxy[$F_{ij}$]

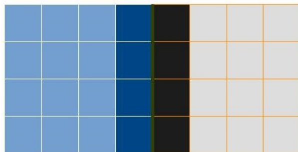
GRADIE_B3 :
GRADIE_B3.1
- In : Gradient[$C_i$],Drxy[$F_i$]
- Out : Gradient[$C_i$]

GRADIE_B3.2
- In : Gradient[$C_i$] , Gradient[$C_j$], Drxy[$F_{ij}$]
- Out : Gradient[$C_i$] , Gradient[$C_j$]

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Task version : Distinguishing inner and border of Ci



Blue : $C^{int}_1$
Dark Blue : $C^{bord}_1$
White : $F_1$

Gray : $C^{int}_2$
Dark Gray : $C^{bord}_2$
Orange : $F_2$

Green : $F_{1,2}$

GRADIE_B1 :
GRADIE_B1.1.1
In : Intens[$C^{Bord}_i$]
Out : Drxy[$F_i$]
GRADIE_B1.1.2
In : Intens[$C^{int}_i$]
Out : Drxy[$F_i$]
GRADIE_B1.2
In : Intens[$C^{Bord}_i$] , Intens[$C^{Bord}_j$]
Out : Drxy[$F_{ij}$]

GRADIE_B3 :
GRADIE_B3.1.1
In : Gradient[$C^{bord}_i$],Drxy[$F_i$]
Out : Gradient[$C^{bord}_i$]
GRADIE_B3.1.2
In : Gradient[$C^{int}_i$],Drxy[$F_i$]
Out : Gradient[$C^{int}_i$]
GRADIE_B3.2
In : Gradient[$C^{bord}_i$] , Gradient[$C^{bord}_j$], Drxy[$F_{ij}$]
Out : Gradient[$C^{bord}_i$] , Gradient[$C^{bord}_j$]

AIRBUS DEFENCE & SPACE

Inria INVENTEURS DU MONDE NUMÉRIQUE

# Task version : Distinguishing inner and border of Ci

Gradie B1 : $C_i$ / $C_j$ / $F_{ij}$

B 1.1 (E1)

B 1.1 (E2)

B 1.2 (E1-E2)

Gradie B1 : $C^{bord}_i$ / $C^{bord}_j$ / $C^{int}_i$ / $C^{int}_j$ / $F_{ij}$

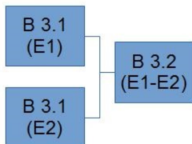
B 1.1.1 (E1) — B 1.1.2 (E1)

B 1.1.1 (E2) — B 1.1.2 (E2)

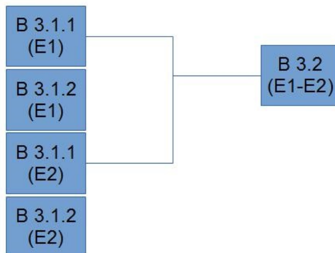
B 1.2 (E1-E2)

# Task version : Distinguishing inner and border of Ci

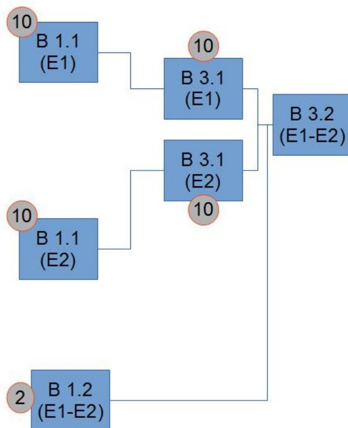
Gradie B3 : $C_i$ / $C_j$ / $F_{ij}$




Gradie B3 : $C^{bord}_i$ / $C^{bord}_j$ / $C^{int}_i$ / $C^{int}_j$ / $F_{ij}$

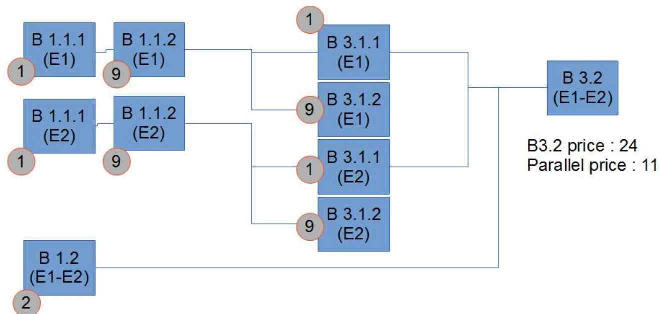


# Task version : Distinguishing inner and border of Ci




B3.2 price : 42
Parallel price : 20

# Task version : Distinguishing inner and border of Ci




GRADIE_B1 :
GRADIE_B1.1.1
  In : Intens[$C^{Bord}_i$]
  Out : Drxy[$F_i$]
GRADIE_B1.1.2
  In : Intens[$C^{int}_i$]
  Out : Drxy[$F_i$]
GRADIE_B1.2
  In : Intens[$C^{Bord}_i$] , Intens[$C^{Bord}_j$]
  Out : Drxy[$F_{ij}$]

GRADIE_B3 :
GRADIE_B3.1.1
  In : Gradient[$C^{bord}_i$],Drxy[$F_i$]
  Out : Gradient[$C^{bord}_i$]
GRADIE_B3.1.2
  In : Gradient[$C^{int}_i$],Drxy[$F_i$]
  Out : Gradient[$C^{int}_i$]
GRADIE_B3.2
  In : Gradient[$C^{bord}_i$] , Gradient[$C^{bord}_j$], Drxy[$F_{ij}$]
  Out : Gradient[$C^{bord}_i$] , Gradient[$C^{bord}_j$]

AIRBUS DEFENCE & SF

Inria INVENTEURS DU MONDE NUMÉRIQUE