



# **Compilation d'applications de traitement d'images sur architecture MPPA-Manycore**

Pierre Guillou

## **► To cite this version:**

Pierre Guillou. Compilation d'applications de traitement d'images sur architecture MPPA-Manycore. Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS 2014), Apr 2014, Neuchatel, Suisse. <hal-01096993>

**HAL Id: hal-01096993**

**<https://minesparis-psl.hal.science/hal-01096993v1>**

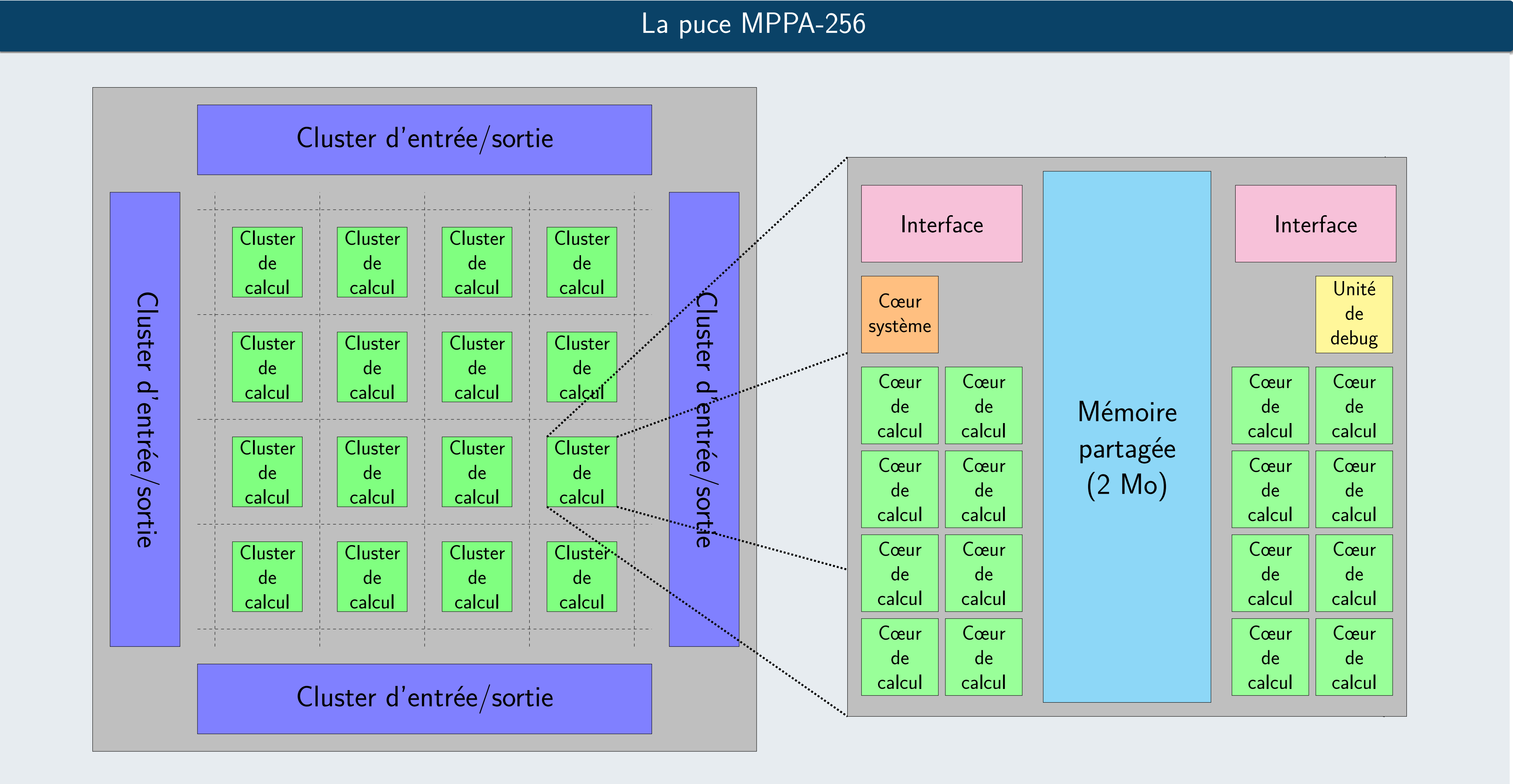
Submitted on 18 Dec 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization



Analyse d'images

analyse d'images : reconnaissance des éléments contenus dans une image par utilisation de la...

morphologie mathématique : fondée sur l'algèbre des treillis


FREIA : API pour l'analyse d'images (MINES ParisTech, Télécom Bretagne, Thalès)

Opérateurs d'analyse d'images accessible via FREIA :


- opérateurs arithmétiques
  - pixel + paramètre, une image en entrée
  - pixel + pixel, deux images en entrée
  - + - × ÷ min max = & | ~
- opérateurs morphologiques
  - calculs au voisinage
  - kernel matrice 3 × 3
  - érosion, dilatation, convolution
- opérateurs de réduction
  - maximum, minimum global +/- coordonnées
  - volume : somme des pixels
- autres opérateurs
  - seuil, masque, log2,...

⇒ bibliothèque d'agents *Sigma-C*

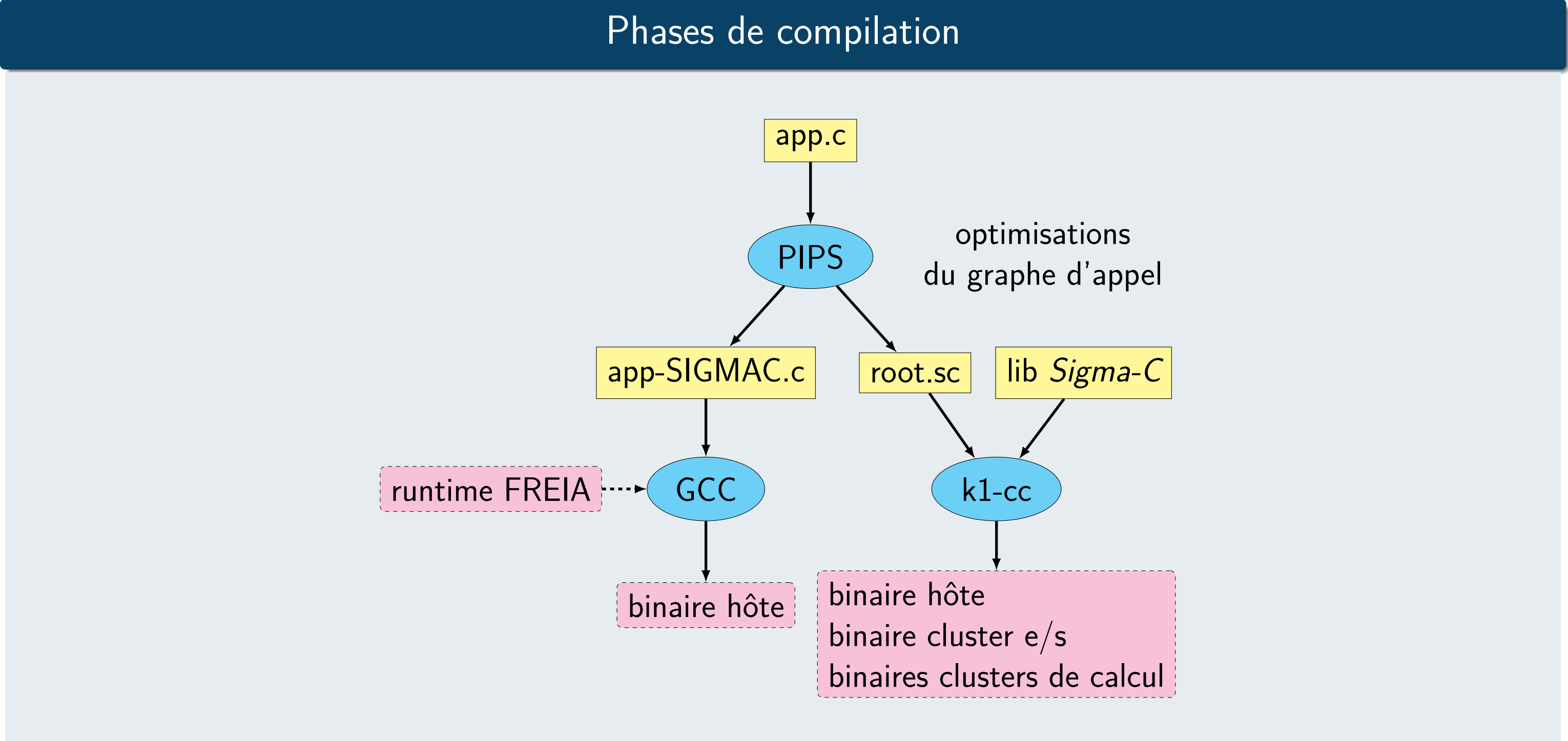
Exemple : extraction de plaques d'immatriculation



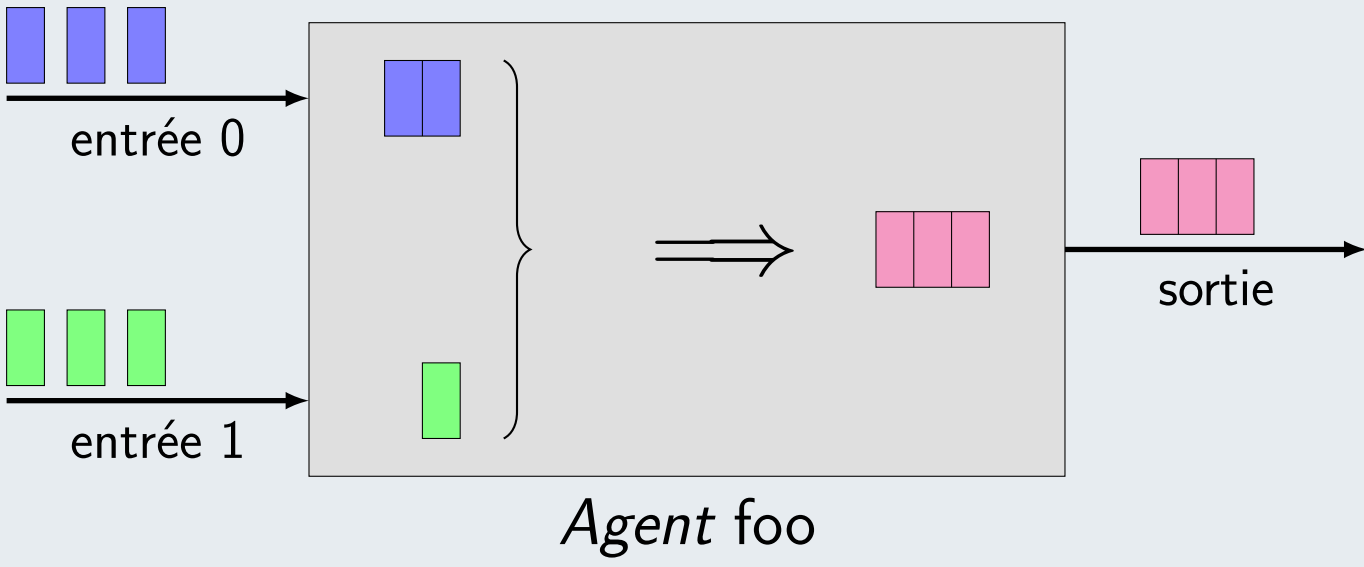
(a) Image d'entrée



(b) Image de sortie



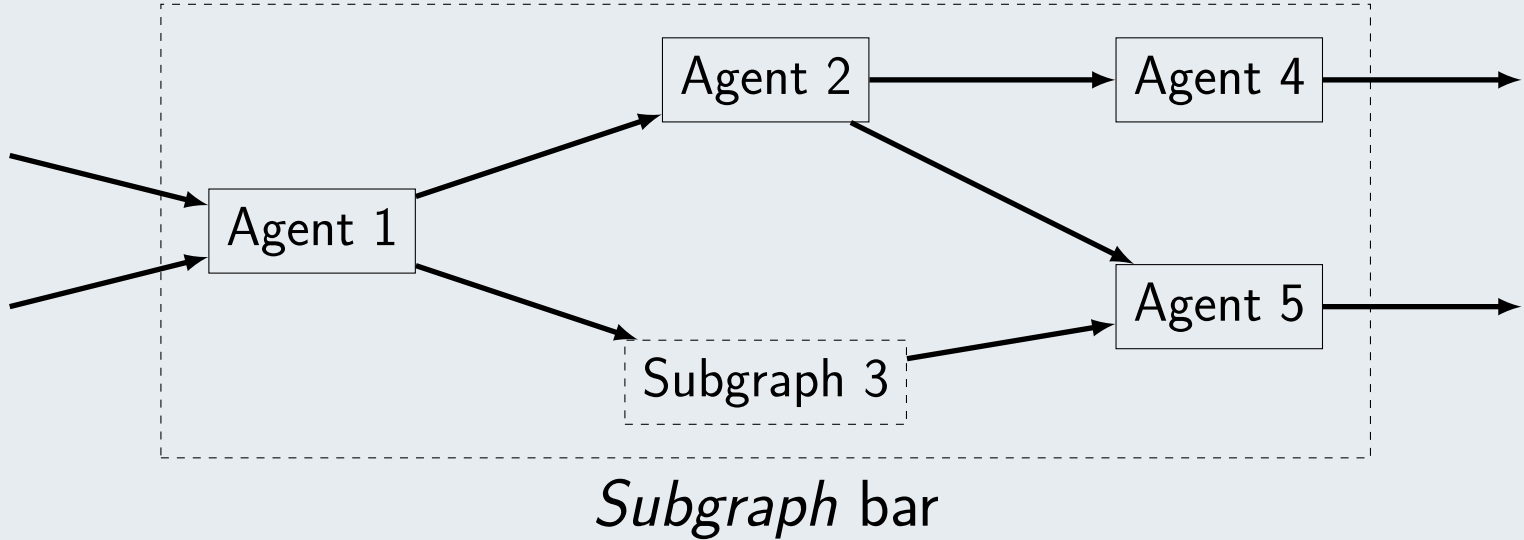
Sigma-C, un langage *dataflow* (Kalray – CEA-LIST)



```

agent foo() {
  // describe agent interface
  interface {
    in<int> input0;
    in<int> input1;
    out<int> output;
  }
  // declare the state machine
  spec{input0[2], input1, output[3]};
}
// loop over the state
void start() exchange (input0 inp0[2],
                      input1 inp1,
                      output outp[3]) {
  outp[0] = inp0[0];
  outp[1] = inp1;
  outp[2] = inp0[1];
}

```



```

subgraph bar() {
  // describe subgraph interface
  interface { ... }
  map {
    // instantiate agents
    agent a1 = new Agent1();
    ...
    agent a3 = new Subgraph3();
    // connect agents to subgraph interfaces
    connect (input0, a1.input0);
    ...
    connect (a5.output, output1);
    // connect agents
    connect (a1.output0, a2.input);
    ...
    connect (a3.output, a5.input1);
  }
}

```

