Introduction à l'utilisation de MPI dans Fluent

Jean-Baptiste Keck

Laboratoire Jean Kuntzmann - Grenoble

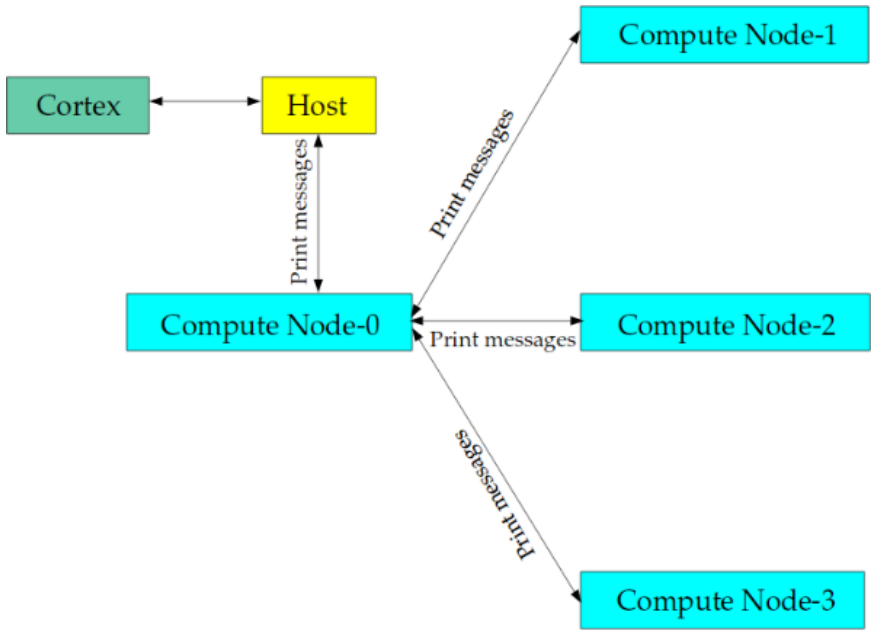
1er février 2017

Table des matières

Organisation des processus

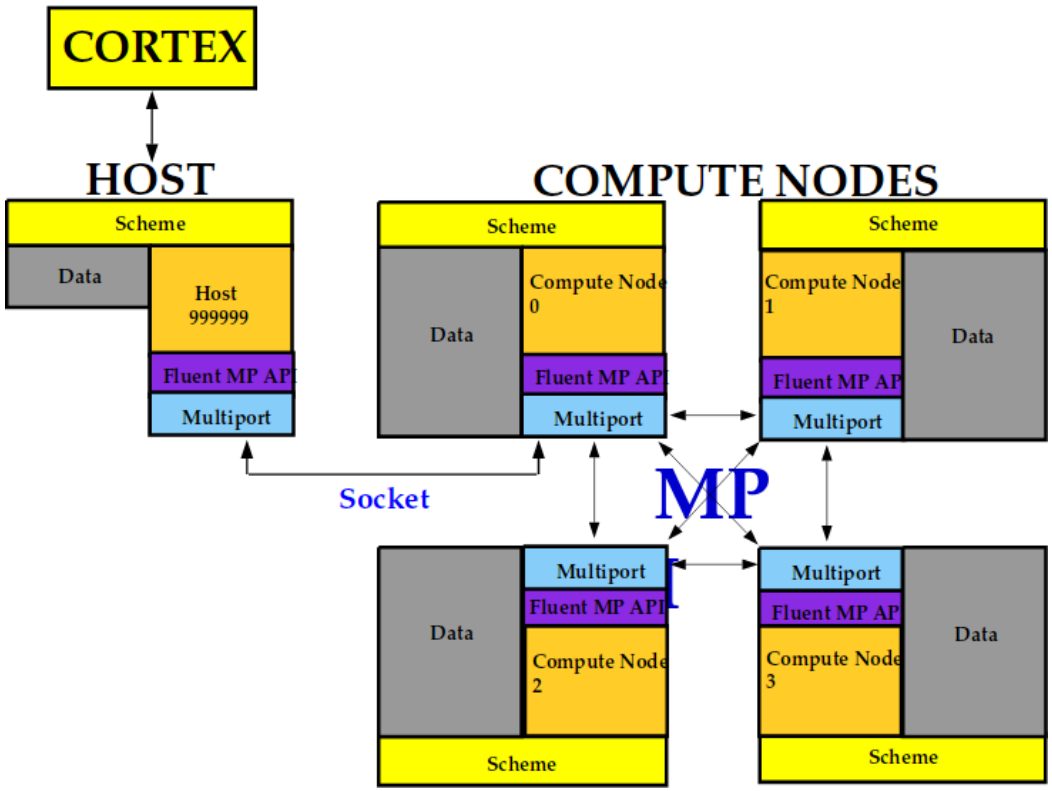
Pour N noeuds de calcul, Fluent execute $N + 2$ processus :

- Cortex : Processus responsable de l'interface utilisateur.
- Host : Reçoit et interprète les commandes de Cortex puis les passes au noeud de calcul 0.
- Compute nodes : N noeuds de calculs, numérotés de 0 à $N - 1$. Tout ces noeuds reçoivent les commandes par le biais du noeud 0.



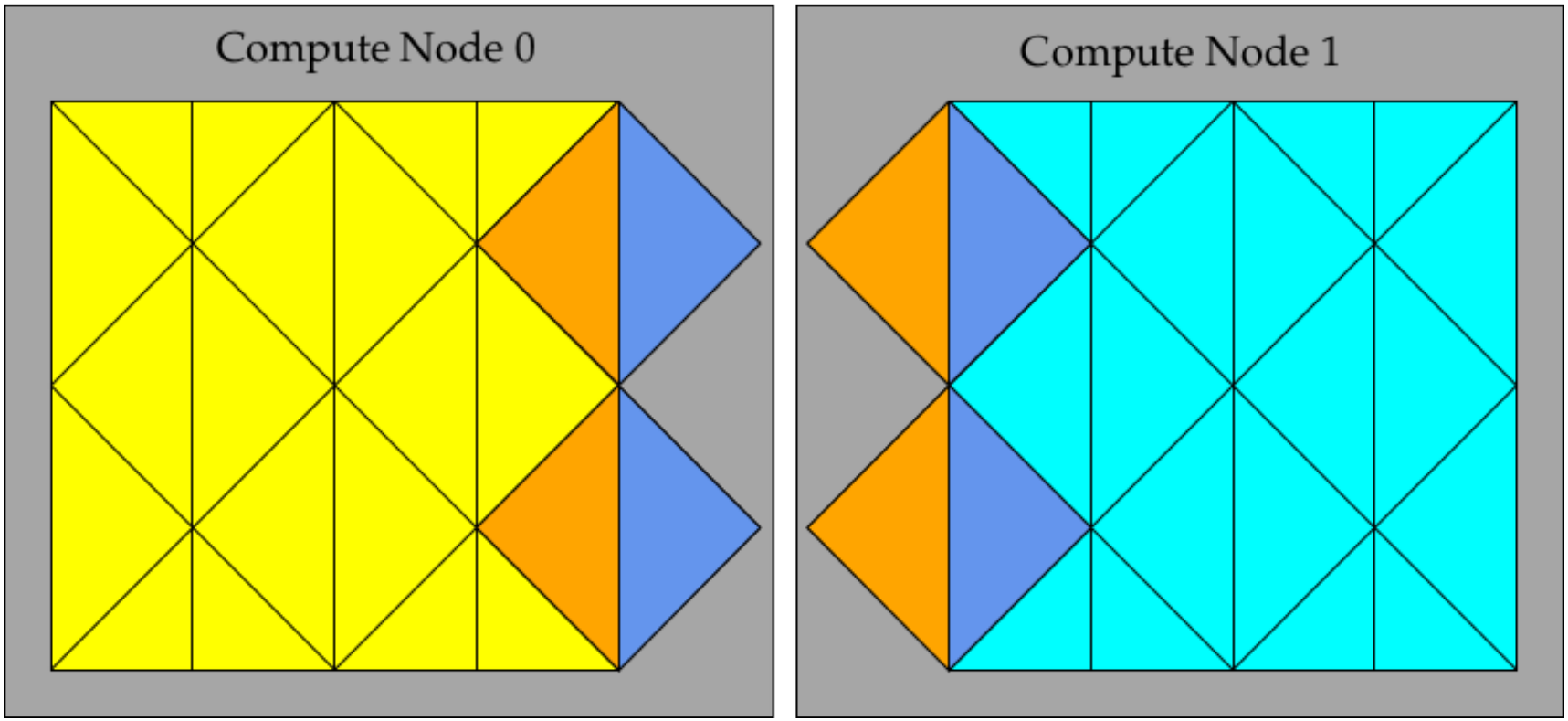
Que paralléliser dans un UDF

- Entrées/sorties : print, lecture et écriture de fichiers...
- Reductions : operations globales (somme,min,max,...)
- Logique : récupérer des états globaux (capteurs,...)
- Algorithmes : itération sur les maillages.



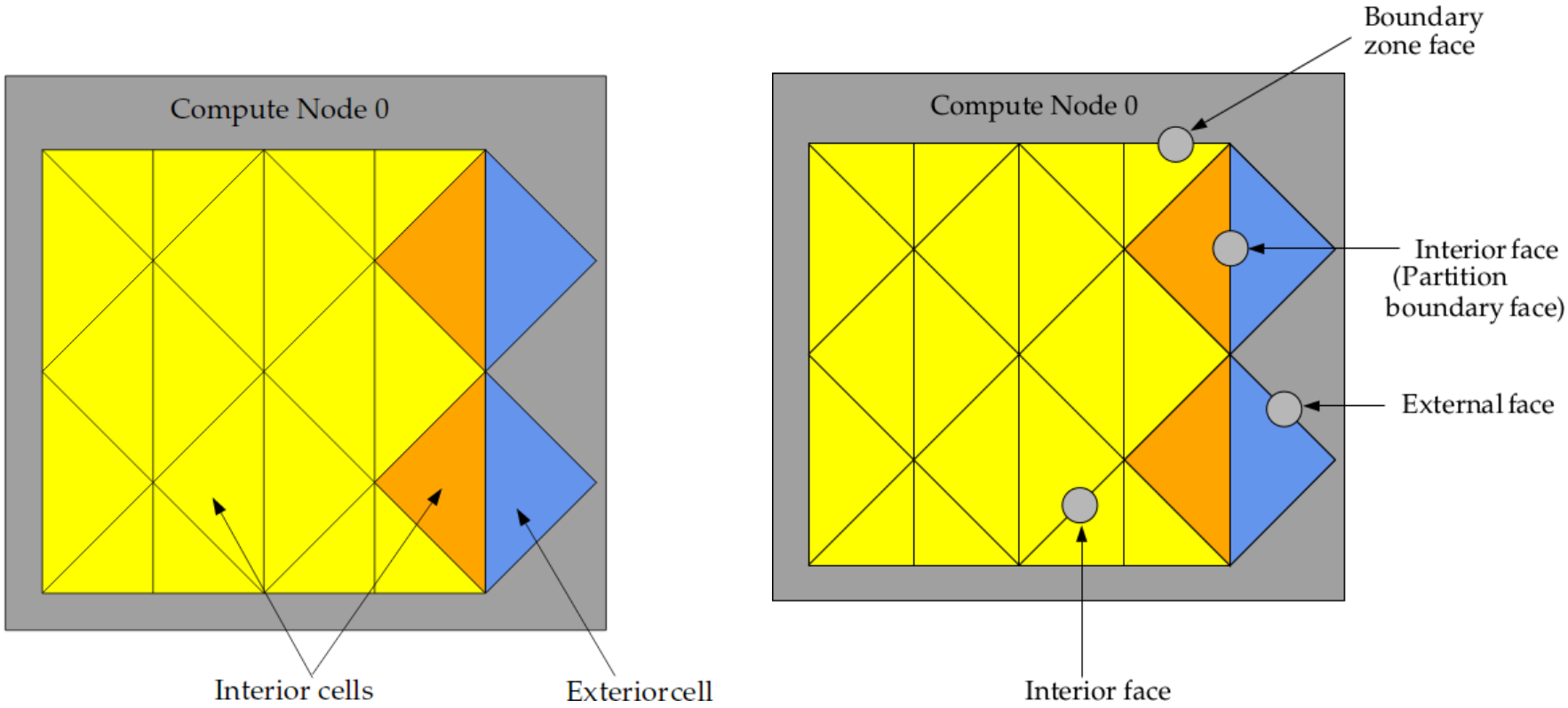
Partition du problème

Fluent découpe le maillage en plusieurs partitions et les affecte chacune à un noeud de calcul. Chaque noeud de calcul exécute ensuite le même algorithme sur ses propres données. L'hôte ne contient pas de maillage.



Partition du maillage

Les données d'un noeud de calcul incluent des recouvrements sur les maillages voisins (ghosts).



1. Introduction
   - Organisation des processus
   - Que paralléliser dans les UDF
   - Partition du problème

2. Paralléliser les UDF
   - Différencier les processus
   - Communiquer entre l'hôte et les noeuds de calculs
   - Communiquer entre le noeud de calcul 0 et l'hôte
   - Communication noeud à noeud
   - Itérer sur les maillages
   - Entrées-Sorties

Différencier les processus

L'UDF est exécuté par tous les processus mentionnés précédemment.
Fluent fournit des macros préprocesseur pour différencier les processus :

- `RP_NODE` : Noeud de calcul
- `RP_HOST` : Host
- `PARALLEL` : Host ou noeud de calcul

Toutes les macros peuvent être trouvés dans le header `para.h`.

```
1  #if RP_HOST
2  /* only host process is involved */
3  #endif
4
5  #if RP_NODE
6  /* only compute nodes are involved */
7  #endif
8
9  #if PARALLEL
10 /* both host and compute nodes are involved */
11 #endif
```

Variables globales prédéfinies

Quelques variables globales sont déjà prédéfinies :

```
1  int node_zero   = 0;
2  int node_one    = 1;
3
4  int node_host   = 999999;
5  int node_serial = 1000000;
6
7  /* id of the last compute node */
8  int node_last;
9
10 /* number of compute nodes */
11 int compute_node_count;
12
13 /* id of the current node */
14 int myid;
```

Prédicats prédéfinis

Quelques prédicats sont également déjà prédéfinis :

```
1  /* Predicate definitions from para.h header file */
2  #define I_AM_NODE_HOST_P (myid == node_host)
3  #define I_AM_NODE_ZERO_P (myid == node_zero)
4  #define I_AM_NODE_ONE_P (myid == node_one)
5  #define I_AM_NODE_LAST_P (myid == node_last)
6
7  #define I_AM_NODE_SAME_P(n) (myid == (n))
8  #define I_AM_NODE_LESS_P(n) (myid < (n))
9  #define I_AM_NODE_MORE_P(n) (myid > (n))
10
11 #define MULTIPLE_COMPUTE_NODE_P (compute_node_count > 1)
12 #define ONE_COMPUTE_NODE_P (compute_node_count == 1)
13 #define ZERO_COMPUTE_NODE_P (compute_node_count == 0)
```

Communiquer entre l'hôte et les noeuds de calculs

Host to node data transfer :

- `host_to_node_[TYPE]_[NUM]` : Transfère `NUM` (max 7) variables de type `TYPE` de l'hôte vers tous les noeuds de calculs (en passant par le noeud 0).

```
1 /* integer and real variables passed from
2  * host to nodes */
3 host_to_node_int_1(count);
4 host_to_node_real_5(r0,r1,r2,r3,r4);
```

⇒ Pas besoin de protéger l'appel de cette macro :

1. L'hôte envoie les données au noeud de calcul 0.
2. Les noeud 0 récupère les données et les `scatter` sur les autres.
3. Les autres processus ne font rien.

Communiquer entre l'hôte et les noeuds de calculs

Host to node data transfer :

- `host_to_node_[TYPE](ptr,size)` : Transfère un tableau de `TYPE` de l'hôte vers tous les noeuds de calculs (en passant par le noeud 0).

```
1  /* array variables passed from host to nodes */
2  char name[] = "test";
3  int ids[8] = {1,29,5,32,18,2,55,21};
4
5  host_to_node_string(name,6);
6  host_to_node_int(ids,8);
```

Attention : Pour les chaines de caractères, penser à prendre en compte le caractère de terminaison (`NUL`).

Communiquer entre le noeud 0 et l'hôte

Node to host data transfer :

- `node_to_host_[TYPE]_[NUM](args)` : Transfère `NUM` (max 7) variables de type `TYPE` du noeud 0 vers l'hôte.
- `node_to_host_[TYPE](ptr,size)` : Transfère un tableau de `TYPE` du noeud 0 vers l'hôte.

$\Rightarrow$ Pas besoin de protéger l'appel de cette macro :

1. Le noeud de calcul 0 envoie les données à l'hôte.
2. L'hôte récupère les données.
3. Les autres processus ne font rien (aucun code n'est généré sur les noeuds de calculs $\neq 0$).

Communication noeud à noeud

Node to node data transfer :

- `PRF_CSEND_[TYPE](dst_id, buffer, nelem, tag)` : Transfère un tableau de `nelem` variables de type `TYPE` vers le noeud de calcul `dst_id`. La communication est identifiée par l'entier `tag`.
- `PRF_CRECV_[TYPE](dst_rank, buffer, nelem, tag)` : Equivalent en réception de la commande précédente.

⇒ `[TYPE]` peut être `CHAR`, `INT`, `BOOLEAN`, `REAL`, `FLOAT`, `DOUBLE`

Attention : Il faut toujours faire correspondre un `SEND` et un `RECV` car ce sont des appels bloquants.

Reductions globales

Réductions entre noeuds de calcul

- `PRF_G[TYPE][OP]1(val)` : Effectue une réduction (avec l'opération `OP`) sur des scalaires de type `TYPE` entre tous les noeuds de calculs. Le résultat est retourné de la macro.
- `PRF_G[TYPE][OP](ptr,size,work)` : Effectue une réduction (avec l'opération `OP`) sur des vecteurs de type `TYPE` entre tous les noeuds de calculs. Un tableau temporaire `work` doit être préalloué. Le résultat est stocké dans le tableau initial `ptr`.

⇒ `[TYPE]` peut être `I` (`int`), `R` (`real`) ou `L` (`bool`).

⇒ `[OP]` peut être `SUM` (somme)
`HIGH` (max), `LOW` (min)
`OR` (ou logique), `AND` (et logique)

Échange de variables entre noeuds de calcul

Échange de champs (storage variables) :

- `EXCHANGE_SVAR_MESSAGE(domain, [SVARS])` : Échange les `cell data` régulières entre les noeuds.
- `EXCHANGE_SVAR_MESSAGE_EXT(domain, [SVARS])` : Échange les `cell data` régulières et étendues entre les noeuds.
- `EXCHANGE_SVAR_FACE_MESSAGE(domain, [SVARS])` : Échange les `face data` entre les noeuds.

⇒ `[SVARS]` doit être sous la forme `(SV_F0, SV_F1, ..., SV_NULL)`

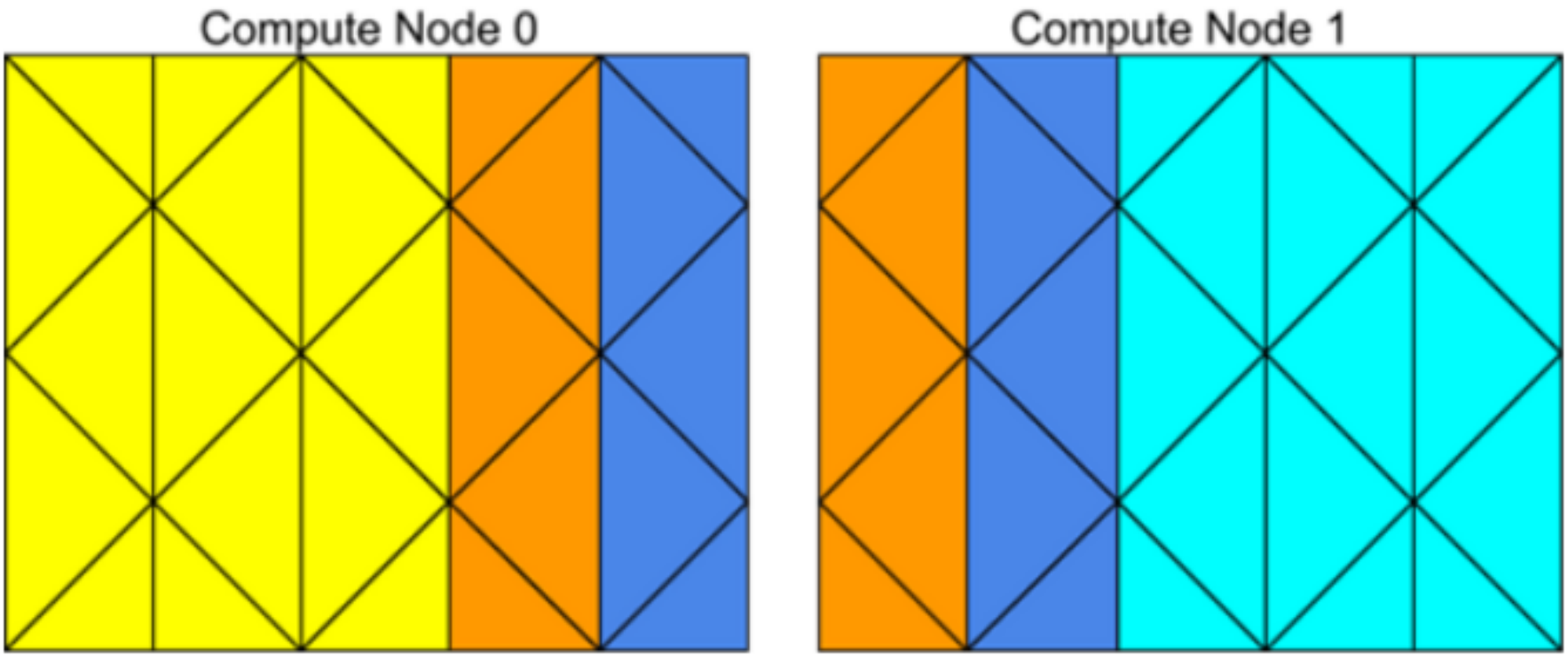
Les noms des des variables peuvent être obtenus dans les headers qui les définissent.

```
1 /* Echange la pression et la temprature entre les noeuds */
2 EXCHANGE_SVAR_MESSAGE(domain, (SV_P, SV_T, SV_NULL));
```

Itérer sur les maillages

Il existe différentes macros pour boucler sur les maillages (`face` et `cell`).

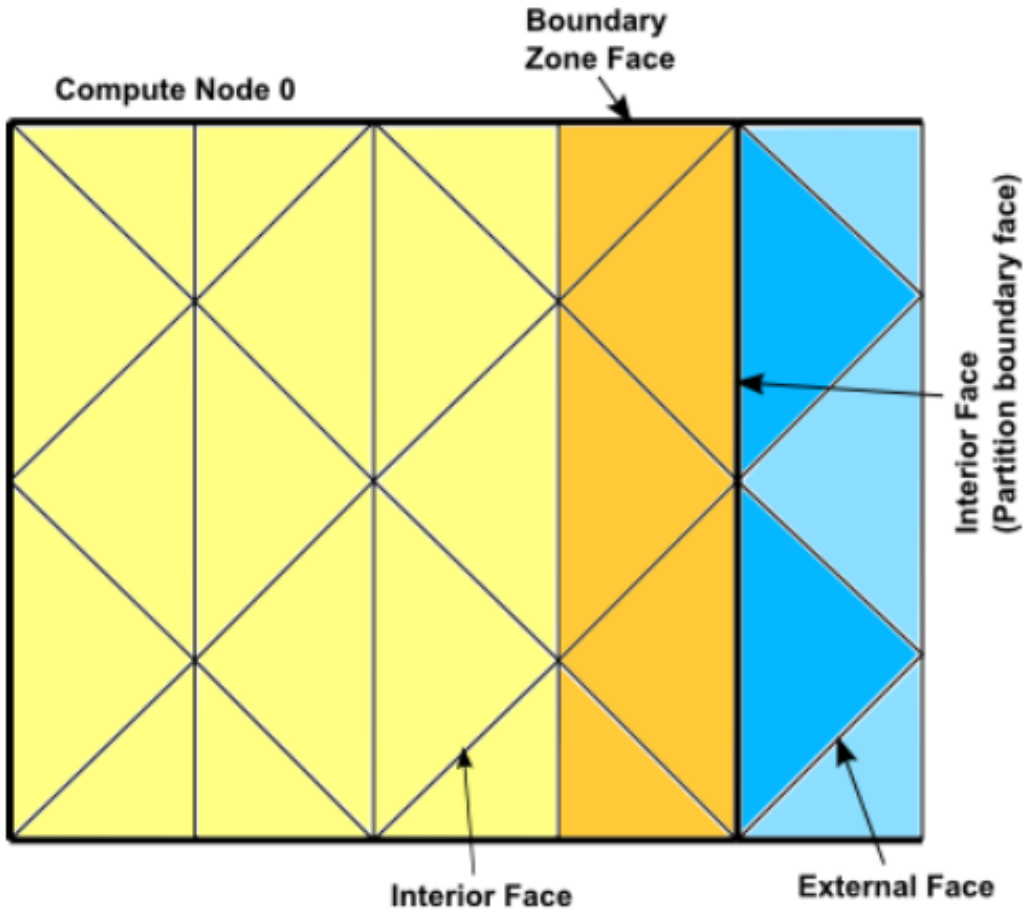
Partition des maillages :



Itérer sur les maillages

Il existe différentes macros pour boucler sur les maillages (`face` et `cell`).

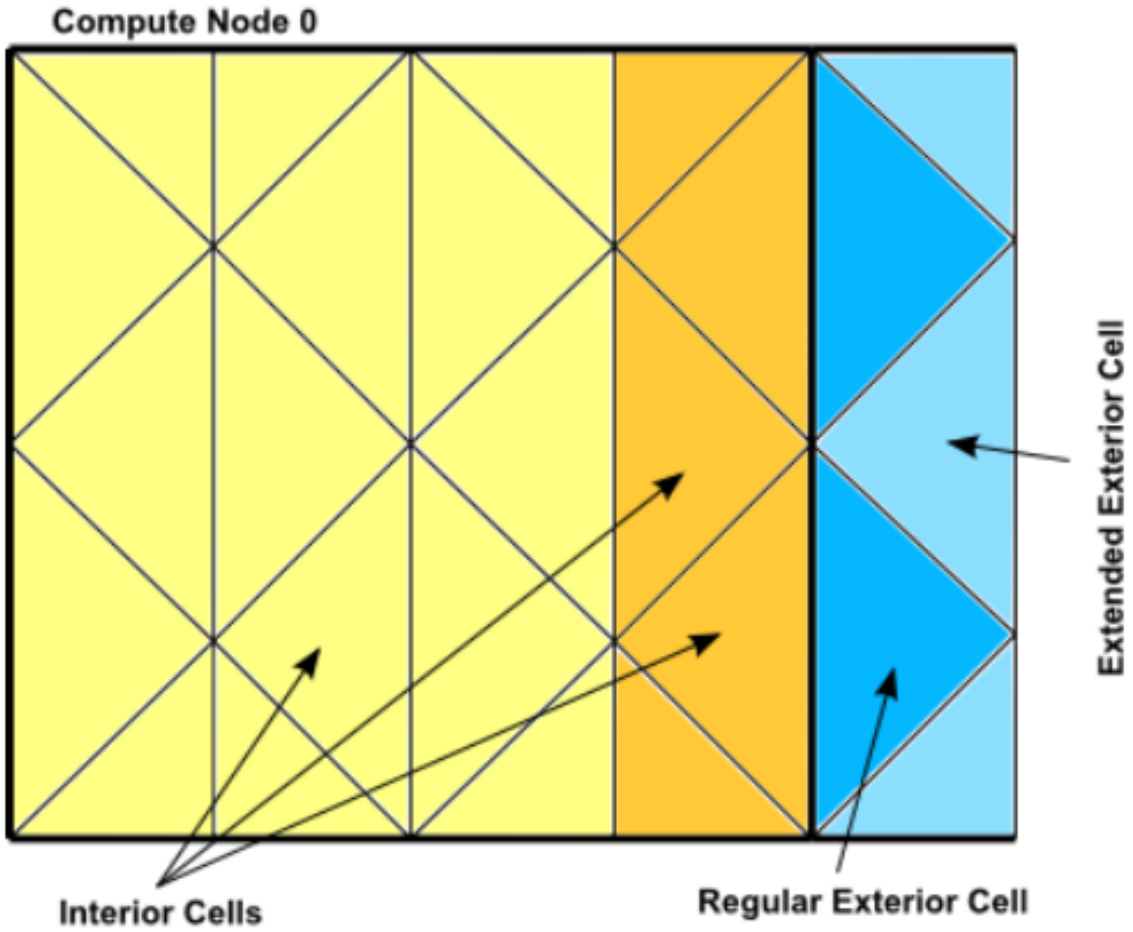
Dénomination des `faces` :



Itérer sur les maillages

Il existe différentes macros pour boucler sur les maillages (`face` et `cell`).

Dénomination des `cells` :



Entrées-Sorties

Messages

- `Message0(msg)` : Specialisation de la fonction `Message` sur le noeud de calcul 0.

```
1  /* Let Compute Node-0 display messages */
2  #if I_AM_NODE_ZERO_P
3      Message("Total Area: %f\n",total_area);
4  #endif
5
6  /* or simply */
7  Message0("Total Area: %f\n",total_area);
```

Lecture de fichiers

- `host_to_node_sync_file(file_path)` : Copie le fichier `file_path` de l'hôte vers les noeuds de calcul. La macro retourne le nombre d'octets lus.

```
1  DEFINE_ON_DEMAND(host_to_node_sync) {
2  #if PARALLEL
3  #if RP_HOST
4      int bytes = host_to_node_sync_file("d:\\test.dat");
5  #else
6      int bytes = host_to_node_sync_file("/tmp");
7  #endif /* RP_HOST */
8      printf ("Total number of bytes copied is %d\n", bytes);
9  #endif /* PARALLEL */
10 }
```

Écriture de fichiers

```
1  const int local_size = 10;
2  const int total_size = local_size * compute_node_count;
3  double* all_data = (real*)malloc( total_size * sizeof( real ));
4
5  #if PARALLEL
6  #if RP_NODE
7      double* mydata = (double*)malloc(local_size*sizeof( real ));
8      /* each compute node fills its own data */
9
10     if (I_AM_NODE_ZERO_P) {
11         /* include my own data */
12         memcpy(all_data,mydata,local_size*sizeof( real ));
13         /* receive data from all other compute nodes */
14         compute_node_loop_not_zero(src_id) {
15             PRF_CRCV_REAL(src_id, all_data+src_id*local_size, local_size, src_id);
16         }
17         /* send the whole array to host */
18         node_to_host_real(all_data, total_size );
19     }
20     else {
21         /* other compute nodes send their local data to compute node 0 */
22         PRF_CSEND_REAL(myid, mydata, local_size, myid);
23     }
24 #else
25     /* host receives all data from node 0 and writes it to file */
26     node_to_host_real(all_data, total_size );
27     FILE *f = fopen("test.dat","w");
28     for(int i=0; i<total_size; i++) { fprintf("%f\n", all_data[i]); }
29     fclose (f);
30 #endif /* RP_NODE */
31 #endif /* PARALLEL */
```

Divers

- `PRF_GSYNC()` : Synchronisation globale entre les noeuds de calcul. L'exécution continue lorsque tout le noeuds de calcul ont atteint cet appel.
- Paralléliser des UDFs DPM (Discrete Phase Model) : Des précautions particulières sont a prendre pour la lecture et l'écriture de fichiers.