www.idris.fr

Institut du développement et des ressources en informatique scientifique

JDEV 2017

Panorama des paradigmes de parallélisation pour les architectures pré-exascales

P.-Fr. Lavallée – 4-7 juillet 2017

TOP500 – Parallélisme massif

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-269? 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
5	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890
6	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939
7	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu Joint Center for Advanced High Performance Computing Japan	556,104	13,554.6	24,913.5	2,719
8	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu RIKEN Advanced Institute for Computational Science (AICS) Japan	705,024	10,510.0	11,280.4	12,660
9	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom , IBM DOE/SC/Argonne National Laboratory United States	786,432	8,586.6	10,066.3	3,945
10	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	301,056	8,100.9	11,078.9	4,233

Architectures pré-exascales

Quatre types d'architectures pré-exascales sont annoncées :

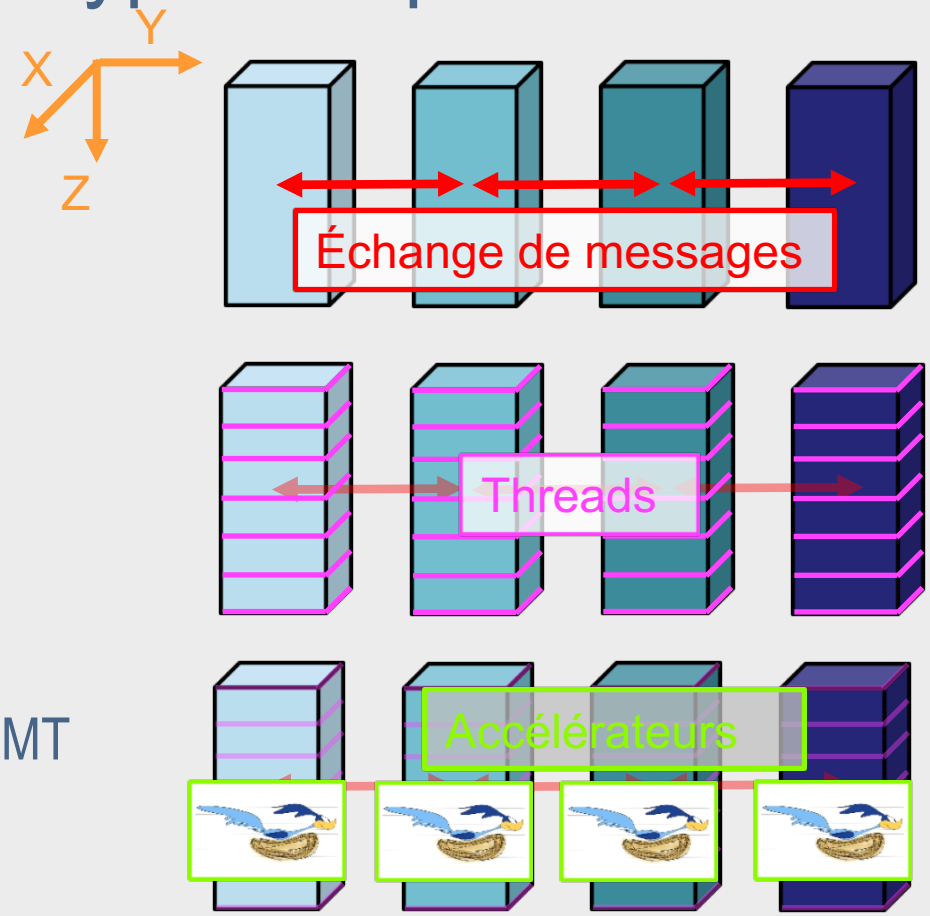
- Homogène généraliste x86 (Xeon Skylake, …) – Pangea (TOTAL)
- Homogène ARM (Fujitsu ARMv8 + SVE) – Post-K (RIKEN)
- Homogène many-coeurs (Xeon KNL, KNH) – Frioul (CINES) – AURORA (Argonne, http://aurora.alcf.anl.gov/)
- Hétérogène accélérée (OpenPOWER, x86+GPU, ...) – Ouessant (IDRIS) – Summit (OAK Ridge, https://www.olcf.ornl.gov/summit/)

Ouverture élargie des deux prototypes (Frioul et Ouessant) de la cellule de veille technologique de GENCI aux porteurs de projet de la communauté scientifique nationale depuis depuis le 1er mai.

- Informations complémentaires pour l'accès à ces machines : https://www.edari.fr/
- Informations techniques : http://www.idris.fr/ouessant/prototype.html, https://www.cines.fr/un-nouveau-prototype-pour-le-calcul/

Typologie des différents types de parallélisme

- ***Domain Parallelism***
  - Parallélisation inter-nœuds par échange de messages
- ***Shared Memory Parallelism***
  - Parallélisation intra-nœud mémoire partagée type threads légers
- ***Offload (if accelerated architecture)***
  - Parallélisation accélérateur de type SIMT
- ***Data Parallelism***
  - Vectorisation SIMD
- ***Instruction Level Parallelism***
  - Exécution concurrentes d'instructions indépendantes (*hardware*)

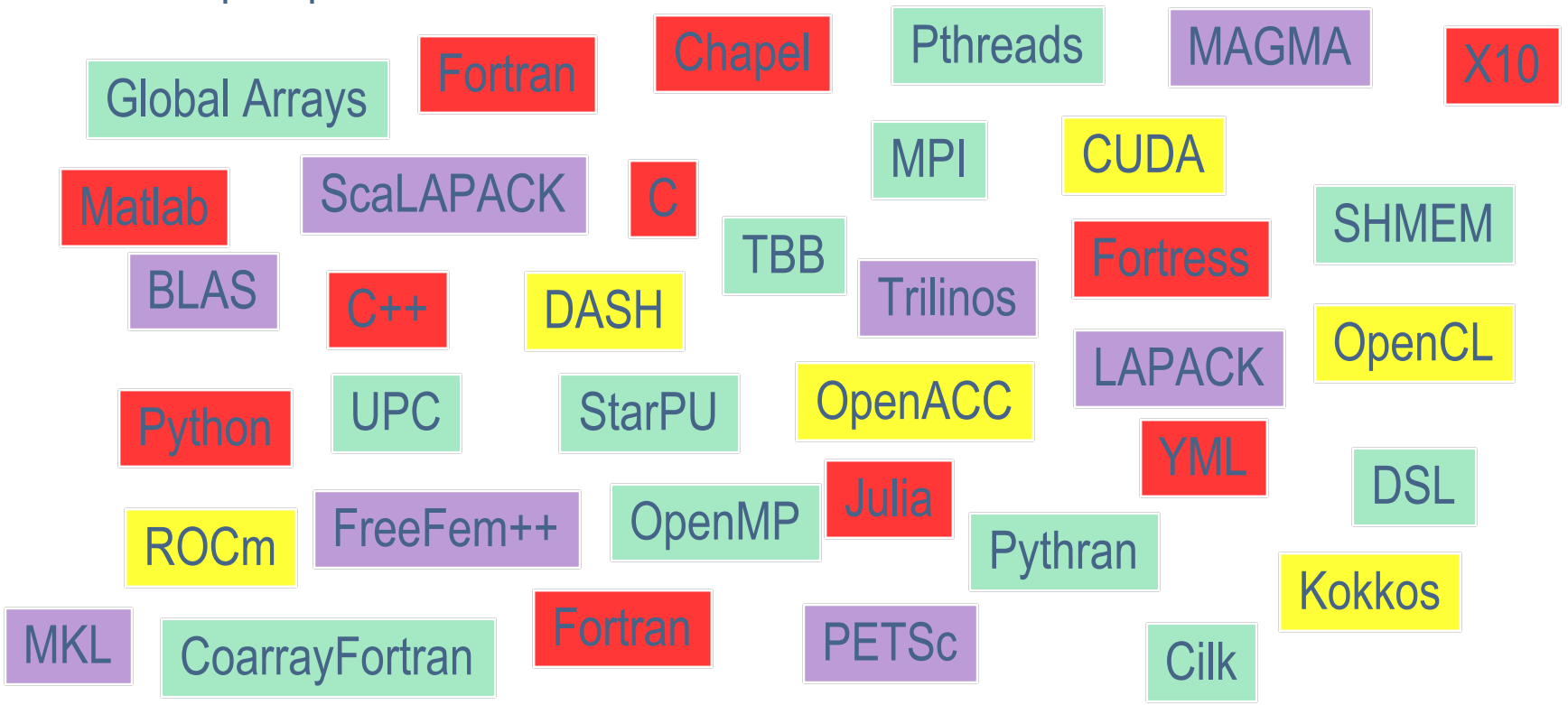



```
V--- > DO i=1,N
|          R(i)=A(i)+B(i)
V--- > ENDDO
```

```
add r3 ← r1, r2
mul r0 ← r0, r1
sub r1 ← r3, r0
```
Parallel

Langages et paradigmes de parallélisation…

L'écosystème de programmation et de parallélisation est riche, très riche, beaucoup trop riche…



Comment faire les bons choix ?

Contraintes d'un centre national Tier1

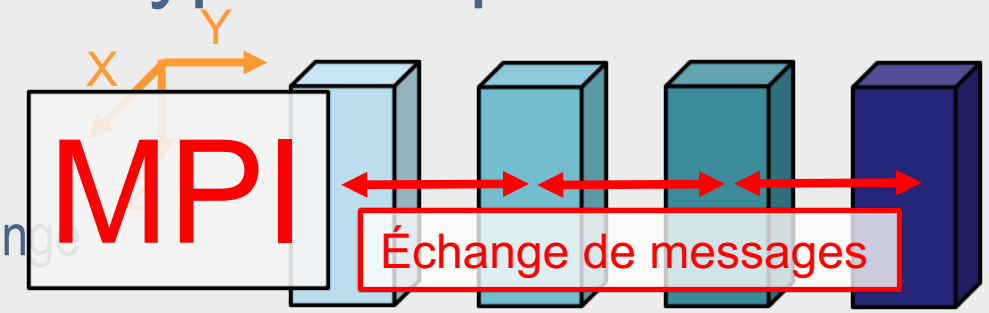
- Vision d'un centre national (IDRIS) gérant 300 projets et 1000 utilisateurs représentant quasiment toutes les disciplines scientifiques ayant des besoins en calcul numérique haute performance
- Centre Tier1 avec des supercalculateurs adaptés à la grande diversité des communautés utilisatrices des ressources (i.e. pas trop exotiques ou spécialisés), un écosystème de développement stable, robuste et pérenne, accessible aussi bien au néophyte qu'aux experts du HPC
- Contrainte forte de production pour optimiser l'utilisation des ressources (24 h / 24, 365 jours par an, disponibilité supérieure à 99 %, charge moyenne mensuelle entre 80 et 95 %)
- Approche parfois conservative par rapport à d'autres acteurs du domaine qui n'ont pas nécessairement les mêmes contraintes (centres thématiques, centres Tier2, laboratoires de recherche, etc.)
- Productivité, pérennité, portabilité, performance, extensibilité, investissement humain, complexité d'implémentation, facilité de débogage, etc.

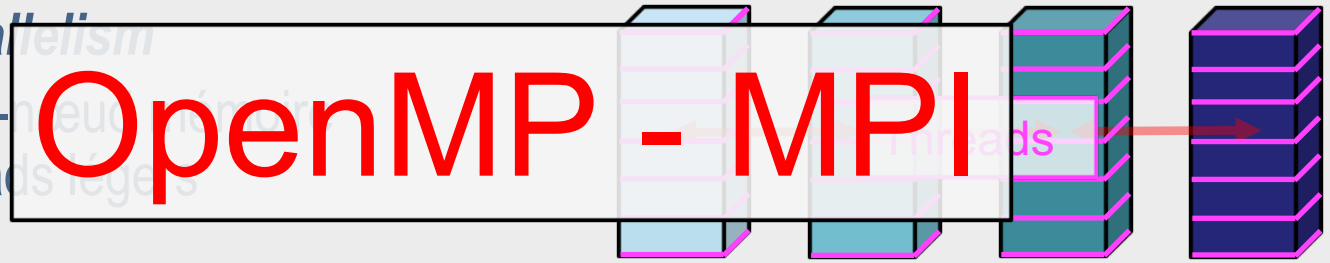
Vision d'un centre national

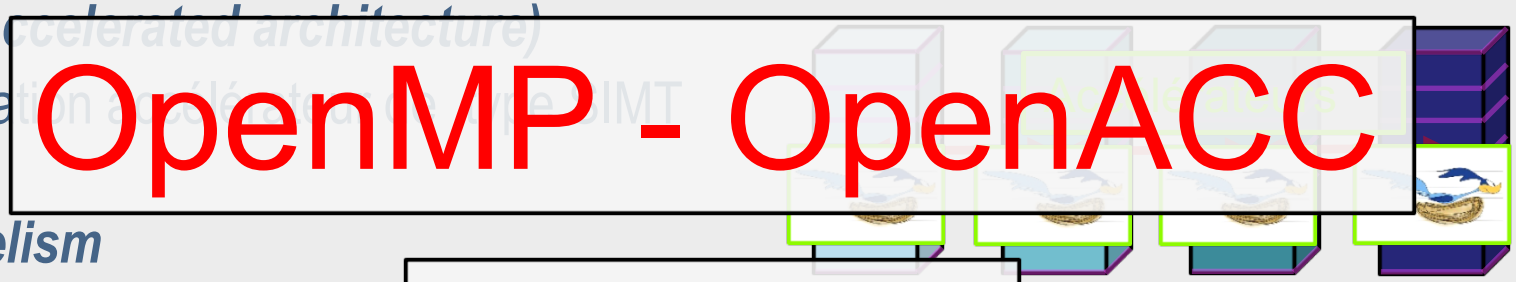
- Rôle de conseil et d'expertise vis-à-vis de nos utilisateurs
- Tous les environnements de développements mis à disposition doivent être robustes, matures, stables, avoir un statut de norme (officielle ou officieuse) avec une communauté d'utilisateurs et de développeurs / contributeurs suffisamment importante et une pérennité établie
- Vision partagée par un grand nombre de centres nationaux à travers le monde
- Permet de faire un filtre et de ne se focaliser que sur un nombre limité d'environnements de développement :
  - Langages de programmation : Fortran, C, C++ et Python
  - Parallélisme : MPI, OpenMP
  - Accélérateurs : OpenACC, OpenMP et CUDA (non recommandé, mais disponible)

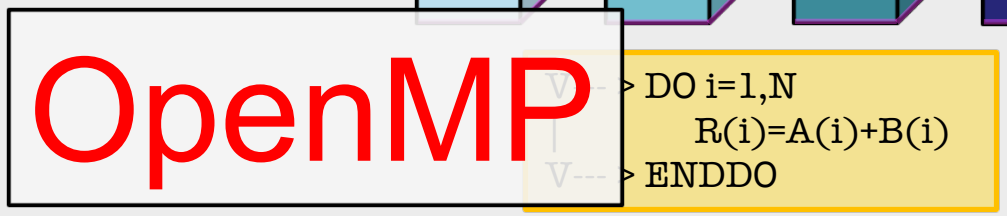
Typologie des différents types de parallélisme

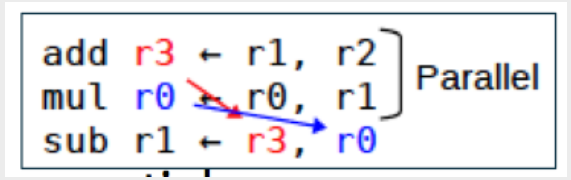
- ***Domain Parallelism***
  - Parallélisation inter-nœud par échange de messages
- ***Shared Memory Parallelism***
  - Parallélisation intra-nœud à mémoire partagée type threads légers
- ***Offload (if accelerated architecture)***
  - Parallélisation accélérateur type SIMT
- ***Data Parallelism***
  - Vectorisation SIMD
- ***Instruction Level Parallelism***
  - Exécution concurrentes d'instructions indépendantes (*hardware*)











Les bibliothèques

De nombreuses bibliothèques sont disponibles dans des domaines très variés (systèmes linéaires, problèmes aux valeurs propres, optimisation, FFT, EDP, AMR, etc.). Lorsque cela est possible, c'est **LA voie à privilégier** :

- des optimisations de bas niveau et des performances excellentes impossibles à obtenir à la main… ;
- gestion implicite du parallélisme et des accélérateurs si la bibliothèque a implémenté ces fonctionalités ;
- portabilité d'une machine à une autre si la bibliothèque n'est pas propriétaire et est disponible ;
- optimisation de l'investissement, plus rapide à implémenter, à maintenir ;
- généralement simple à utiliser, même si on est parfois contraint d'utiliser les structures associées ;
- robustesse, stabilité numérique, MAJ régulière avec des méthodes / algorithmes récents, correction de bogues, etc. ;
- disponibilité de documentations, forums d'aide, etc. ;
- interface avec les langages de calcul scientifique standards (Fortran/C/C++).

MPI

- MPI est une bibliothèque standardisée d'échange de messages, librement disponible, qui vise performance et portabilité sur une grande variété d'architectures (à mémoire distribuée, à mémoire partagée, cluster de SMP, etc.)
- Le standard MPI est disponible : http://www.mcs.anl.gov/mpi/standard.html
- Régulièrement enrichi avec de nouvelles fonctionnalités au fil des versions
- Une application parallélisée avec MPI est un ensemble de processus autonomes exécutant chacun leur propre code et communicant via des appels à des sous-programmes de la bibliothèque MPI :
  - gestion de l'environnement MPI ;
  - communications point à point ;
  - communications collectives ;
  - communications RMA ou OSC ;
  - mémoire partagée au sein d'un nœud ;
  - topologies et types dérivés ;
  - entrées-sorties parallèles avec MPI-IO.

MPI – Environnement

```
program MonCodeMPI
use  mpi
call MPI_INIT(code)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nb_procs,code) ! # de processus MPI
call MP                                                          cessus
...
call MP
end pro
```

MPI_Init_thread

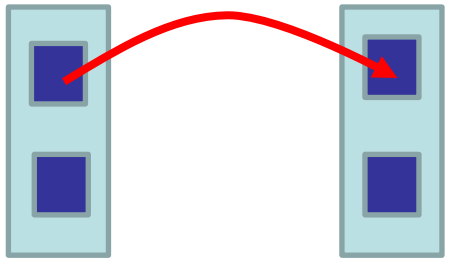
```
int MPI_Init_thread(int *argc, char *((*argv)[]),
                    int required, int *provided)
MPI_INIT_THREAD(REQUIRED, PROVIDED, IERROR)
```

Le niveau de support demandé est fourni dans la variable *required*. Le niveau effectivement obtenu (et qui peut être moindre que demandé) est récupéré dans *provided*.

- `MPI_THREAD_SINGLE` : seul un *thread* par processus peut s'exécuter
- `MPI_THREAD_FUNNELED` : l'application peut lancer plusieurs *threads* par processus, mais seul le *thread* principal (celui qui a fait l'appel à `MPI_Init_thread`) peut faire des appels MPI
- `MPI_THREAD_SERIALIZED` : tous les *threads* peuvent faire des appels MPI, mais un seul à la fois
- `MPI_THREAD_MULTIPLE` : entièrement *multithreadé* sans restrictions

Attenti
pour le
()

MPI – Communications point à point (P2P)

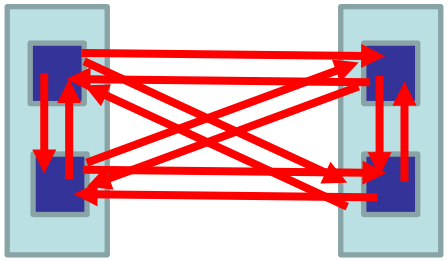


Mode	*Bloquant*	*Non bloquant*
Envoi standard	`MPI_SEND()`	`MPI_ISEND()`
Envoi synchrone	`MPI_SSEND()`	`MPI_ISSEND()`
Envoi *bufferisé*	`MPI_BSEND()`	`MPI_IBSEND()`
Réception	`MPI_RECV()`	`MPI_IRECV()`

Machine	*Niveau*
Blue Gene/Q, PAMID_THREAD_MULTIPLE=0	32%
Blue Gene/Q, PAMID_THREAD_MULTIPLE=1	100%
Ada+POE	37%
Ada+POE MP_CSS_INTERRUPT=yes	85%
Ada+IntelMPI I_MPI_ASYNC_PROGRESS=no	4%
Ada+IntelMPI I_MPI_ASYNC_PROGRESS=yes	94%

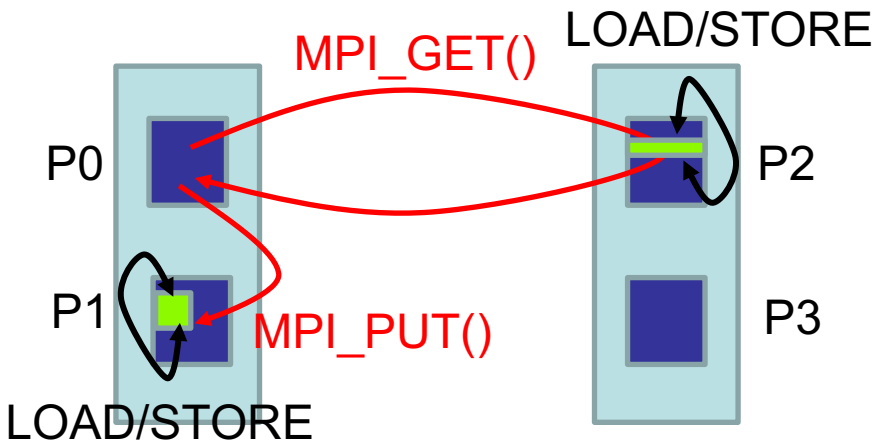
MPI – Communications collectives

- Les communications collectives permettent de faire en une seule opération une série de communications point à point
- Une communication collective concerne toujours tous les processus du communicateur indiqué. Attention aux risques de *deadlock* (si il manque un processus à l'appel) !



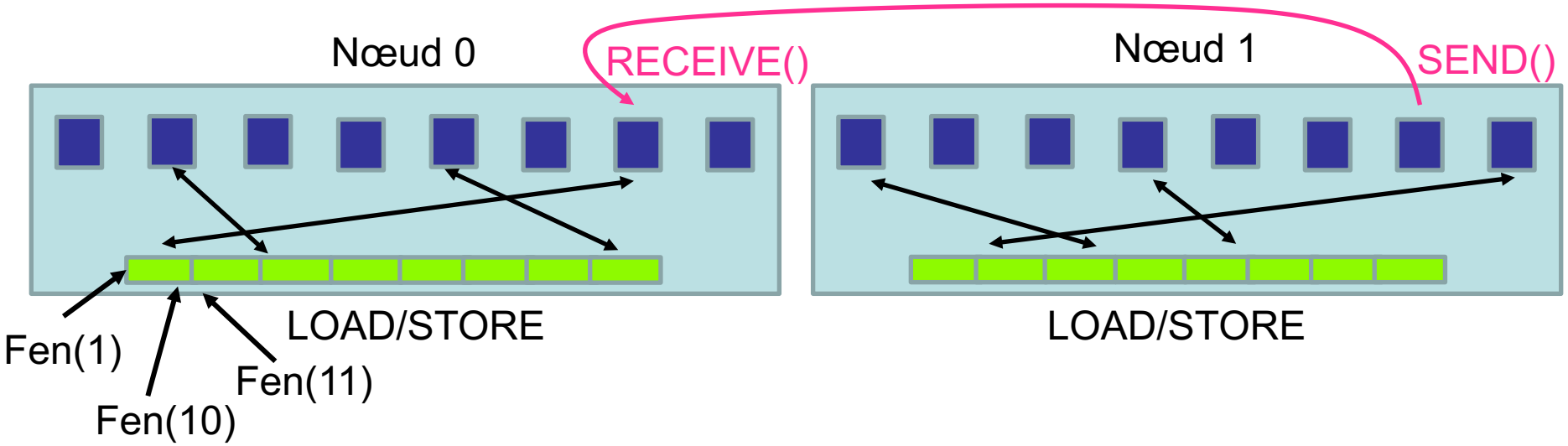
- Depuis MPI-3, il existe des versions non-bloquantes des primitives de communications collectives.
- Préfixée par I (*immediate*) : MPI_IREDUCE(), MPI_IBCAST(), MPI_IBARRIER(), etc.
- Ces primitives non bloquantes permettent d'implémenter un recouvrement calculs / communications
- Attente avec les appels MPI_WAIT(), MPI_TEST() et leurs variantes
- Attention, suivant la machine, l'implémentation MPI et l'environnement d'exécution, cette fonctionnalité peut être implémentée de façon plus ou moins efficace !

MPI – Communications mémoire à mémoire



- Les communications mémoire à mémoire (RMA ou OSC pour *Remote Memory Access* ou *One Sided Communications*) consistent à accéder en écriture ou en lecture à la mémoire d'un processus distant sans que ce dernier doive gérer cet accès explicitement. Le processus cible n'intervient donc pas lors du transfert.
- Nécessite la création d'une fenêtre mémoire et d'un mécanisme spécifique de synchronisation
- Intérêt : optimisation des performances et simplification du codage de certains algorithmes

MPI – Utilisation de la mémoire partagée



- Depuis MPI-3, on peut créer une fenêtre en mémoire partagée pour les processus d'un même nœud
- Les données de la fenêtre mémoire partagée sont accessibles en lecture / écriture (via les mécanismes de *load / store* std) par tous les processus du nœud
- Intérêts : performances accrues, simplicité de programmation, réduction de l'empreinte mémoire…
- On peut ainsi implémenter une parallélisation hybride MPI-MPI, avec un premier niveau MPI intra-nœud utilisant la mémoire partagée et un deuxième niveau MPI (échange de messages) entre les nœuds

OpenMP

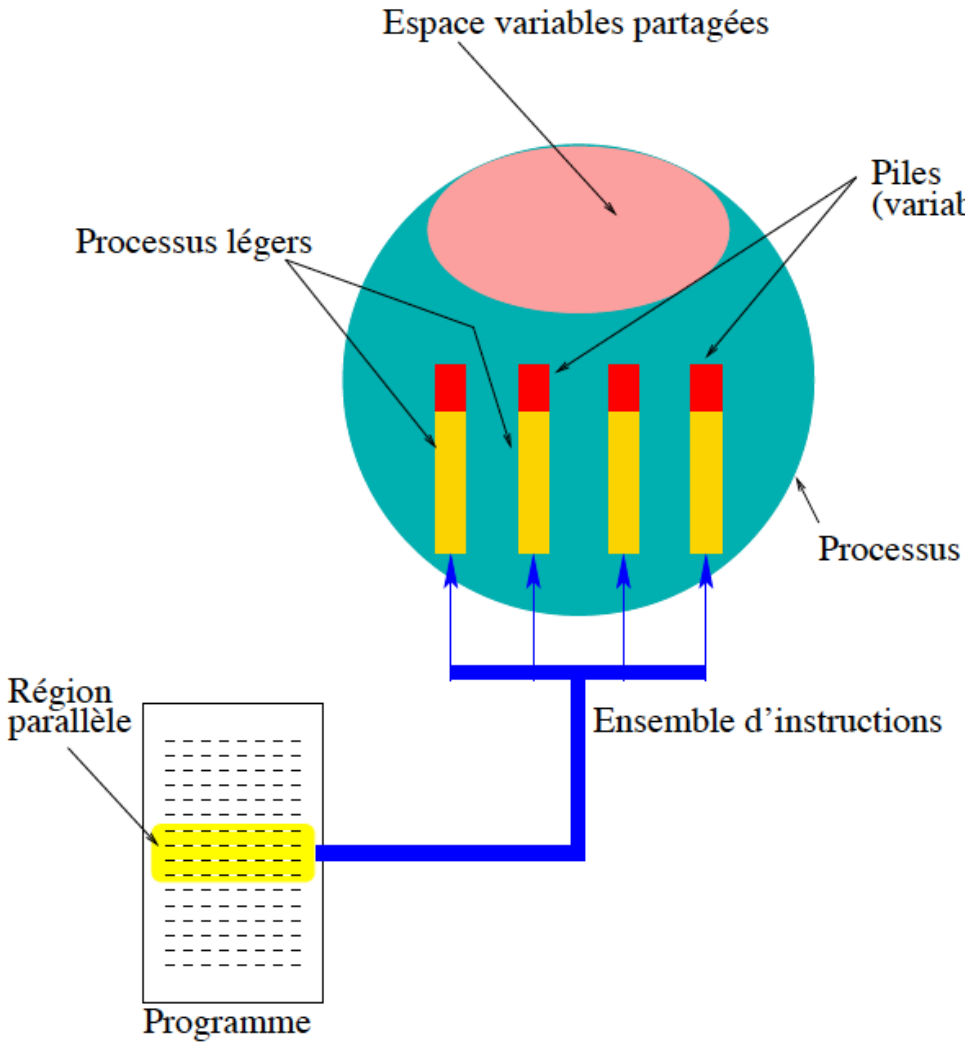
OpenMP est
C++ ou Fortr

- OpenMP
  de variabl
- Spécificat
  - créatio
  - gestior
  - partage
  - synchr
    (ATOM
  - vectori
  - créatio
  - suppor

gée de codes C,

ue de fonctions et
ode

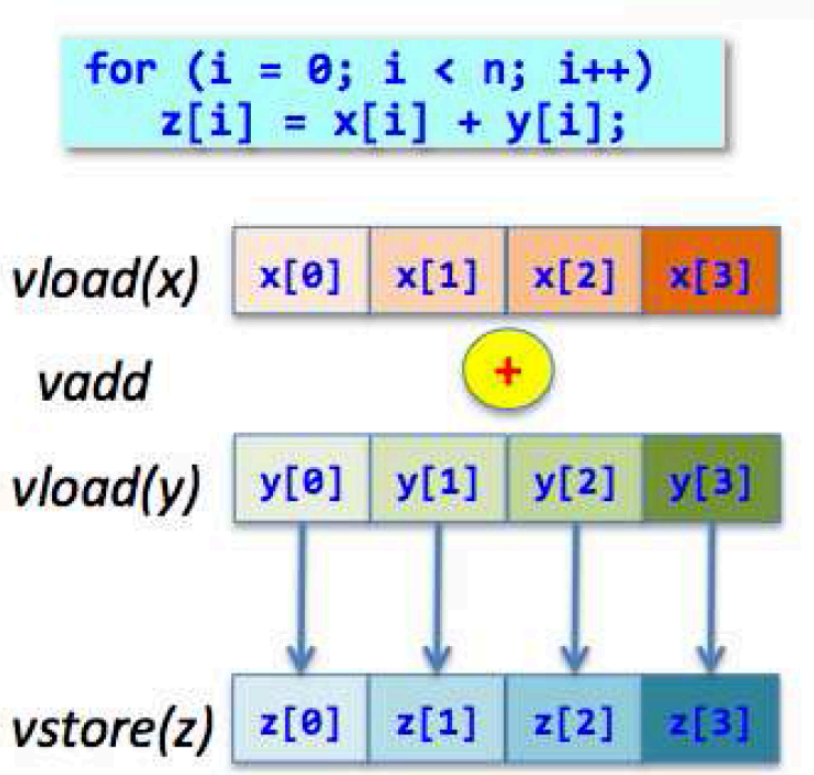
E, SECTION, …) ;
n mutuelle



OpenMP – Vectorisation SIMD

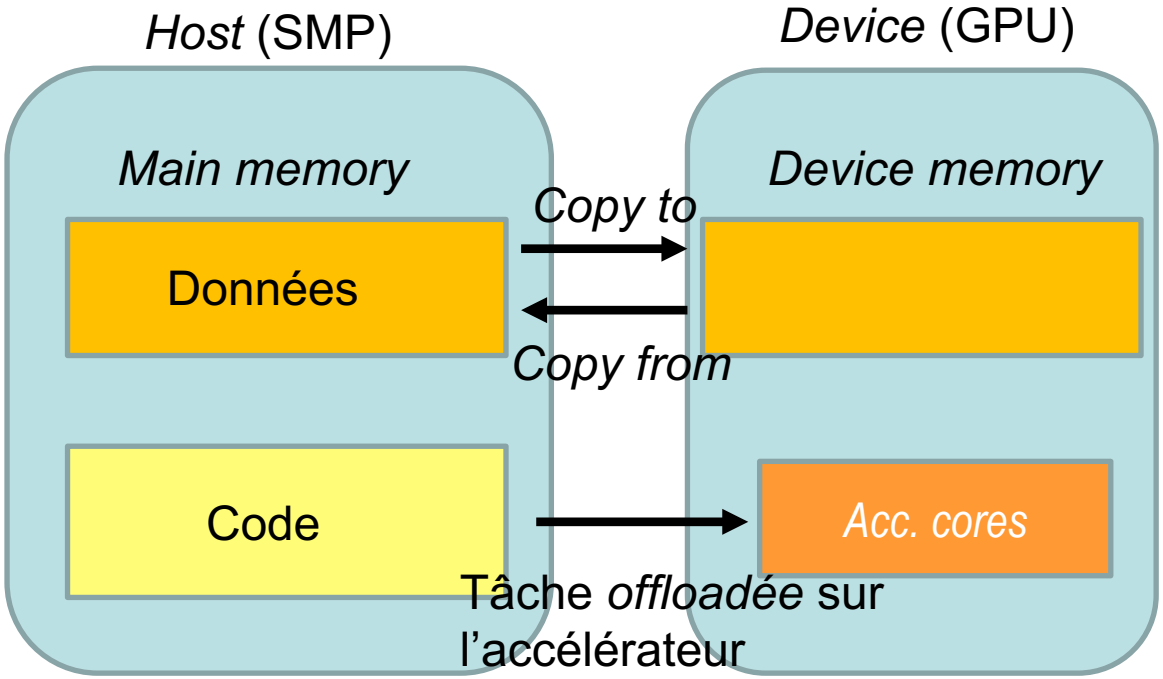
- SIMD = *Single Instruction Multiple Data*
- Une seule instruction / opération agit en parallèle sur plusieurs éléments
- OpenMP 4.0 offre la possibilité de gérer la vectorisation SIMD de façon portable et performante en utilisant les instructions vectorielles (AVX, AVX2, AVX512, SVE) disponibles sur l'architecture cible

```
somme=0
! $OMP SIMD REDUCTION(+:somme)
do i=1,n
   somme=somme+A(i)*B(i)
enddo
```



OpenMP – Offload accélérateurs

- OpenMP 4.0 propose un modèle d'exécution de type *offload* (*data*+code) pour les accélérateurs
- C'est à l'utilisateur de :
  - définir les parties de codes (*TARGET*) à exécuter sur l'accélérateur ;
  - gérer le transfert des données (*MAP*) depuis/vers (*TO/FROM*) la mémoire de l'accélérateur.



OpenMP – *Offload* accélérateurs

```
void vadd_openmp(float *a, float *b, float *c, int size)
{
   #pragma omp target map(to:a[0:size],b[0:size],size) map(from: c[0:size])
   {
      int i;
      #pragma omp teams distribute parallel for
      for (i = 0; i < size; i++)
         c[i] = a[i] + b[i];
   }
}
```

- Les appels *offload* d'OpenMP sont bloquants. L'asynchronisme peut cependant être implémenté en utilisant la notion de tâche explicite.
- Un soin particulier doit être porté aux transferts des données entre les mémoires de l'*host* et du *device,* sous peine d'obtenir des performances décevantes
- Dans la pratique, le but est de ne pas dépasser une dégradation de plus de 10 à 20 % par rapport à une approche bas niveau de type CUDA…

OpenACC – *Offload* accélérateurs

```
void vadd_openmp(float *a, float *b, float *c, int size)
{
   #pragma acc data copyin(a[0:size],b[[0:size]) copyout(c[0:size])
   {
      int i;
      #pragma acc parallel loop
      for (i = 0; i < size; i++)
         c[i] = a[i] + b[i];
   }
}
```

- Le principe est le même que l'*offload* OpenMP
- Spécifications complètes : https://www.openacc.org/
- Les compilateurs sont plus matures et stables que ceux d'OpenMP
- Les fonctionnalités sont plus avancées que celle d'OpenMP (*kernel*, asynchronisme, etc.)
- D'un point de vue utilisateur, on espère une fusion OpenACC / OpenMP *offload* pour ne plus avoir qu'un seul jeu de directives à gérer
- En attendant, OpenACC est une alternative temporaire lorsque les fonctionnalités ou les performances font défaut à l'approche *offload* OpenMP

Conclusions

- Les architectures exascales exhiberont une complexité importante (hiérarchie mémoire, hétérogénéité) et un parallélisme extrême (many-cœurs, SIMD, SIMT, etc.)
- Pour les utilisateurs, quelle que soit l'architecture cible, un travail conséquent d'adaptation des codes de calcul sera nécessaire pour obtenir des performances :
  - exposer plus de parallélisme intra-nœud dans les applications ;
  - augmenter le caractère vectoriel SIMD des applications ;
  - prendre en compte les hiérarchies et les aspects NUMA des différents niveaux de mémoire disponibles (affinité mémoire, affinité processeur, *binding*, …) ;
  - utiliser localement des directives pour les architectures hybrides accélérées ;
  - utiliser plusieurs niveaux de parallélisme, potentiellement avec des paradigmes différents, au sein des applications
- Les outils et les environnements de développement arrivent à maturité, mais il reste à adapter les applications de manière pérenne, un travail qui peut prendre plusieurs années…