Formation utilisateurs HPC@LR

Gestion des ressources pour Muse

Marc Mendez-Bermond – Expert Solutions HPC

Au programme

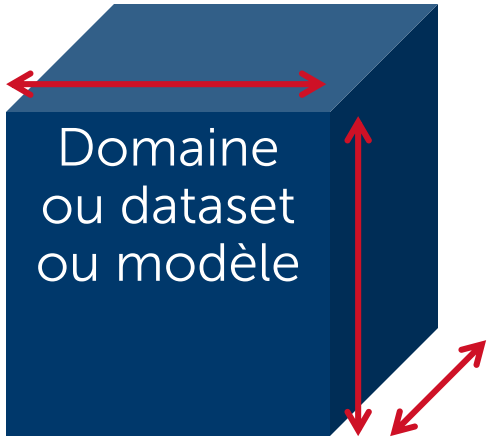
- **Parallélisme et paradigmes de programmation**
  - OpenMP, MPI et OpenMP/MPI
  - Traitement par lots
- **Gestion des ressources**
  - SLURM et ses concepts
  - Scripts de lancement
- **Spécificités Intel MPI et Intel TrueScale**
  - Environnement

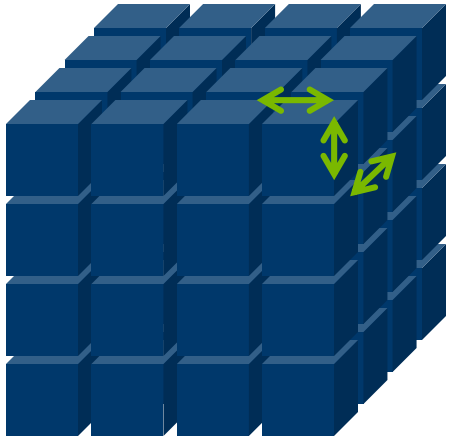
Parallélisme et paradigmes de programmation

Le calcul à hautes performances

- L'objectif est de simuler, traiter ou générer avec l'outil informatique :
  - Simulation d'évènements
  - Traitement de données pour analyse
  - Génération de contenu numérique
- La technologie évolue rapidement :
  - Les performances des processeurs
  - Les capacités et vitesses d'accès des mémoires et du stockage
  - Les débits et latences des réseaux d'interconnexion rapides
- Mais nous demandons :
  - Des modèles physiques plus précis
  - Des maillages plus fins
  - Des temps de restitution plus courts

Principes du calcul parallèle



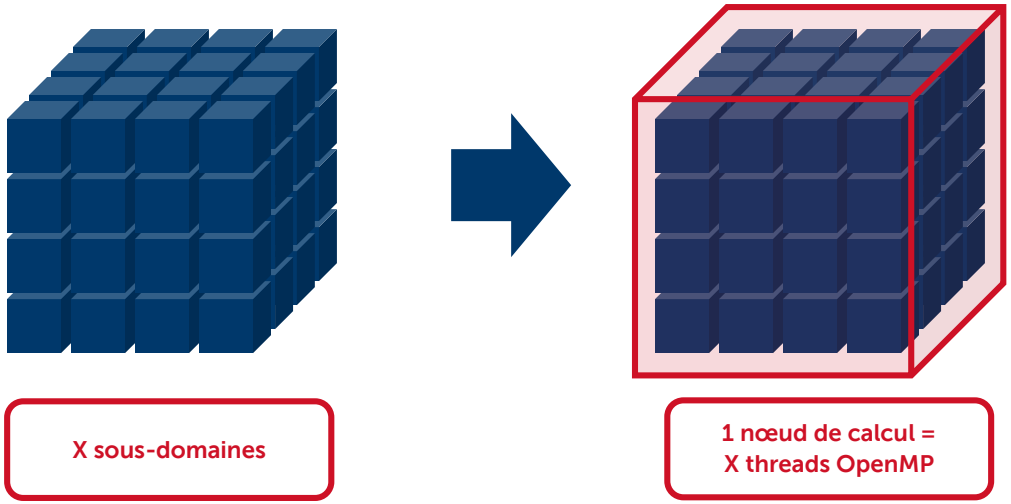


1 problème de taille N
Tseq. = X * Y * Z + seq global

N problèmes de taille 1
T// = x * y * z + seq. global

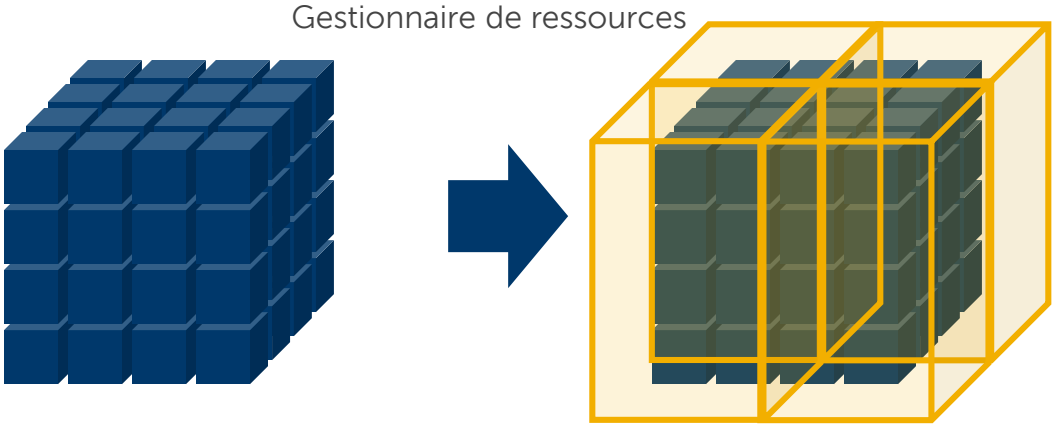
- Décomposition du domaine en sous-domaines adaptés à la physique du phénomène
- L'algorithme numérique implémentera un motif de communications pour coordonner les calculs réalisés pour chacun des sous-domaines
- Multiplication de la puissance de calcul et de la mémoire disponibles au prix d'un effort sur les algorithmes et la gestion de l'infrastructure

Programmation de calculs parallèles : **OpenMP**



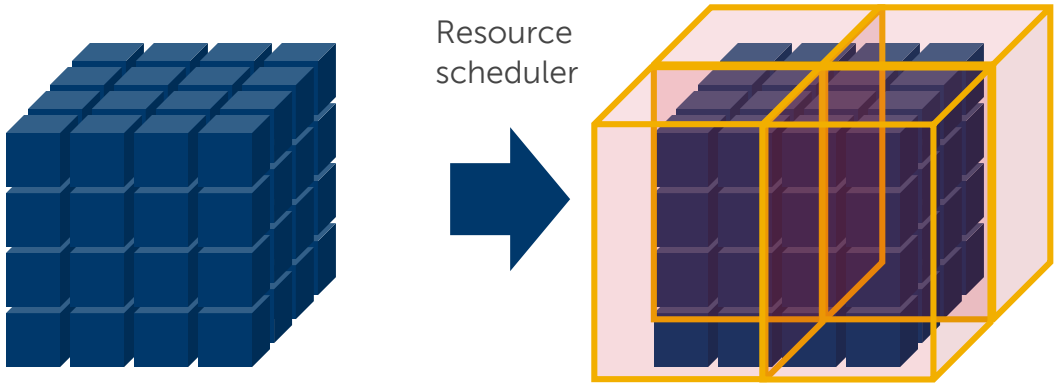
- OpenMP : parallélisation par **multithreading**
- Usage : directives simples insérées dans le code source pour indiquer les sections parallèles, synchronisations, données partagées et autres réglages
- **(Aujourd'hui) aucun support pour les communications inter-nœuds**

Programmation de calculs parallèles : **MPI**



- MPI : bibliothèque d'envoi de messages entre **tâches MPI**
- Usage : programmation spécifique, totalement explicite
- **Communications intra et inter-nœuds pour 1 à 100000 nœuds**
- Réseau d'interconnexion rapide largement souhaité : faible latence et haut débit

Programmation de calculs parallèles : **MPI/OpenMP**



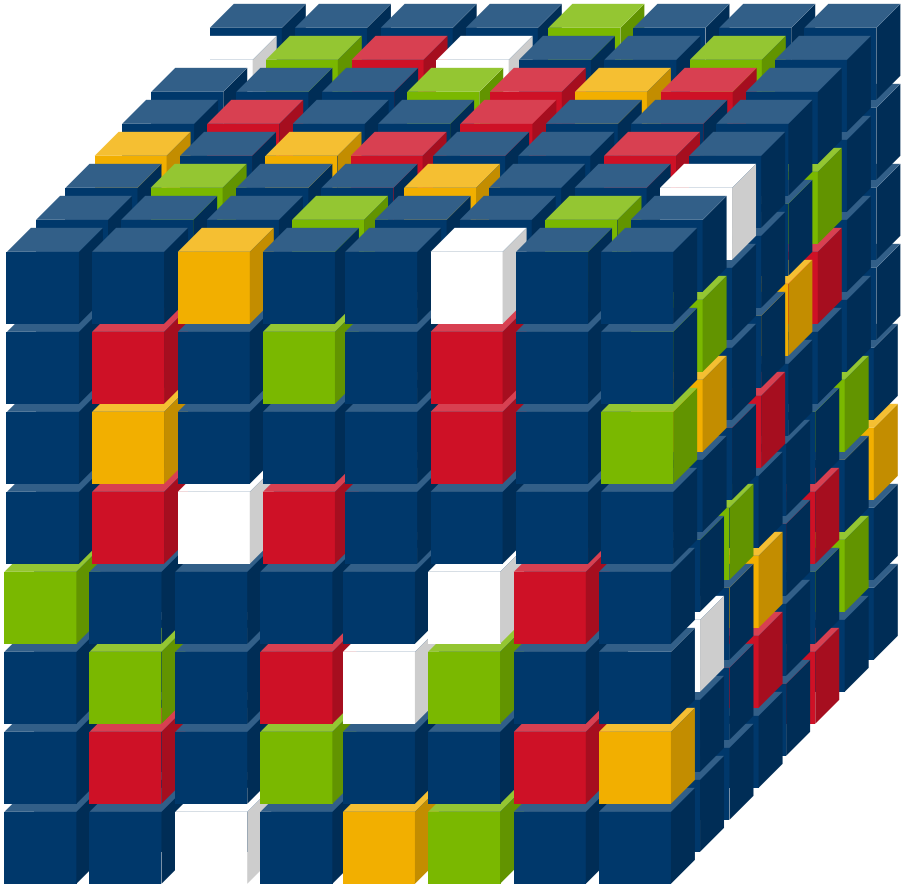
Exemple :

64 sous-domaines

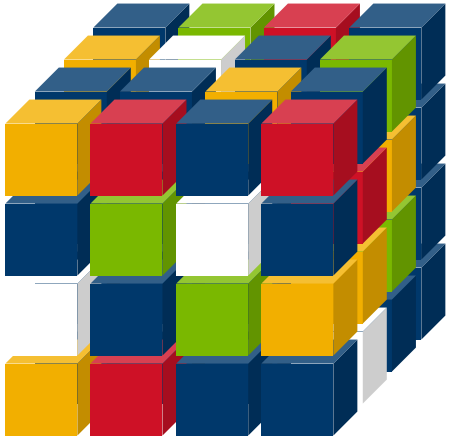
4 tâches MPI
x 16 threads OpenMP
= 64 cœurs

- Mode hybride avec :
  - OpenMP en intra-nœud pour distribuer des **threads** sur les cœurs de calcul
  - MPI pour les communications inter-nœuds entre **tâches MPI**
- **Généralement un paradigme très performant, évolutif et très efficace énergétiquement**

Traitement parallèle par lot : grilles de calcul/séquentiel



Batch processing



X problèmes de taille 1 exécutés sur une grille de N cœurs

=

Accélération ~N

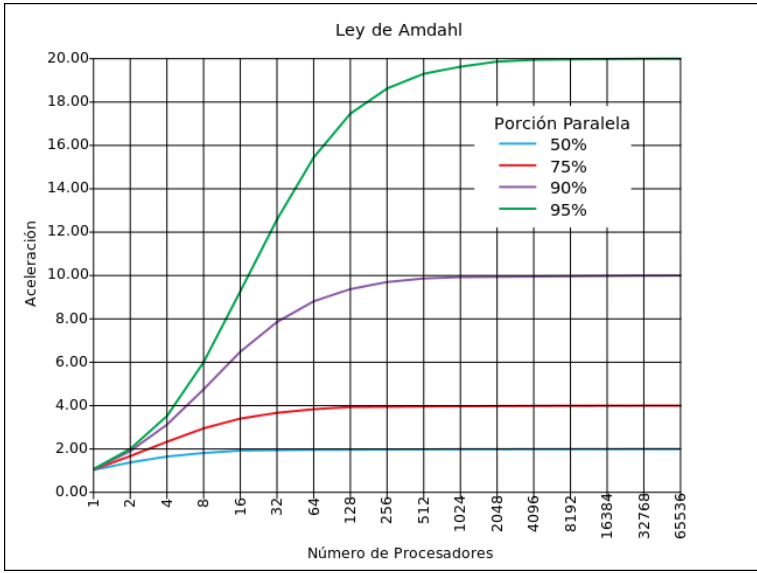
Les nœuds de calcul peuvent être distribués géographiquement !

Facteurs de performances

- Le parallélisme permet d'accroitre les performances dans les limites de la loi d'Amdhal :

$$R = \frac{1}{(1-s) + \frac{s}{N}}$$

- Cette formule met en relation le taux de parallélisme de l'application (s, en fraction du temps d'exécution) avec le nombre de processeurs disponibles pour une exécution parallèle (N).



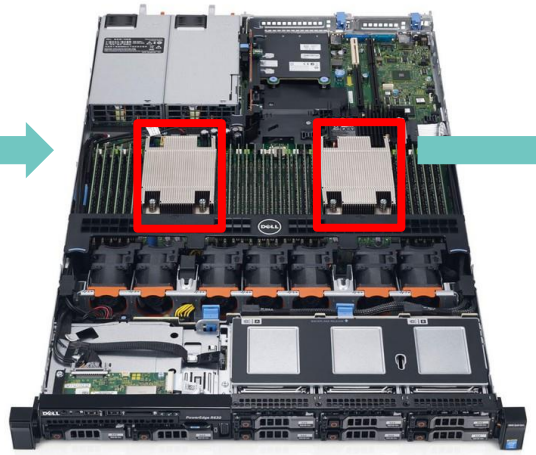
Gestion de ressources

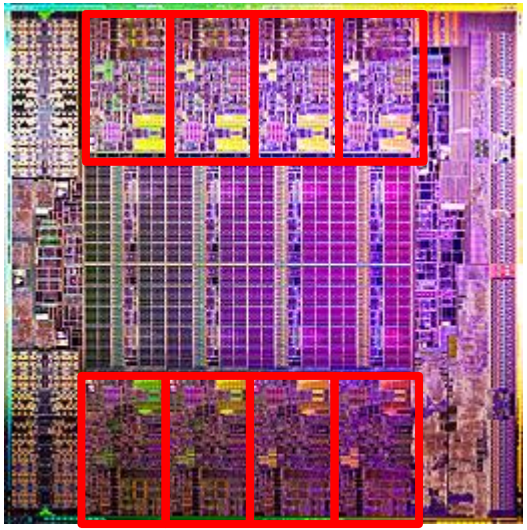
Gestion de ressources et SLURM

- Un **ordonnanceur** organise l'allocation des ressources disponibles du supercalculateur.
- Les **travaux** de l'ensemble des utilisateurs sont orchestrés en fonction :
  - Des ressources demandées (nombre de cœurs/nœuds, mémoire, …) – ajustées par l'utilisateur
  - De la politique de priorités du supercalculateur – ajustée par l'administrateur
- L'**ordonnanceur** est un outil qui organisera l'exécution, la collecte des informations de l'exécution, les fichiers de sortie et les éventuelles défaillances.
- SLURM est l'ordonnanceur installé sur Muse
  - SLURM (*Simple Linux Utility for Resource Management*) est un pur développement libre démarré par le LLNL pour répondre aux besoins des supercalculateurs dotés de CPU multi-cœurs.

Hiérarchie des ressources des grappes de calcul

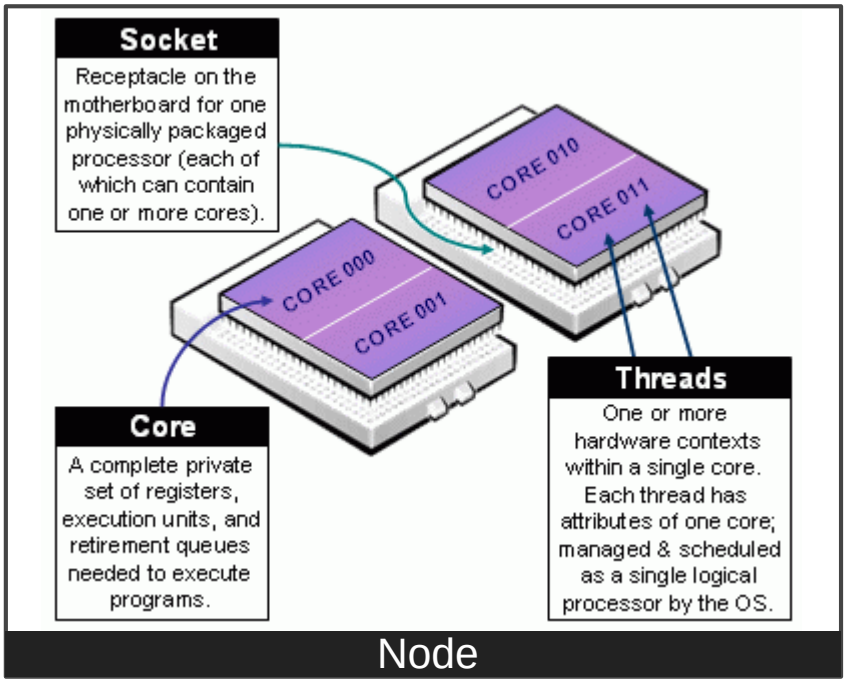
La **grappe** est composée de **nœuds de calcul**	Chaque **nœud de calcul** dispose d'1 ou plusieurs **processeurs**	Les **processeurs** sont composés de plusieurs **cœurs** de calcul





SLURM : Multi-core/Multi-thread

Il est nécessaire de comprendre les correspondances entre le vocabulaire SLURM et celui des systèmes et applications !



Le **node** correspond à un nœud de calcul ou serveur.

Le **socket** est un arrangement physique pour les cœurs de processeur qui partagent certaines ressources : cache L3 et canaux mémoire en particulier.

Le **cœur** est l'unité élémentaire de calcul qui est le grain le plus fin d'allocation. Il est parfois aussi appelé processeur ou CPU du fait d'un lexique remontant aux processeurs mono-cœurs.

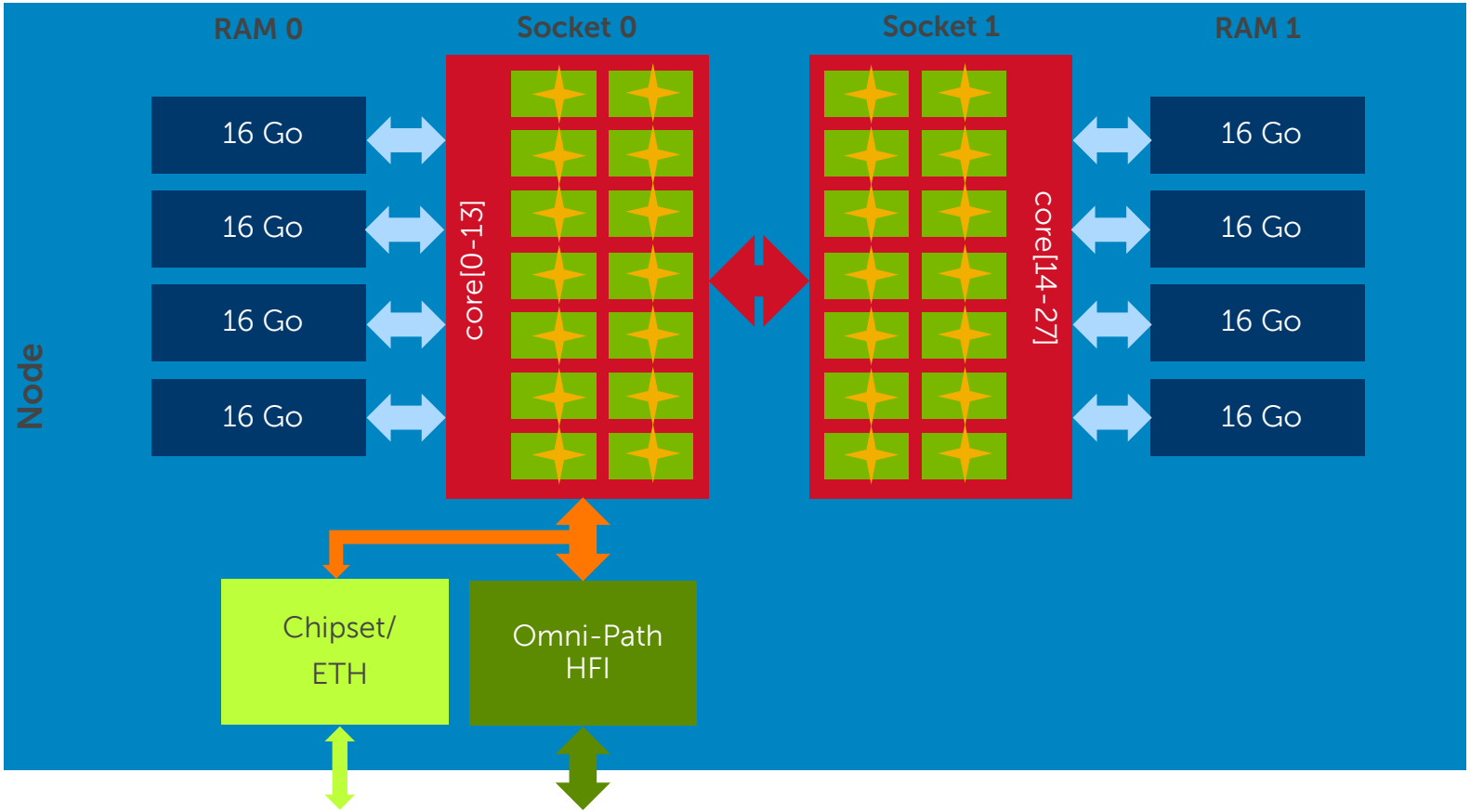
Le **thread** SLURM correspond aux contextes matériels proposés par la technologie Intel Hyper-Threading. Vous pouvez généralement assumer qu'un thread est équivalent à un cœur.

Le supercalculateur MUSE

280 Tflop/s Linpack - 115 kW – 10 armoires 42U

- 308 nœuds de calcul fins [Dell PowerEdge C6320]
  - 2x processeurs 14 cœurs @ 2.4 GHz (Intel Xeon E5-2680 v4)
  - 128 Go @ 2400 MT/s
  - Diskless
  - Interface Intel Omni-Path 100 Gb/s
- 1 espace de stockage pérenne NFS [Dell NSS H/A 6.0]
  - 350 To - ~2 Go/s W
  - Haute-disponibilité
- 1 espace de stockage rapide Lustre [Dell IEEL/OPA 3.0]
  - 1.1 Po - ~6 Go/s W
  - Haute-disponibilité
- Réseau d'interconnexion Dell/Intel Omni-Path 100 Gb/s
  - Support MPI, NFS, Lustre
- Réseau d'administration Dell Networking Ethernet 10 Gb/s
  - Administration, supervision et autres communications d'infrastructures

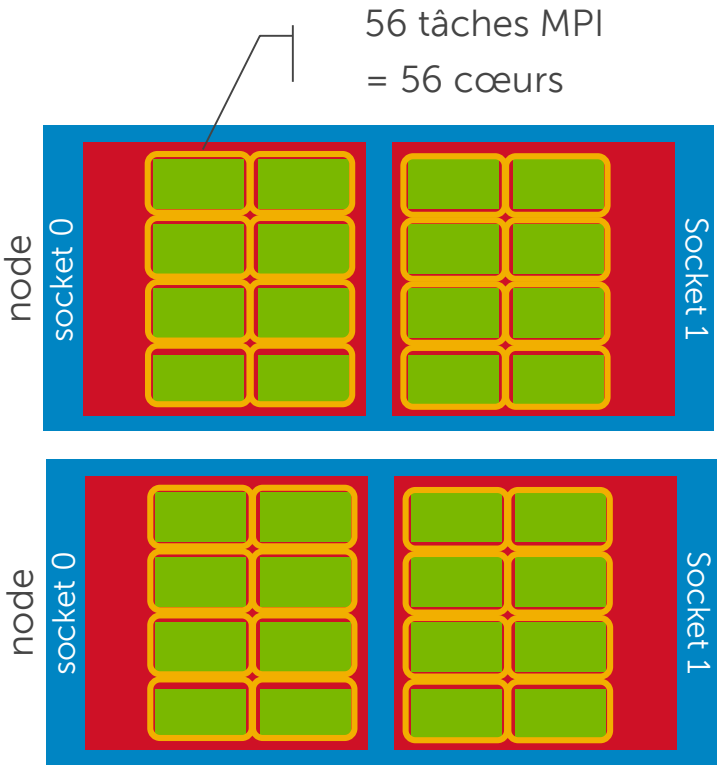
Architecture d'un nœud de calcul



Exécution MPI (simple)

- Exécution MPI avec N tâches
- SLURM requiert les paramètres suivants :
  - Le nombre de tâches
  - Ou le nombre de nœuds

```
$ srun –n 56 ./mpi.x
          ou
$ srun –N 2 ./mpi.x
```



n= 56 ou N=2

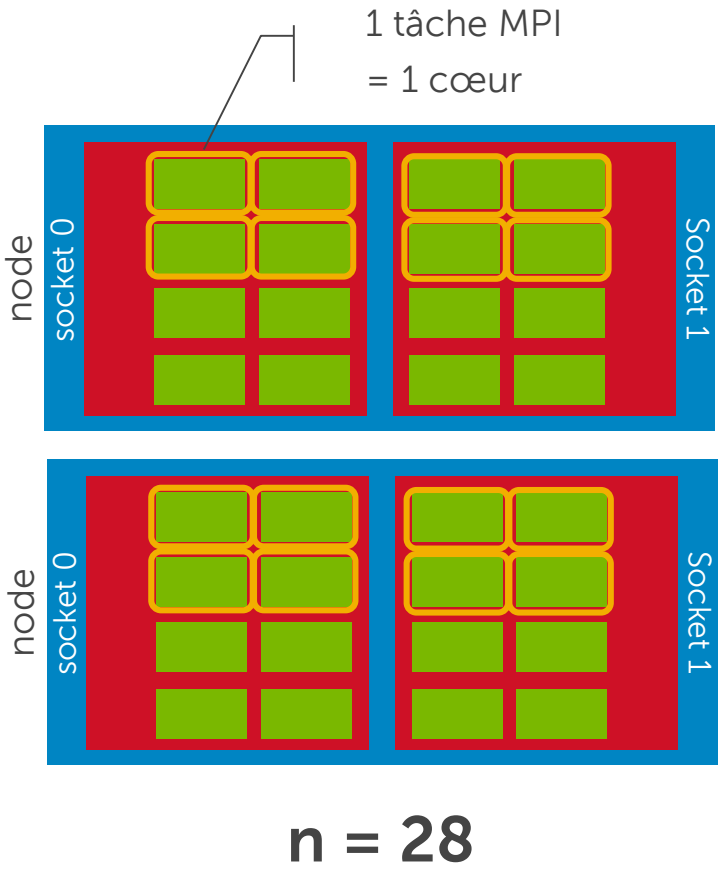
Exécution batch MPI (simple toujours)

Intel MPI ou OpenMPI

```
#!/bin/bash
#SBATCH --job-name mpi
#SBATCH --output mpi-%j.out
#SBATCH -N 2

cd $SLURM_SUBMIT_DIR
mpirun ./mpi.x
```

Exécution MPI (moins simple)

- Exécution MPI avec N tâches en « dépeuplant » de 50%
- SLURM requiert les paramètres suivants :
  - Le nombre de nœuds : *nodes*
  - Le nombre de tâches par nœud : *tpn*
  - Avec n = *nodes* * *tpn*



n = 28

```
$ srun -N 2 --ntasks-per-node 14 ./mpi.x
```

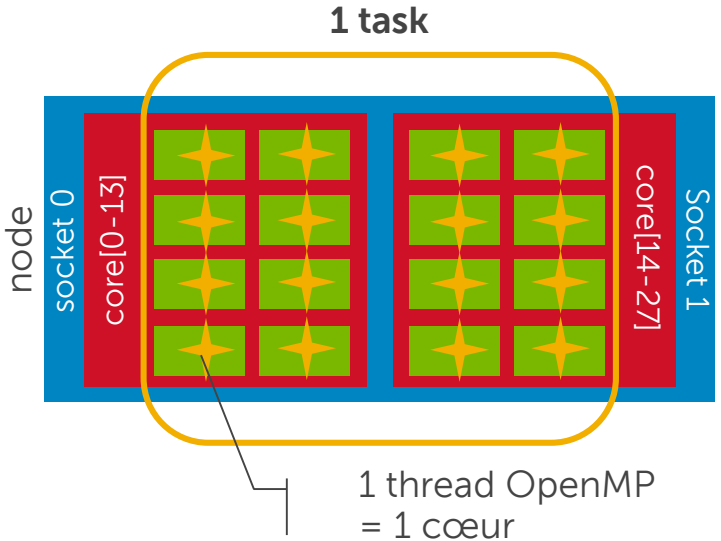
Exécution batch MPI – placement logique

Intel MPI ou OpenMPI

```
#!/bin/bash
#SBATCH --job-name mpi2
#SBATCH --output mpi2-%j.out
#SBATCH -N 2
#SBATCH --ntasks-per-node 14

cd $SLURM_SUBMIT_DIR
mpirun ./mpi.x
```

Exécution OpenMP

- Exécution OpenMP sur 1 nœud et N threads
- SLURM requiert les paramètres suivants :
  - 1 nœud
  - 1 tâche par nœud
  - N cœurs par tâche (N doit être inférieur ou égal au nombre de cœurs disponibles sur 1 nœud !!!)




```
$ OMP_NUM_THREADS=28 srun -N 1 --ntasks-per-node=1 --cpus-per-task=28 ./openmp.x
                                        Ou
$ OMP_NUM_THREADS=28 srun -n 1 --cpus-per-task=28 ./openmp.x
                                        Ou
$ OMP_NUM_THREADS=28 srun -N 1 ./openmp.x
```

Exécution batch OpenMP

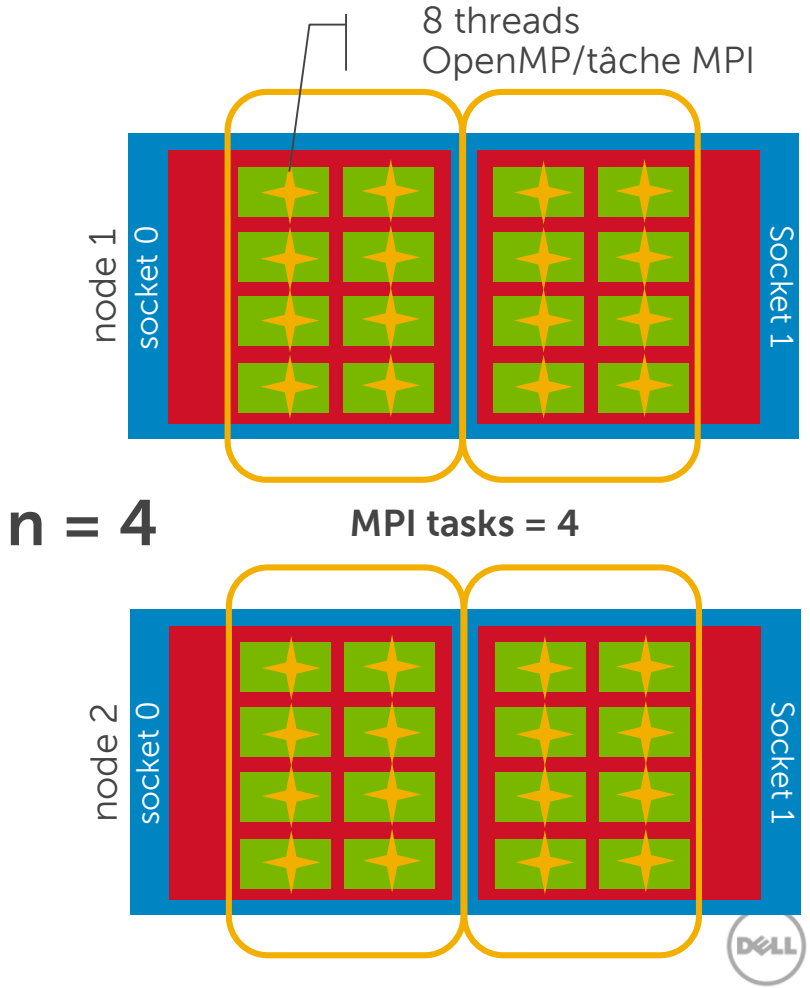
Intel C/C++/Fortran ou GNU Compilers

```
#!/bin/bash
#SBATCH --job-name openmp
#SBATCH --output openmp-%j.out
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28

cd $SLURM_SUBMIT_DIR

# OpenMP runtime settings
export OMP_NUM_THREADS=$SLURM_CPUS_ON_NODE

./openmp.x
```

Exécution OpenMP/MPI

- Exécution MPI/OpenMP
  - 2 niveaux hiérarchiques pour le parallélisme
    - Le nombre de tâches MPI à distribuer sur les nœuds
    - Le nombre de threads OpenMP par tâche MPI
- SLURM requiert les paramètres suivants :
  - Le nombre de tâches
  - Le nombre de tâches par nœud
  - Le nombre de processeurs/cœurs par tâche

```
$ srun –N 2 --ntasks-per-node 2 \
       --ncpus-per-task 14 \

       ./ompi.x
```



Exécution batch OpenMP/MPI

Intel MPI ou OpenMPI – Intel C/C++/Fortran ou GNU

```
#!/bin/bash
#SBATCH --job-name ompi
#SBATCH --output ompi-%j.out
#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=14

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

cd $SLURM_SUBMIT_DIR
mpirun ./ompi.x
```

Merci !

marc_mendez_bermond@dell.com