Portage KNL et GPU

Patrick Bousquet-Mélou, CRIANN
4 juin 2018

CRIANN

Sommaire

Application de traitement d'image

Application considérée

Détection de fissure en surface de béton

- Application de recherche en traitement mathématique d'image
  - Auteures du LMI - EA 3226 (Laboratoire de Mathématiques de l'INSA Rouen)
    - Carole Le Guyader et Noémie Debroux

- Caractéristiques influentes pour le développement et la performance HPC
  - Différences finies 2D, maillage structuré
  - Conditions aux limites
    - Bords : Neumann homogène
    - Coins : symétrie
  - Schéma en espace d'ordre élevé
    - ordre 2 avec diagonales => schéma à 17 points
  - Schéma en temps explicite
  - Taille du problème : image considérée de (2224 x 3996) pixels
  - Double précision

- Potentiel de vectorisation (en plus d'une parallélisation MPI)

Historique des développements HPC

de l'application considérée

- 2017 : MPI
  - Travail motivé par le hackathon 2017 (suggéré par Genci auprès des mésocentres)
  - Parallélisation (décomposition en domaines 2D) de deux modèles par Carole Le Guyader et Noémie Debroux, avec la participation de trois étudiants INSA Génie Mathématique (et conseils du CRIANN)
    - Code «local»
    - Code «non local» : terme intégral supplémentaire dans l'équation aux dérivées partielles (EDP)
  - Propriétés de la version 2017 du code «local»
    - Ses résultats numériques sont strictement indépendants du nombre de tâches MPI, et strictement identiques à ceux du code séquentiel initial
    - 98% d'efficacité parallèle de 1 à 448 cœurs Broadwell
- 2018 : portage sur architectures many-core du code «local»
  - Xeon Phi KNL
  - GPU K80 et P100

Architectures considérées

Broadwell, KNL, K80 et P100

Architectures de Myria

- CPU
  - Broadwell (E5-2680 v4)
    - version 14 cœurs 2,4 GHz
    - unités vectorielles de 256 bits (AVX2)
    - 2 x 0,537 TFlop/s dans un serveur
  - KNL (Xeon Phi(TM) CPU 7210)
    - version 64 cœurs 1,3 GHz
    - unités vectorielles de 512 bits (AVX512)
    - configuration : clustering *quadrant* / mémoire *flat* (mémoire rapide MCDRAM en espace d'adressage : voir lien knl ci-contre)
    - 2,66 TFlop/s
- GPU
  - Serveurs à 2 cartes K80
  - Serveurs à 2 cartes P100

Liens utiles

http://www-tech.criann.fr/calcul/tech/myria-doc/config/

http://www-tech.criann.fr/calcul/tech/myria-doc/knl

http://www-tech.criann.fr/calcul/tech/myria-doc/gpgpu/

Modèle de carte GPU	**K80 (2 unités de traitement GPU)**	**P100**
Mémoire	2 x 12 Go	12 Go
Nombre de cœurs	2 x 2496	3584
Nombre de multiprocesseurs	2 x 13	56
Puissance crête (SP = simple précision, DP = double précision)	2 x 4,37 TFlops SP 2 x 1,455 TFlops DP	9,3 TFllops SP 4,7 TFlops DP

Vectorisation sur Broadwell et KNL

Optimisation et vectorisation

CPU

- Optimisation
  1. certaines copies de tableaux remplacées par des échanges de pointeurs
  2. option motivée par le portage ultérieur en CUDA : tableaux 2D remplacés par tableaux 1D (tab[i][j] -> tab[ixNcol + j])

- Vectorisation : directive *simd* d'*OpenMP 4*
- Compilation (Intel) pour Xeon Broadwell

  *-qopenmp -O3 -xCORE_AVX2 -fp-model precise *

  -qopt-report
- Compilation (Intel) pour Xeon Phi KNL

  *-qopenmp -O3 -xMIC_AVX512 -fp-model precise *

  -qopt-report

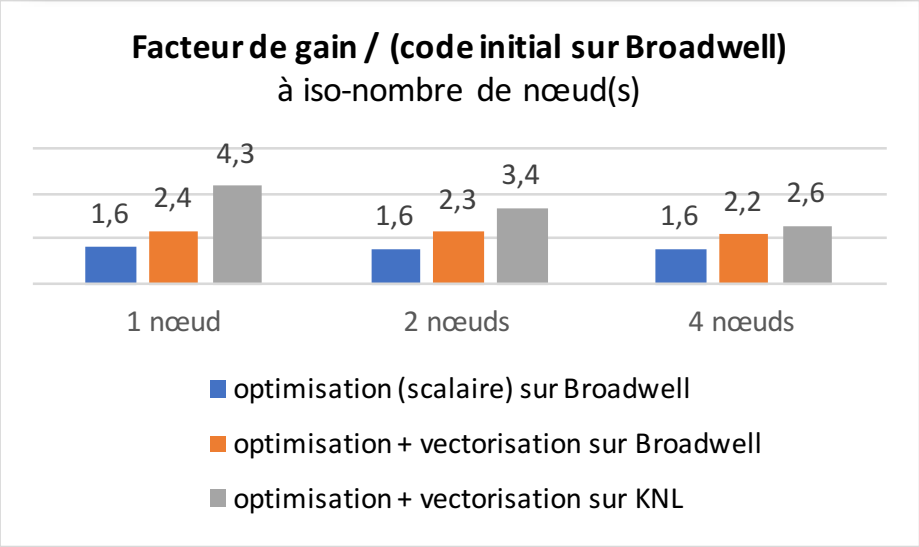
- Résultats numériques strictement identiques à ceux du code initial

```
for (int i = 2; i < N; i++) {
#pragma omp simd private (...)
    for (int j = 2; j < M; j++) {
   ...
```

```
     Report from: Loop nest, Vector & Auto-
parallelization optimizations [loop, vec, par]

LOOP BEGIN at calcul.c(43,8)
   remark #15542: loop was not vectorized: inner
loop was already vectorized

 LOOP BEGIN at calcul.c(45,11)
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
 LOOP END
```

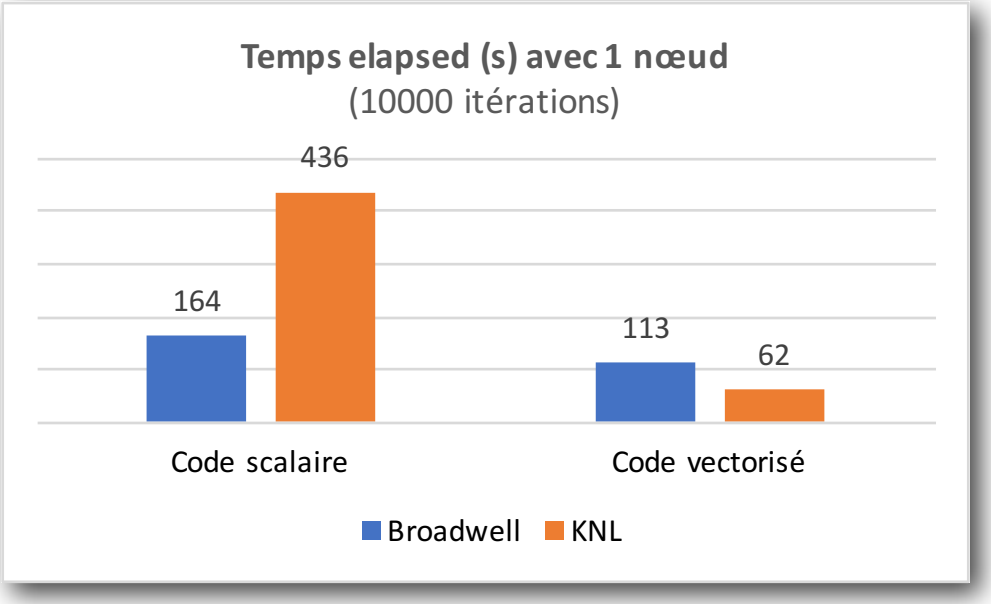


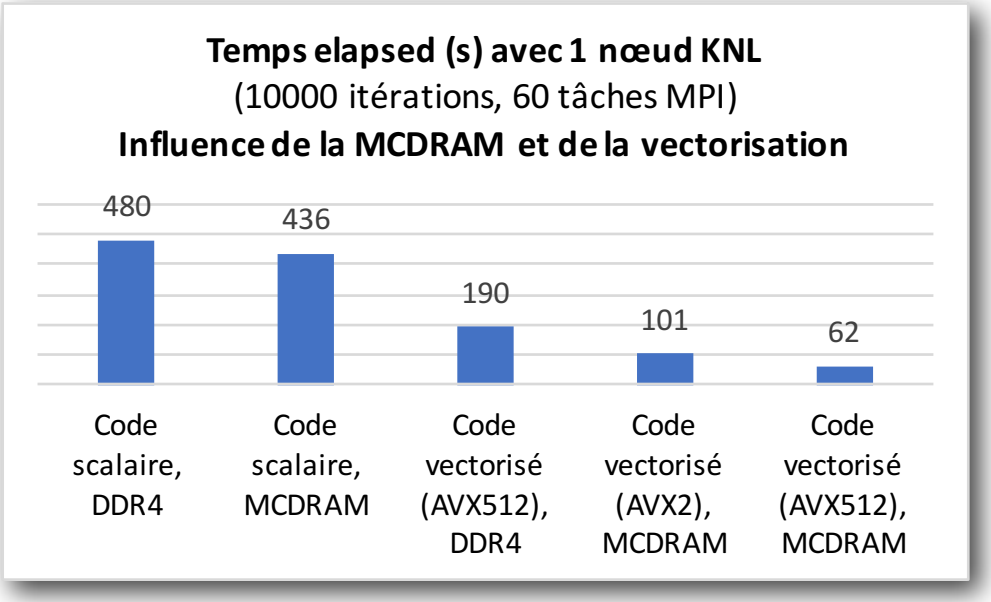
28 tâches MPI par nœud Broadwell,
60 à 64 tâches MPI par nœud KNL

MCDRAM et AVX512

Deux points forts du KNL

- Code scalaire
  - Directive simd d'OpenMP désactivée
- Code vectorisé
  - Directive simd appliquée
    - AVX2 (compilé avec -xCORE-AVX2
    - AVX512 (compilé avec -xMIC-AVX512)
- Utilisation de la mémoire d'un KNL
  - mode de mémoire «flat» (espace d'adressage) configuré pour la MCDRAM
  - usage DDR4 (2133 MHz) : commande :
    - « srun *numactl --membind=0* ./a.out »
  - usage MCDRAM (6400 MHz) : commande :
    - « srun *numactl --membind=**1*** ./a.out »





GPGPU

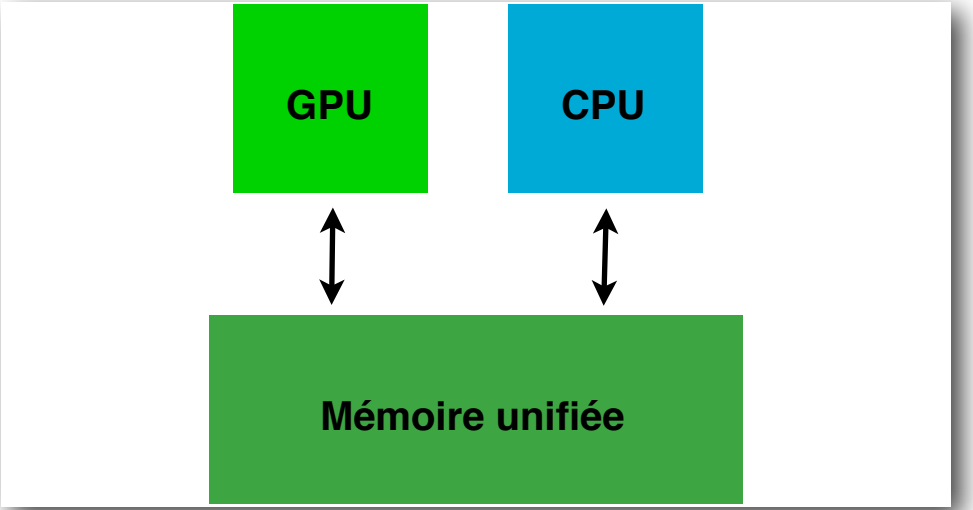
Portage sur GPU

Options délibérées/établies pas à pas

- Mémoire unifiée (CUDA≥6, GPU≥Kepler)
  - Espace d'adressage commun pour mémoire CPU (système) et mémoire globale d'un GPU
    - Mêmes pointeurs pour kernels GPU et code CPU (dont communications MPI)
  - Fonctionnalité de CUDA optimisée au passage de Kepler à Pascal

- 1 tâche MPI par unité de traitement GPU

- Première version GPU par *OpenACC*
  - Après déport sur GPU de l'EDP, les conditions limites doivent-elles être traitées sur CPU ou sur GPU ?
  - OpenACC répond à la question par simple ajout de directives




```
 43 #pragma acc kernels
 44 {
 45 #pragma acc loop independent
 46     for (int i = 2; i < N; i++) {
 47 #pragma acc loop independent
 48         for (int j = 2; j < M; j++) {
    ...
    }
    ...
// Condition au bord
134 #define neumann_bc_NORTH(array) \
    ...
138   if (voisin[N] == MPI_PROC_NULL) {
139 #pragma acc parallel loop
140     for (int j = 2; j < M; j++) {
141         neumann_bc_NORTH(u_new);
    ...
    }
```

OpenACC

Diagnostics à la compilation

```
login@Myria-2:~:$ module purge >& /dev/null
login@Myria-2:~:$ module load pgi/18.4 pgi-18.4_openmpi-3.0.1
load pgi/18.4 environment (OpenACC and CUDA FORTRAN support)
load pgi-18.4_openmpi-3.0.1 environment
login@Myria-2:~:$ make
```

```
 mpicxx -acc -Minfo=accel -ta=tesla:cc60 -ta=tesla:managed -O2 -c calcul.c
 calcul(int *, double *, double *, double *, double *, double *, double *,
double *, double *,  double *, double *, double *, double *, double *, int
*):
  44, Generating implicit copy(g2_local_mat_new[:])
      Generating implicit copyin(tab_bounds[:4])
      Generating implicit copy(u_local_mat_new[:])
      Generating implicit copyin(v1_local_mat[:])
      Generating implicit copy(v1_local_mat_new[:],v2_local_mat_new[:])
      Generating implicit copyin(v2_local_mat[:])
      Generating implicit copy(g1_local_mat_new[:],Q_local_mat_new[:])
      Generating implicit

copyin(u_local_mat[:],F_local_mat[:],Q_local_mat[:],g1_local_mat[:],g2_loca
l_mat[:])
     46, Loop is parallelizable
     48, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
     46, #pragma acc loop gang, vector(4) /* blockIdx.y threadIdx.y */
     48, #pragma acc loop gang, vector(32) /* blockIdx.x threadIdx.x */
```

-Minfo=accel :
diagnostics

-ta=tesla:cc60 :
architecture Pascal

-ta=tesla:managed :
allocations
dynamiques en
mémoire unifiée

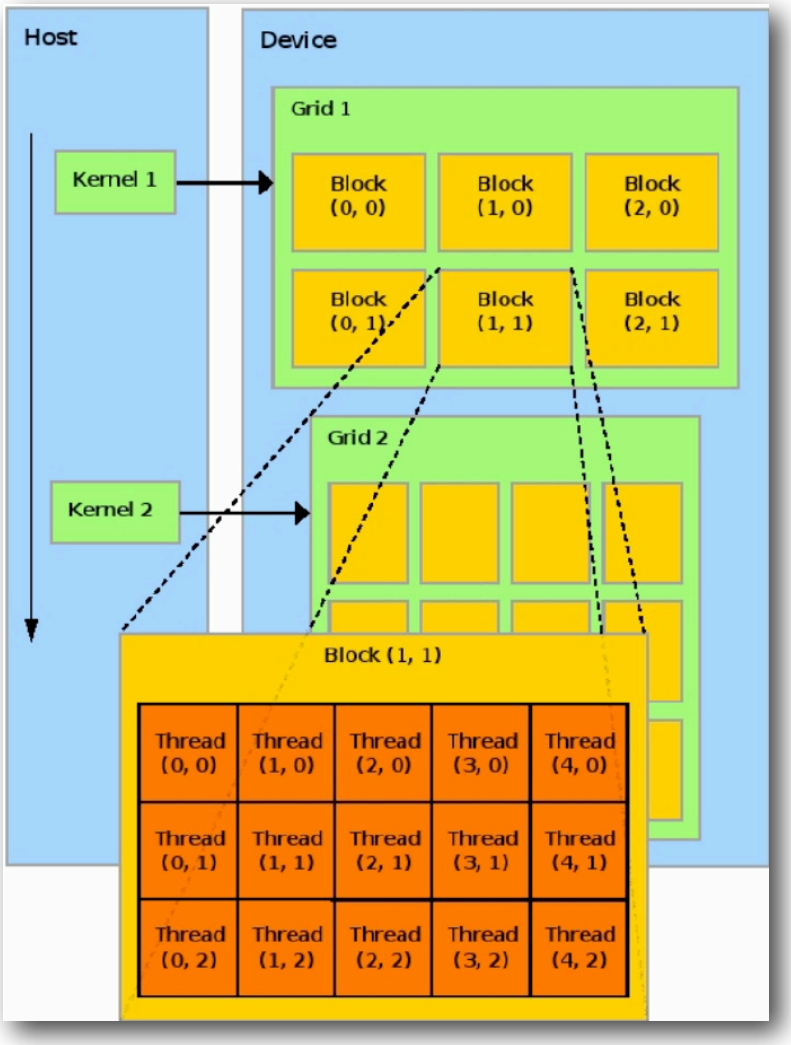
Code CUDA

A chaque itération en temps

1. Kernel de calcul (EDP)
2. Kernel d'application des conditions aux bords (plutôt que traitement sur CPU)
   - peu d'opérations, mais ***évite le coût d'un transfert de données*** **CPU -> GPU par le bus PCIe**
3. **Kernel *scalaire*** (un bloc, un thread) d'application des conditions aux coins (plutôt que traitement sur CPU)
   - ***évitant le coût d'un transfert de données*** **CPU -> GPU par le bus PCIe**
4. Synchronisation (*cudaDeviceSynchronize*)
5. Mise à jour des inconnues (=> second membre de l'EDP à l'itération suivante)
   - échange de pointeurs (alloués en mémoire unifiée)
6. Communications MPI

Grille 2D de blocs de threads
Nombre de threads en x et y : paramètres
Nombre de blocs en x et y : déduits

«Kernel» = code déporté sur GPU



Validation numérique

Précision

- Critère de comparaison de versions
  - écart maximal, sur l'ensemble des pixels, de la fonction d'encodage de fissure
  - fonction écrite en fichier (MPI-IO) après la boucle en temps

- Code CUDA sur K80
  - Résultats numériques strictement indépendants du nombre de GPU(s), i.e. de tâche(s) MPI

- Code CUDA sur P100
  - Résultats numériques strictement indépendants du nombre de GPU(s), i.e. de tâche(s) MPI

- Code CUDA sur K80 vs code CPU
  - Ecart maximal < 7.E-16

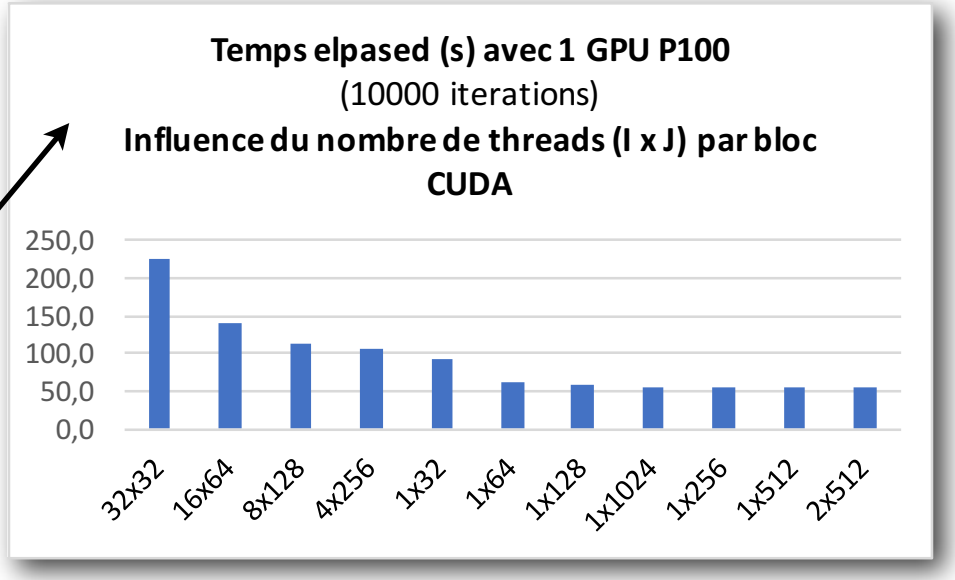
- Code CUDA sur P100 vs code CPU
  - Ecart maximal < 7.E-16

Meilleurs paramètres

Blocs de threads et domaines MPI

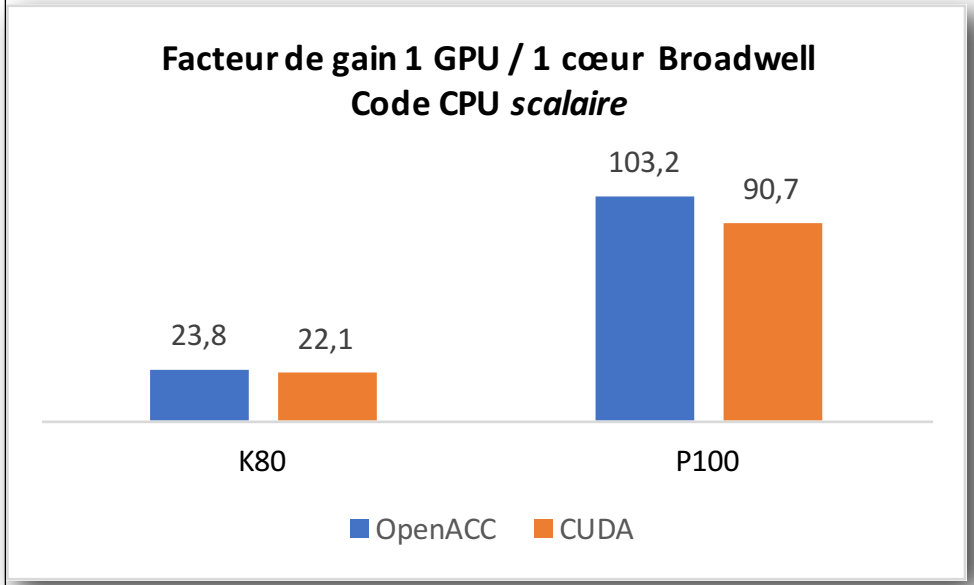
- Nombre de threads en I et J par bloc CUDA
  - Les accès à la mémoire globale d'un GPU sont plus rapides avec beaucoup plus de threads en J qu'en I

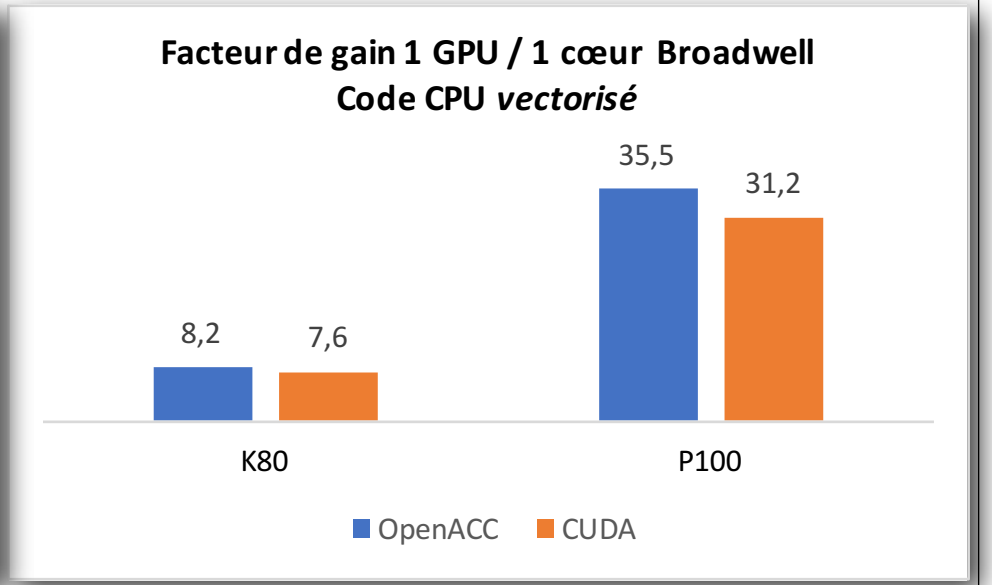
- Décomposition en domaines MPI
  - Grande lenteur de transfert CPU <-> GPU pour des sous-ensembles de tableau discontigus en mémoire
    - (1 domaine en I) x (2 domaines en J), 2 GPUs P100 : 10 itérations en 15 secondes
    - (2 domaines en I) x (1 domaine en J), 2 GPUs P100 : 500 itérations en 2 secondes
  - => Décomposition 1D dans la direction I (1 seul domaine en J)



Performance mono-GPU

OpenACC et CUDA

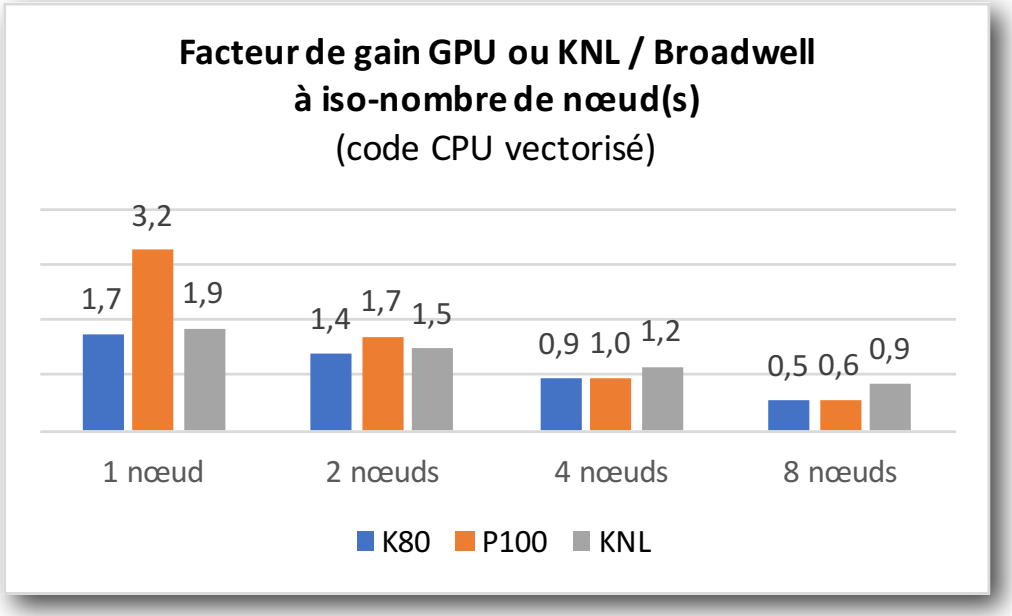


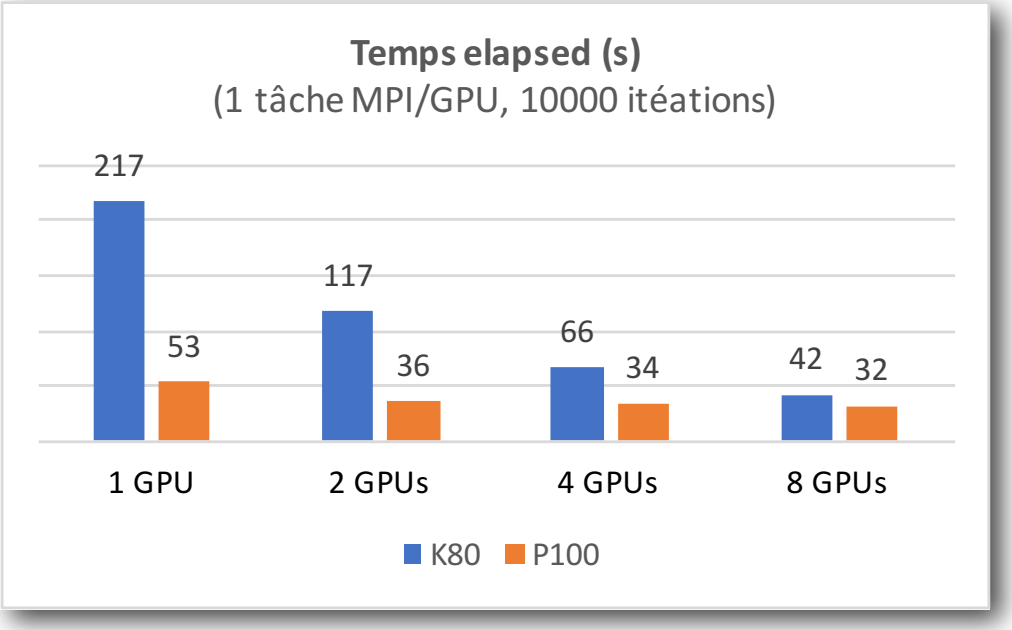


Performance multi-GPUs

CUDA + MPI

- L'usage de 2 GPUs P100 permet de concentrer une puissance importante dans un seul serveur
- Un K80 étant 4 fois plus lent qu'un P100 pour l'application en question, les transferts CPU <-> GPU (nécessaires aux échanges MPI) pèsent nettement moins sur l'accélération multi-GPUs K80
- Pour améliorer l'accélération multi-GPUs avec des GPUs Pascal ou Volta
  - Programmation : recouvrir les transferts de données et les calculs
  - Ou technologie
    - Chemin GPU-GPU plus direct (GPUDirect)
    - Lien rapide NVLink (GPU-GPU et/ou GPU-CPU)





Centre Régional Informatique et d'Applications Numériques de Normandie