Portage d'AVBP sur Xeon Phi

Cyril Fournier
CERFACS, ENSEIRB-MATMECA

Exécution de code sur Xeon Phi

- Modèle Offload : application exécutée sur l'hôte, envoie du code et des données sur le coprocesseur

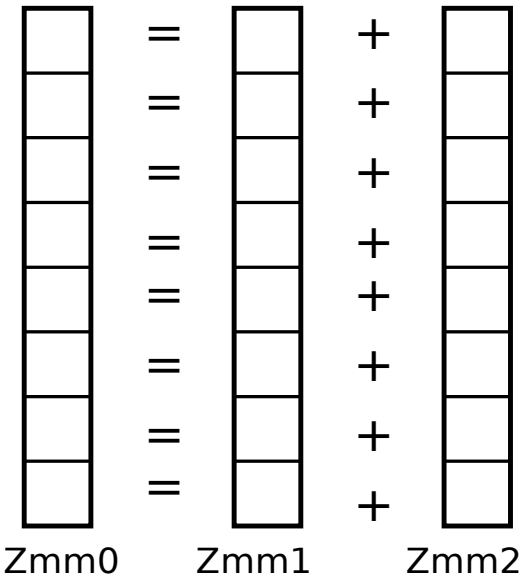
- Modèle natif : exécution directement depuis le coprocesseur

Coprocesseur Xeon Phi

- 1,01 GHz
- 8 GB GDDR5
- 61 core
- 4 threads / core
- Ordonnancement : 1 thread différent à chaque cycle d'horloge

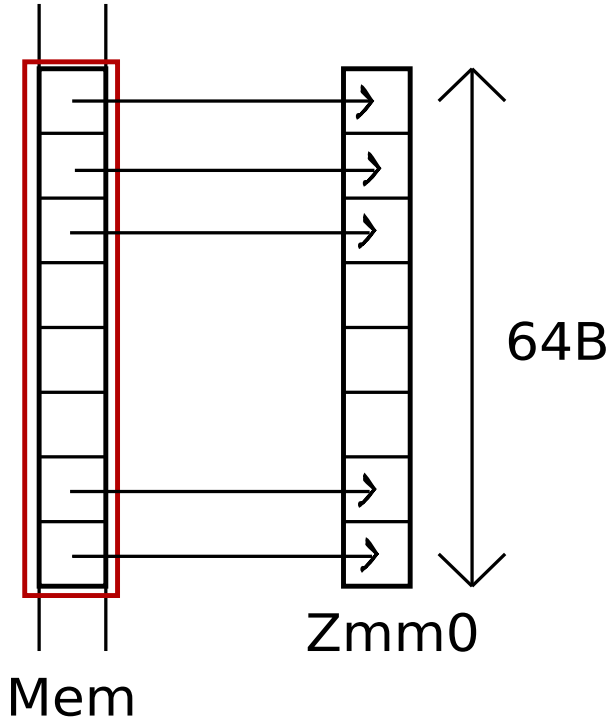
Instructions Vectorielles

- Vecteur 8 doubles, 16 floats (64 B)
- Instruction SIMD
- Latence 4 cycles



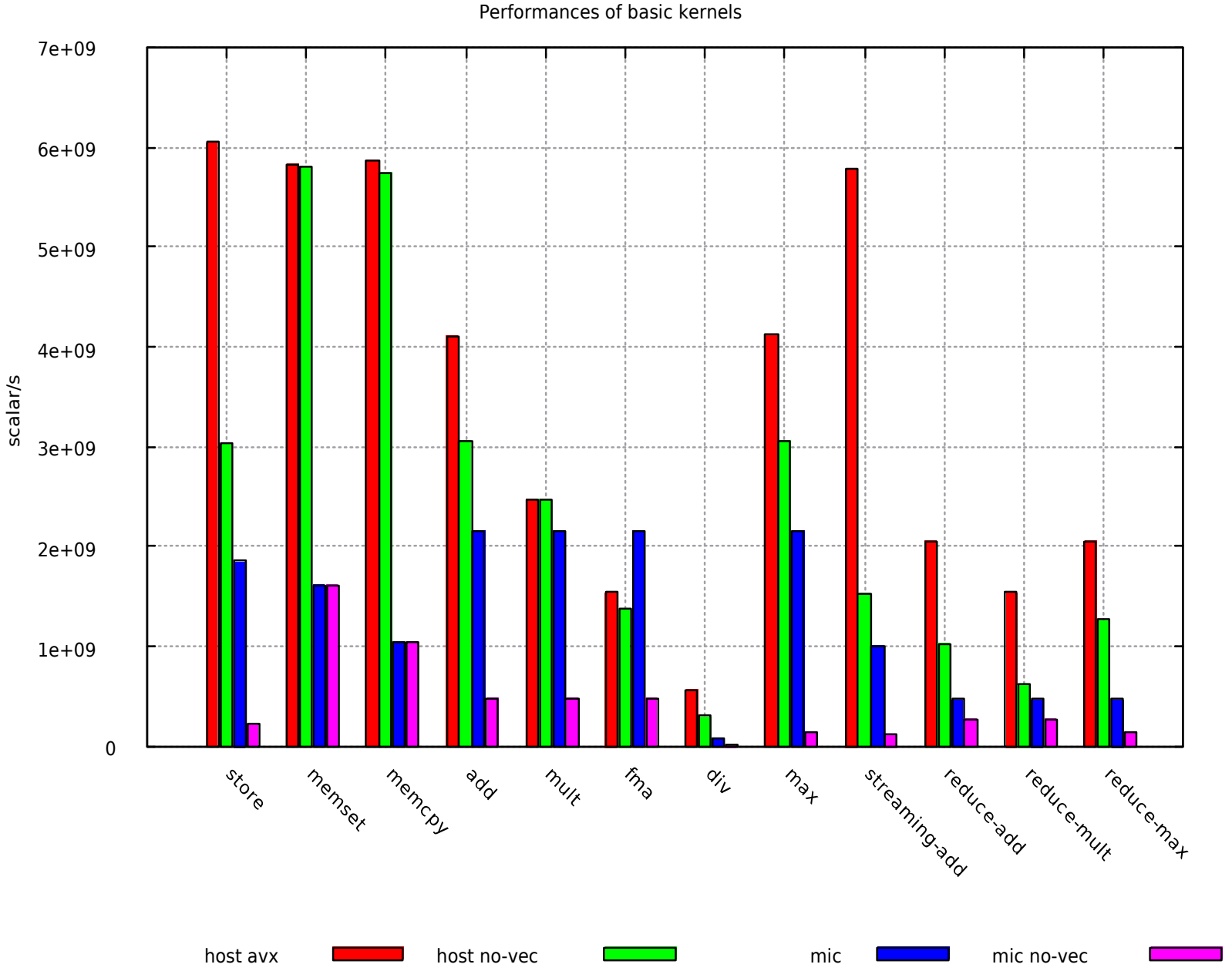
Alignement

- Alignement des données : chargement efficaces des registres
- Alignement sur 64 B



Benchmark

- Performance théorique : 1,07 Tflop/s (double)
- Benchmarks sur instructions vectorielles double précision : instructions simples, masquées, indirections, tableaux de taille fixe, variable...
- Tests en séquentiel (1 thread)


Performances of basic kernels
scalar/s
7e+09
6e+09
5e+09
4e+09
3e+09
2e+09
1e+09
0
store
memset
memcpy
add
mult
fma
div
max
streaming-add
reduce-add
reduce-mult
reduce-max
host avx
host no-vec
mic
mic no-vec

Benchmark : résultats

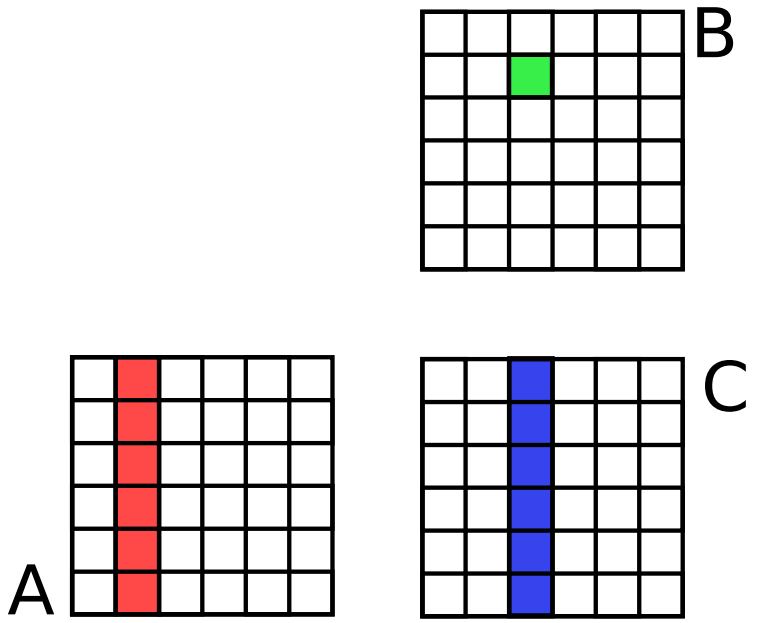
- Xeon Phi 2 à 6 fois plus lent (1thread, tableau 1D taille fixe)
- Dégradation rapide des performances hors conditions idéales :
  - Masque constant : CPU -25 %, Phi -80 %
  - Tableau 2D taille variable : CPU 10x plus lent, Phi 30x plus lent
- Performances envisageables :
  - 1D taille fixe : Phi 200Gflop/s (122 threads), CPU 50Gflops/s (8 threads)
  - 2D taille variable : Phi 7GFlop/s (122 threads), CPU 5Gflop/s (8 threads)

Vectorisation d'un kernel

- Ensemble de 4 routines
- Calcul matriciel sur 3 routines
- Calcul de résidus dans la 4ème routine
- Données : tableaux de matrices 7x7 ou 8x8

Vectorisation d'un kernel

- Optimisations : padding pour matrice 7x7, codage en dur des dimensions et des compteurs de boucle, réécriture calcul matriciel



C[:,j] = sum k (A[:,k] * B[k,j])

Vectorisation : Résultat

- Speed up séquentiels :
  - Phi : 5 – 7, CPU : 2 – 5 pour calcul matriciel
  - 1,5 pour calcul de résidus
  - Phi : 2,7 – 3,6, CPU : 1,8 – 2,6 global
  - Phi 8,5 à 12 fois plus lent que CPU
- Speed up envisageable Phi / CPU : 1,27
- Parallélisation OpenMP : 10 % de gain mesuré

Parallélisation OpenMP

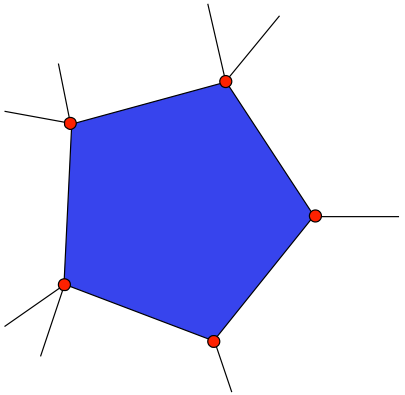
- Comparaison OpenMP / MPI sur 122 threads
- Test sans communication :
  - OMP PARALLEL DO : 3x plus lent que MPI
  - OMP DO nowait : 2x plus lent que MPI
  - Distribution manuelle des données : 1,2x plus rapide que MPI
- Comparaison avec MPI BARRIER : 22x plus lent que MPI sans barrière

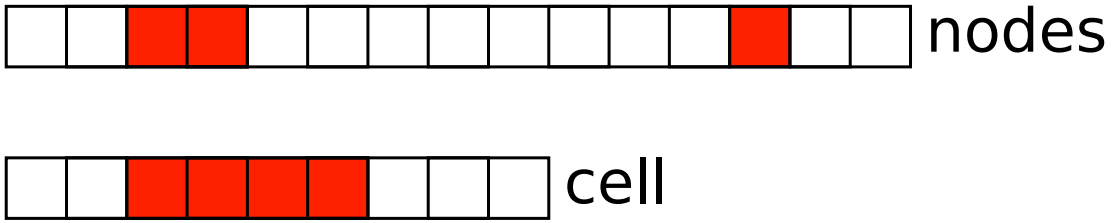
Parallélisation OpenMP

- Comparaison OpenMP / MPI sur 122 threads
- Operation de Réduction :
  - MPI_Allreduce
  - OMP DO reduction
- MPI 30x plus lent que OpenMP

Ajout d'OpenMP dans AVBP

- MPI
- Données d'AVBP : tableaux de noeuds ou de cellules.
- Cellule = collection de vertices

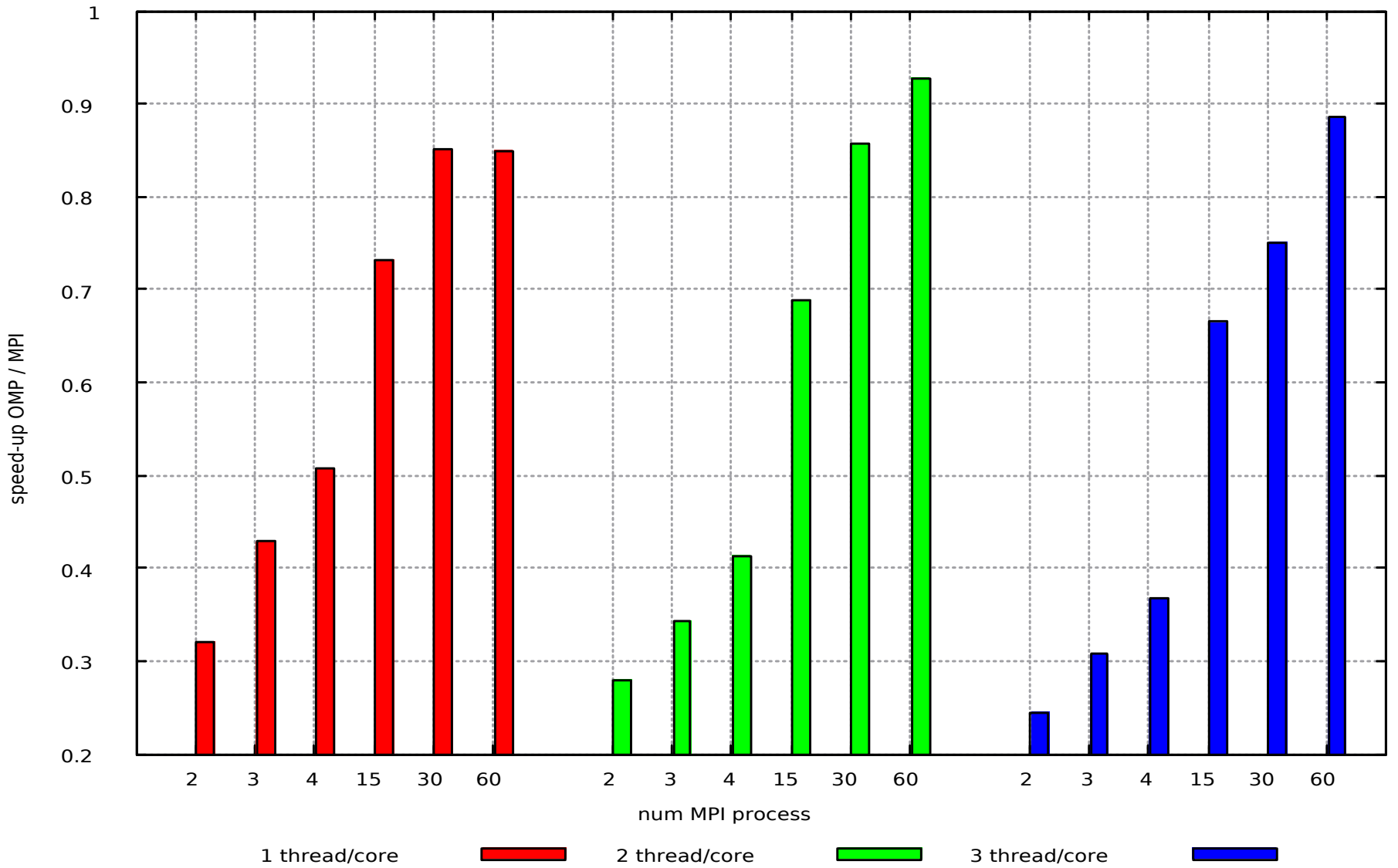




Ajout d'OpenMP dans AVBP

- Parallélisation du cœur de calculs
- 1 seule région parallèle, élimination des barrières
- Barrières restantes : autour des communications MPI, avant création des tableaux de cellules, avant mise à jour des nœuds à partir des cellules

Ajout d'OpenMP : résultats



Conclusion

- Vectorisation :
  - Données contigües, alignées
  - Opérations sur tableaux linéaires
- Parallelisation :
  - Élimination des synchronisations

Questions
?