Problématique des accès mémoires irréguliers causés par les maillages non structurés :

étude comparative entre les machines massivement multicoeurs et les GPU

Loïc Maréchal / INRIA

LJLL, Demi-Journée Calcul Scientifique, 19/12/2013

Architectures actuelles

- Mémoire distribuée : clusters, supercalculateurs
- Mémoire partagée : puce multicœur
- Mémoire partagée distribuée : machine contenant plusieurs puces multicœurs
- GPU : une mémoire partagée ?

Caractérisation des accès mémoires

- Tous permettent de lire et écrire efficacement de manière linéaire. Les architectures se différencient lors des accès aléatoires, inévitables quand les données sont des maillages non structurés.

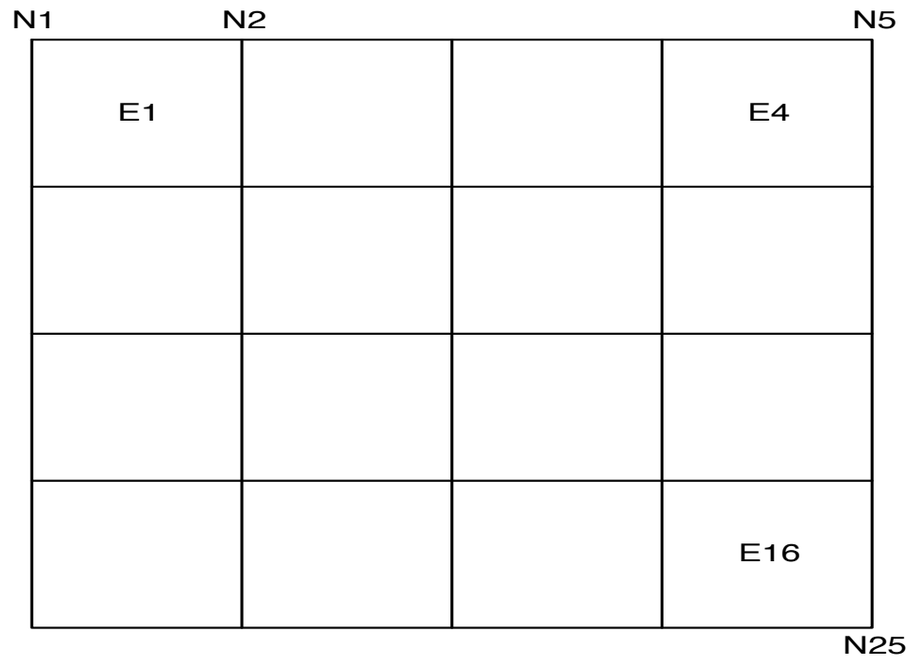
- Mémoire distribuée : lente en lecture et écriture

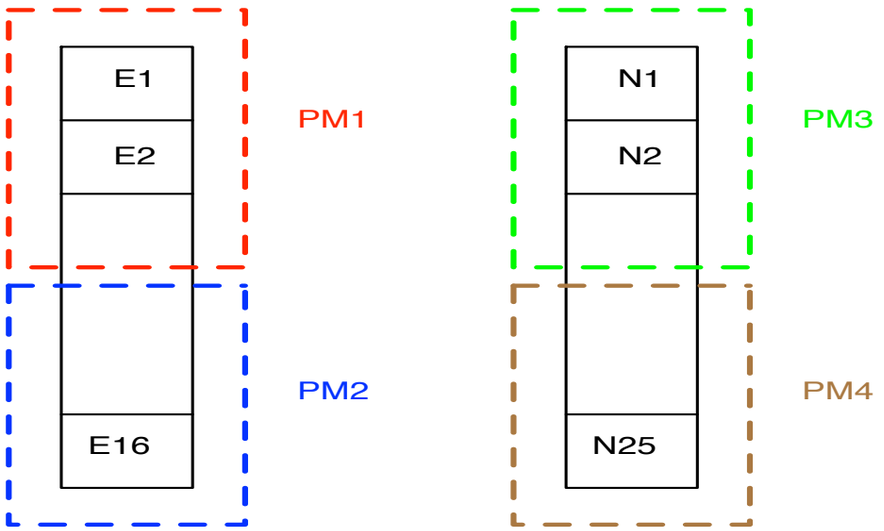
- Mémoire partagée distribuée : rapide en lecture, mais lente en écriture

- Mémoire partagée : rapide en lecture et écriture

Problème type d'accès indirect dans un maillage

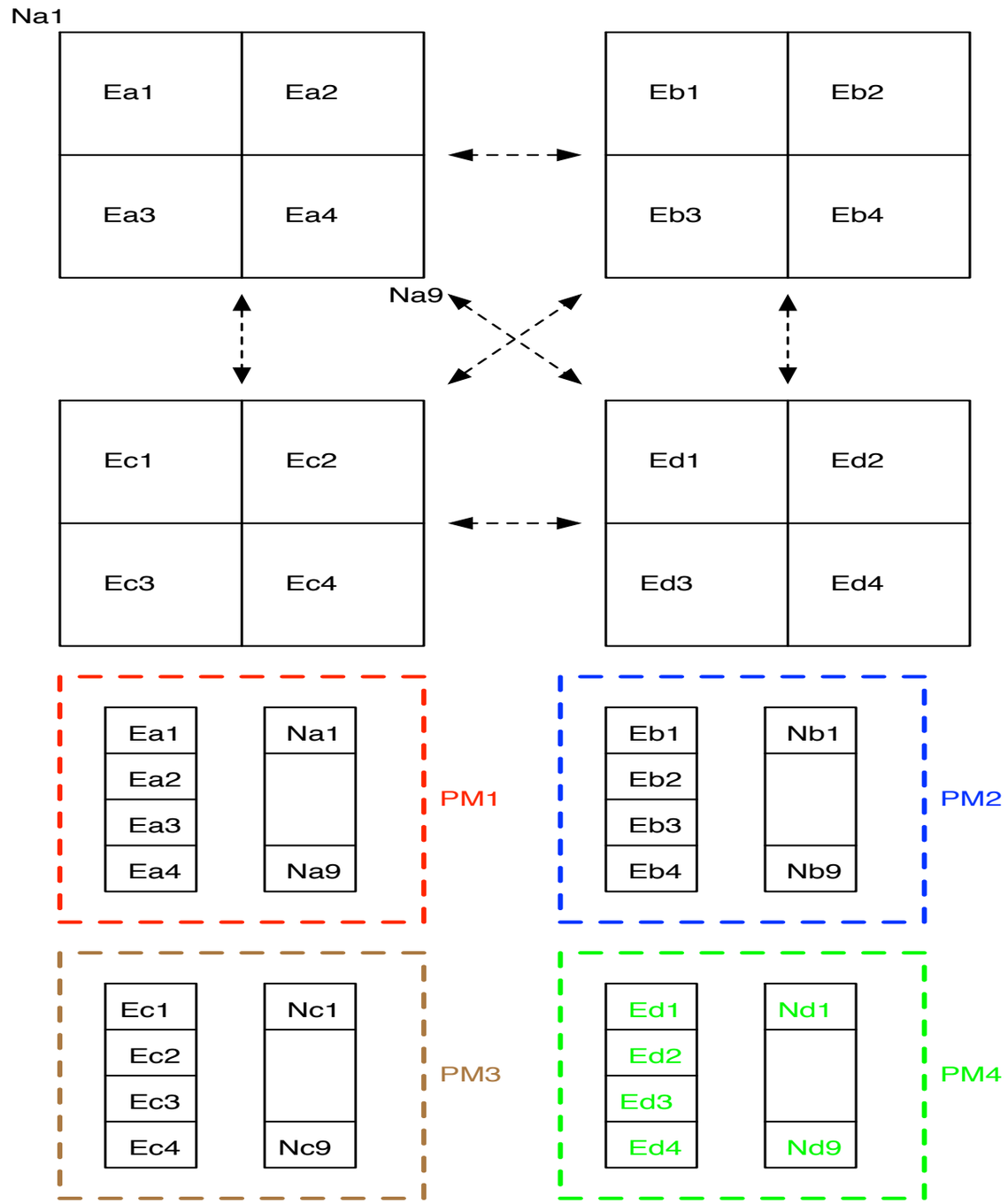
- Boucle sur les éléments : ajout de la température de l'élément à celle de ses nœuds.
- Accès en lecture linéaire sur le tableau des éléments.
- Accès en écriture indirecte sur le tableau des nœuds.
- Mémoires distribuées ou partagées-distribuées : lent.
- Mémoire partagée par une seule puce : rapide.





Le partitionnement de domaines

- Les éléments et nœuds de chaque sous-maillage tiennent entièrement dans une seule région mémoire.
- Lors de l'écriture des températures aux nœuds, il n'y a plus d'accès à une région mémoire différente de celle des éléments.
- Nécessite une renumérotation des nœuds et éléments ainsi que des échanges explicites d'informations entre les sous-maillages : modifications lourdes des codes.
- Efficace sur toutes les architectures mémoires !



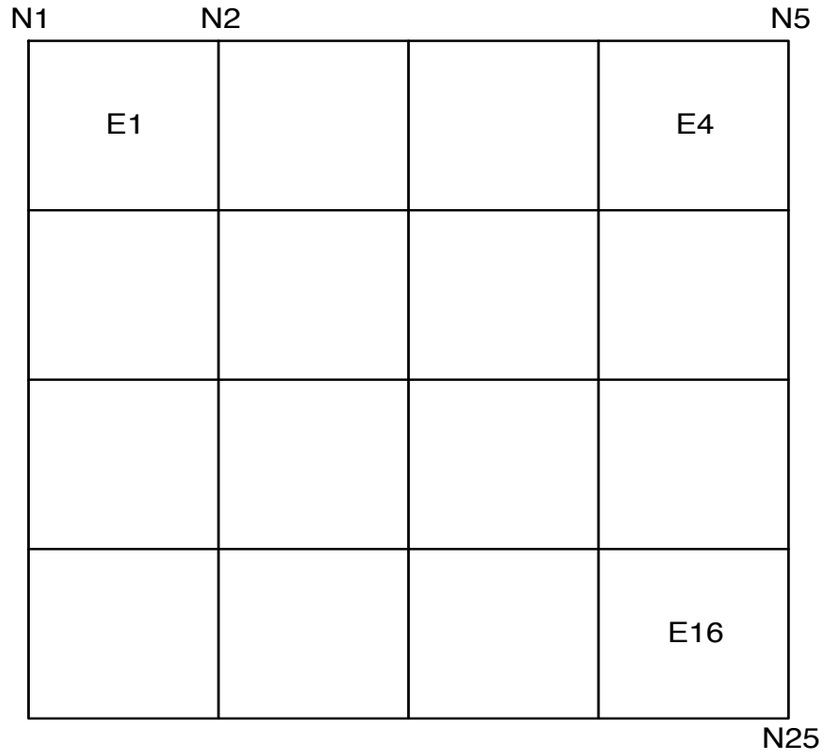
Entrelacement de tableaux et paires „scatter/gather“

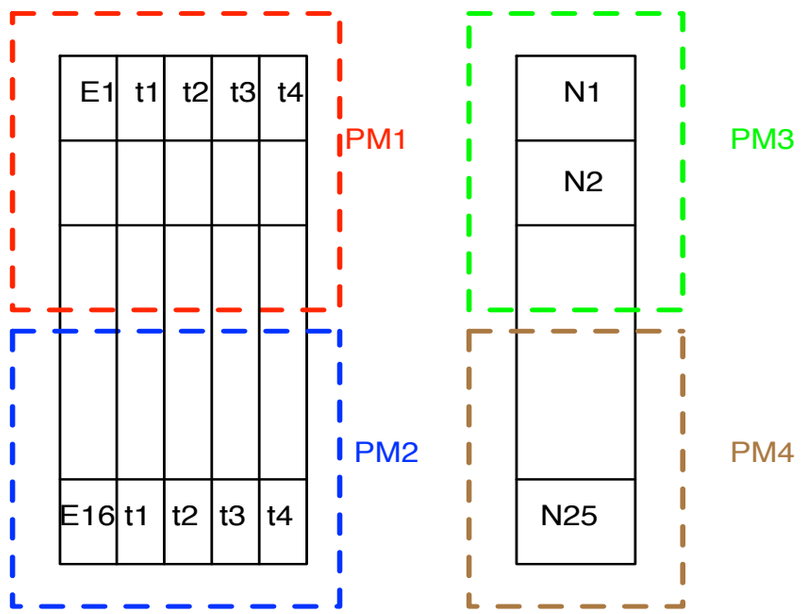
- Le tableau d’élément contient aussi une copie de la température de chacun de ses nœuds et le processus est alors découpé en deux boucles :

- Boucle sur les éléments (scatter) : après calcul de la contribution de l’élément à chacun de ses quatre nœuds, celle-ci est stockée localement et non directement ajoutée au tableau des températures associées aux nœuds (conflit d’écriture). On a donc des lectures distantes, mais des écritures locales.

- Boucle sur les nœuds (gather) : chaque nœud va sommer les contributions locales des éléments de sa boule. Ici aussi l’accès aux régions mémoires distantes (éléments) se fait en lecture seule et les écritures (valeur aux nœuds) se font localement.

- Rapide pour les mémoires partagées et un peu moins pour les mémoires partagées distribuées mais lent pour les mémoires distribuées.





Étude de cas : SGI UV

- Machine HPC1 du LJLL : une machine à 160 cœurs ou bien 10 nœuds de 16 cœurs.

- Vue comme une machine à 160 cœurs : il faut entrelacer la mémoire pour limiter le goulot d'étranglement (numactl —interleave=all), malgré cela, l'accélération stagne à x10 à cause des écritures distantes. Lorsqu'un processeur doit écrire dans la mémoire d'un autre, il lui „prend“ sa ligne de cache afin d'y écrire ses données, créant un chamboulement complet de la contiguïté de la mémoire, très dommageable aux performances.

- Couple scatter / gather, meilleure accélération, mais il faut tout de même payer le coût des lectures mémoires distantes. Numactl -H : 100 ns pour un accès local, 560 ns pour un accès distant, la moyenne est environ à 400 ns. Un facteur de pénalité de quatre est observé, celui-ci est constant et ne nuit pas à l'accélération (*160 / 4 = *40).

- Conclusion : Plus au augmente le nombre de puces et plus on augmente la puissance, mais plus on ralentit l'accès mémoire, annulant quasiment tous les gains ! Avec un très grand nombre de nœuds (>10), on fini par gagner.

Étude de cas : GPU Nvidia TESLA

- Toutes les unités de calculs d'une carte graphique partagent la même mémoire.

- Mais l'écriture simultanée de plusieurs unités dans des régions mémoires voisines nécessite de coûteuses synchronisations. L'utilisation de techniques de coloriage mémoire comme dans le cas de la parallélisation multhithreads fonctionne, mais est très inefficace, car elle accède à la mémoire de manière non linéaire.

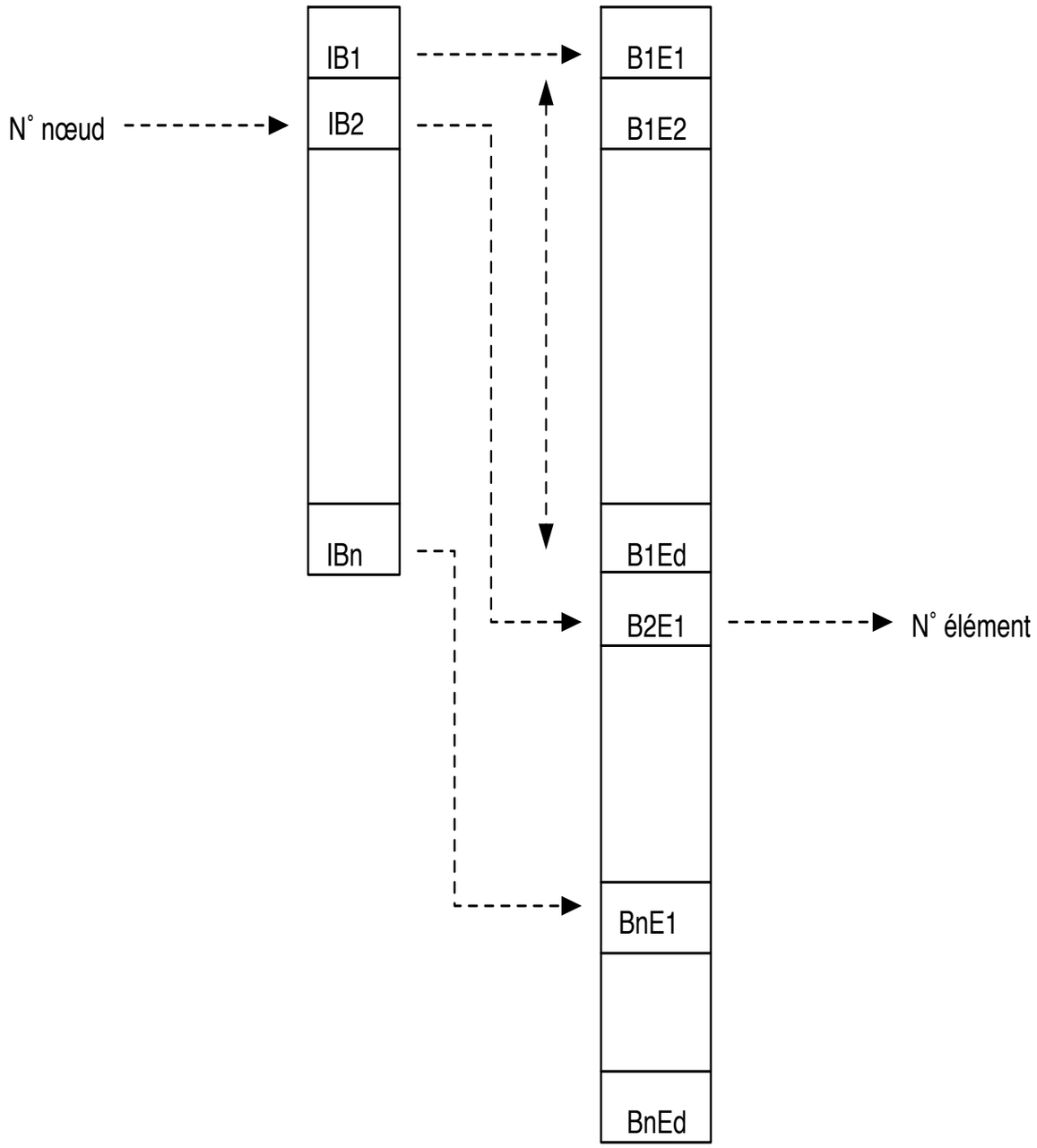
- De fait, un GPU est à considérer comme une machine à mémoire partagée distribuée. Il est nécessaire d'utiliser un mode scatter/gather afin d'en tirer parti.

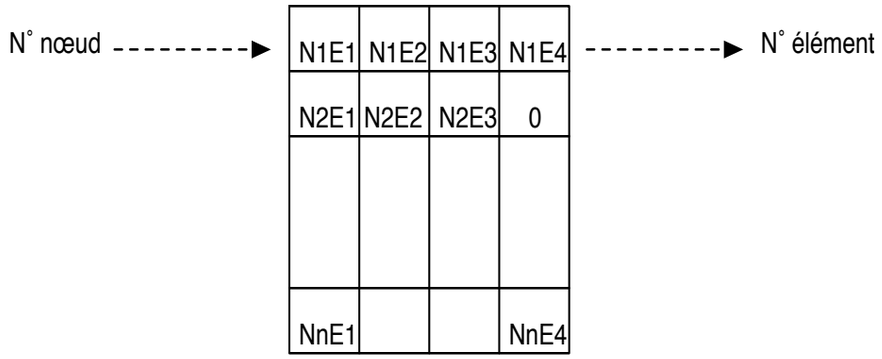
- Contrainte supplémentaire : la double indirection mémoire en lecture. C'est le cas par exemple lorsque l'on souhaite accéder à la liste des éléments partageant un même sommet dans un maillage non structuré (appelé la boule du point).

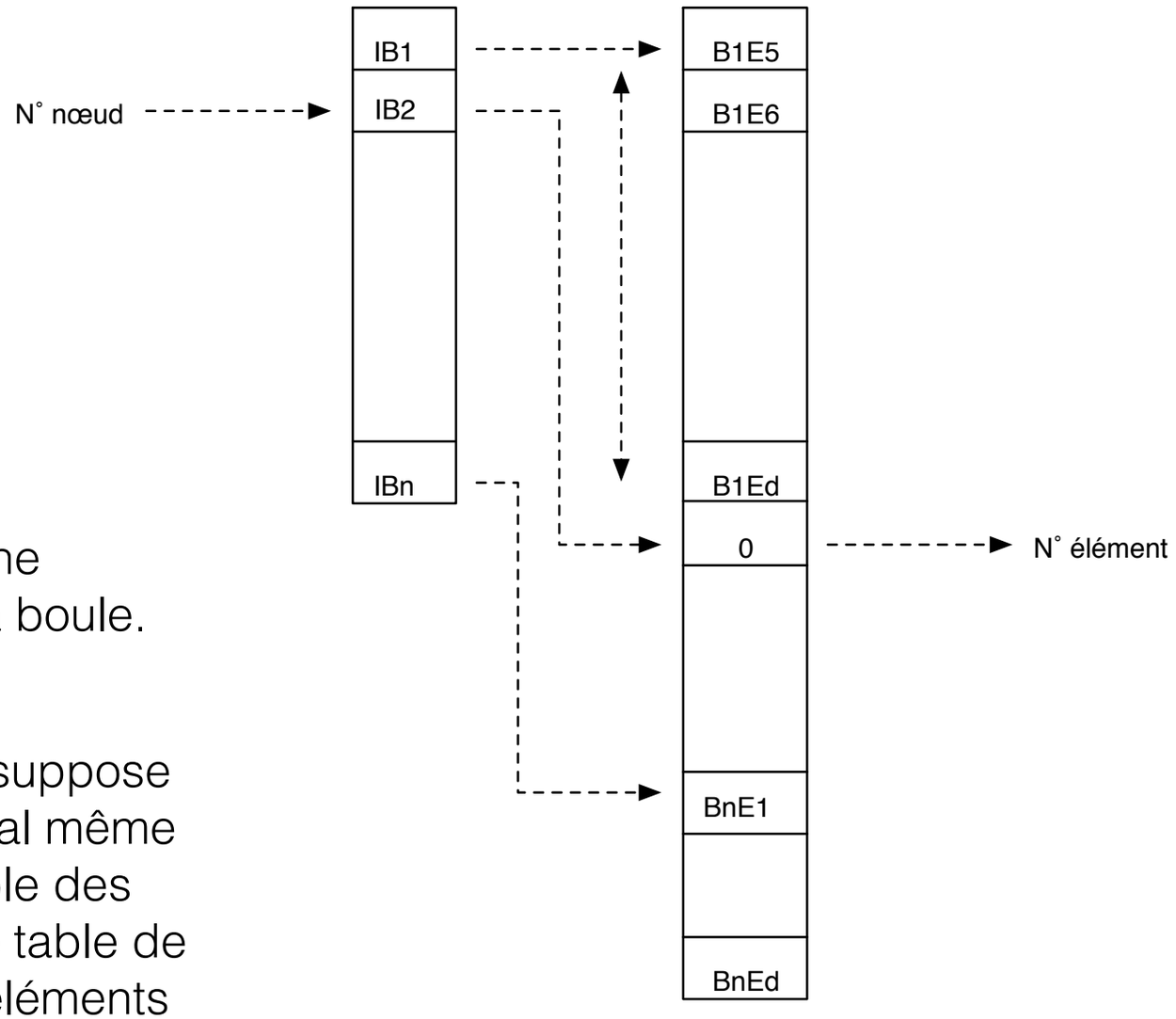
- Chaque indirection mémoire casse la vectorisation du code et insère de très longs temps d'attentes. Ils peuvent être cachés par des calculs, mais l'expérience montre que si un seul niveau d'indirection est tolérable, deux niveaux ralentissent très sensiblement les calculs.

Cassage des doubles indirection : la vectorisation des boules de points

- La table IB donne pour chaque nœud un index dans la table des boules.
- La table des boules donne à partir d'un index, la liste des éléments partageant ce nœud. Dans un maillage non structuré, leur nombre, appelé degré, est variable.
- Dans le cas de maillages structurés, par exemple une grille bidimensionnelle, le degré est constant (4), une seule table est alors nécessaire et fournit directement les 4 éléments pour chaque nœud. Ce type de donnée est souvent utilisé dans les „démos “ pour mettre en avant la vitesse des GPU.
- Il est possible de rendre efficaces les doubles indirections sur GPU par une approche hybride, structurée avec débordement.







- Pour chaque nœud, la table de base donne directement les 4 premiers éléments de la boule.

- Si le degré est supérieur à 4 (ce que l'on suppose rare dans le cas d'un maillage quadrilatéral même non structuré), il faut alors consulter la table des débordements qui fonctionne comme une table de boules standard, mais ne fournit que les éléments au-delà de 4.

- Dans la pratique on choisira une taille de vecteur de base de telle sorte que le nombre d'éléments de débordement soit minimal. Cette taille doit être une puissance de 2 pour être efficace sur GPU.

La GMLIB2, pour un stockage efficace des maillages non structurés sur GPU

- Offre les types de données de maillages usuels : Vertices, Edges, Triangles, Quads, Tetrahedra, Hexahedra ainsi que des champs de solutions associés.

- Gère l'allocation et le transfert entre le CPU et le CPU.

- Génère automatiquement les boules de points vectorisées.

- Testée avec succès sur Hexotic-smoothing (x24) et Wolf (x36) sur une carte Nvidia Tesla par rapport à un core de Xeon X5570 à 2,93 GHz.

- Téléchargeable librement sur www.rocq.inria.fr/gamma/gamma/Membres/CIPD/Loic.Marechal/Research/GM2.html

éléments	vecteur	table de base	table d'extension	surcoût mémoire
arêtes	16	98,72 %	1,28 %	34,60 %
triangles	8	99,98 %	0,02 %	33,35 %
quadrilatères	4	99,99 %	0,01 %	0,02 %
tétraèdres	32	99,71 %	0,29 %	46,75 %
hexaèdres	8	96,87 %	3,13 %	9,79 %

Étude de cas : Xeon PHI, une puce multicœurs

- 60 cœurs, 4 threads par cœur, mémoire partagée à forte latence, mais cachée par l'hyperthreading. L'accélération est bonne (x53 Wolf, x36 pour Hexotic-smoothing) mais la vitesse de chaque cœur est très faible : Pentium original à 1 GHz environ 15 fois plus lent qu'un Core i7 actuel.

- Pour vraiment tirer parti de cette architecture, il faut utiliser les unités vectorielles AVX512 travaillant sur 8 doubles ou 16 floats. Problème : cette architecture est optimisée pour les structures de tableaux et non les tableaux de structures. Elle est efficace pour les anciens codes vectoriels (Cray) mais pas pour les logiciels „multithreadés".

- Nécessite une simple recompilation avec ICC et l'option „-mmic". Il suffit de transférer l'exécutable et les données sur la carte avec la commande „scp", puis de se logguer sur le GPU avec un „ssh".

- Une version avec plus de cœurs, et une unité vectorielle SIMD ou bien utilisant des cœurs un peu plus puissants serait bienvenue.

Étude de cas : Xeon PHI, un GPU

- Cette carte peut être considérée comme une carte „graphique“ standard et être utilisée via des kernels en OpenCL. Elle est compatible avec la GMLIB2 mais la gestion de l’OpenCL par Intel est encore en version „beta“.

- Connexion sur la machine hôte (phi), et lancement de Wolf ou Hexotic de manière habituelle. Le compilateur génère du code X86 du côté hôte et sur la carte, un démon attend les requêtes de transfert de données ou de code et récupère les commandes à exécuter.

- Problèmes lors du transfert de données de petite taille.

- Faible parallélisation, le moteur OpenCL de la carte n’utilise que peu de cœurs simultanément dès qu’une routine n’est pas triviale (il suffit d’utiliser la commande „top“ sur le Xeon PHI ce qui est pratique). C’est un défaut de jeunesse déjà rencontré dans les premières implémentations d’AMD et Nvidia.

- Résultats encore plus décevants qu’en multithreads : 20 passes d’optimisation sur un maillage de 10 millions d’hexaèdres prennent 10mn 26s en série, 17s avec 120 threads et 30s en OpenCL, alors qu’une Tesla met 3s et un octo-core xeon X5570 met 8s.

Quelques comparaisons (durées en secondes)

	Wolf sur aile M6, ordre 1, 1 million tétras, 1000 iters	Hexotic optimisation de 10 millions d'hexas, 50 passes
xeon X5570 2,93 GHz 1c	2033	132
xeon X5570 2,93 GHz 8c	244	19
core i7 2,7 GHz 1c	1339	100
core i7 2,7 GHz 4c	327	29
Xeon X7550 2 GHz 1c	3179	238
Xeon X7550 2 GHz 16c	229	21
Xeon X7550 2 GHz 32c	292	26
Xeon X7550 2 GHz 48c	287	28
Xeon X7550 2 GHz 64c	243	26
core i7 2,7 ghz (OpenCL)		25
Radeon HD 5870 0,85 GHz 1600u	42	5
Quadro 6000 1,15 GHz 448u	56	7
Intel HD 4000 1,2 GHz 64u		29
GeForce 650m 0,95 GHz 384u	172	18
Xeon PHI 1,05 GHz 1c	20189	627
Xeon PHI 1,05 GHz 60c	387	42
Xeon PHI 1,05 GHz 60u		75

Perspectives d’évolutions des architectures

- CPU multicœurs : il est peu efficace d’augmenter le nombre de cœurs (16) sans augmenter le nombre de canaux mémoires (4). Futurs projets d’Intel à 18 cœurs, mais toujours 4 bus mémoires en DDR4 (plus de débit par canal). Y aura il un jour une puce à 8 canaux ?

- Multi-CPU multicœurs : la vitesse d’accès entre les puces ne suit pas l’évolution de la puissance de calcul de ces dernières. Plus au augmente le nombre de puces et plus on augmente la puissance, mais plus on ralentit l’accès mémoire. Tout dépend de la topologie du réseau mémoire.

- GPU : très bonne montée en puissance en fonction du nombre d’unités de calculs jusqu’à présent. Toujours limités par une petite mémoire même si les cartes à 4 GO ou plus se démocratisent un peu. Grande complexité de développement et mise au point.

- Xeon PHI : résultats préliminaires faibles, mais l’idée d’un grand nombre de cœurs de CPU „traditionnels“ couplés à de la mémoire vectorielle de type GPU a un bon potentiel. À suivre…