

## **Retour d'expérience Xeon PHI**

**Damien DUBUC**  
**Expert HPC software**

---

Février 2012

## Quel est notre rôle ?

---

Présenter

- Gérer un portefeuille clients nécessitant des optimisations calculatoire .
- Analyser et trouver les bottlenecks dans la partie calculs des logiciels

Proposer

- Revoir les algorithmes
- Trouver la technologie la plus adaptée
- Réorganiser la gestion de projet et la gestion de ressources

Concevoir

- Optimiser localement l'algorithme
- Changer d'algorithme pour le rendre parallèle
- Exploiter les performances de la technologie choisie

Convaincre

- Valider la précision numérique
- Valider l'optimisation
- Rendre une équipe interne autonome

## Utilisation d'accélérateurs matériels

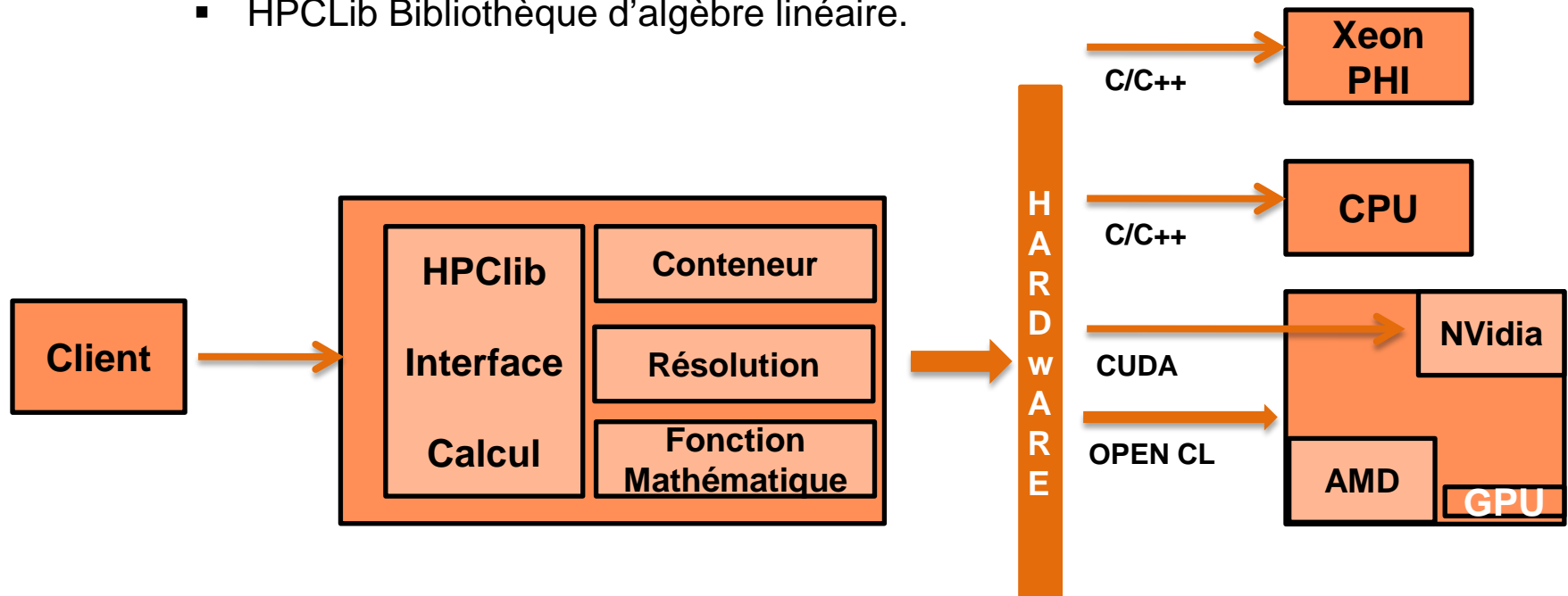
### Quels accélérateurs de calcul utiliser? GPU, Xeon Phi, ...

Intitulé	Client	Référents	Description
Document best practices	CACIB	<b>DD, WK</b> , BdP	Recommandation sur la conduite et la mise en œuvre d'un projet de développement de logiciels optimisés.
Portage de pricer sur GPU et Xeon PHI	BNP	<b>DD</b>	Parallélisation d'une bibliothèque sur GPU et sur MIC grâce à une couche de parallélisation multi-cible.
Partenariat	INTEL	<b>DD, WK</b>	Mise à disposition de machines de test en exclusivité pour évaluer les nouvelles architectures (dont Xeon Phi, sous NDA),
POC Xeon Phi (MIC)	CACIB	<b>DD</b>	Portage d'une bibliothèque sur INTEL Xeon Phi (résultats sous NDA d'INTEL),
Portage d'un pricer C# sur GPU	SGCIB	<b>WK, DD</b>	Parallélisation d'une bibliothèque sur GPU et étude de l'impact de l'adoption de cette technologie sur les processus métiers.
HPCLib, MTPS, Benchs	R&D	<b>DD, WK</b>	Mise au point d'outils optimisés sur GPU et sur MIC (Xeon Phi)

# Retour d'expérience sur le produit Xeon PHI

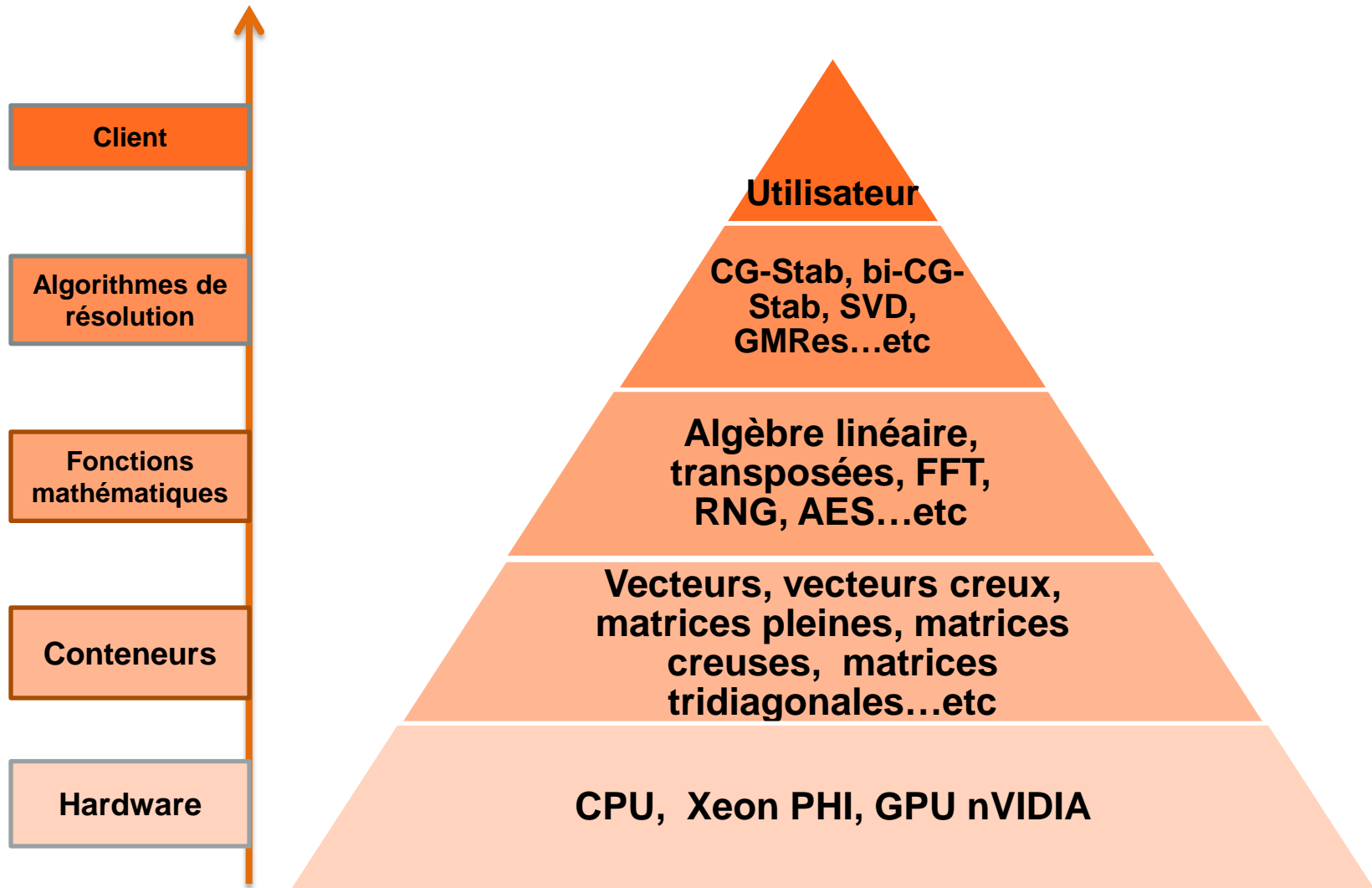
## Les projets

- ❑ 5 expériences Xeon PHI pour nos clients
  - 1 projet de modélisation financière et de calibration.
  - 2 projets de calcul de risques financier.
  - 1 projet de simulations physique.
  - 1 projet d'automatisation.
- ❑ 1 expérience au siège d'ANEO
  - HPCLib Bibliothèque d'algèbre linéaire.



# Retour d'expérience sur le produit Xeon PHI

## Les projets

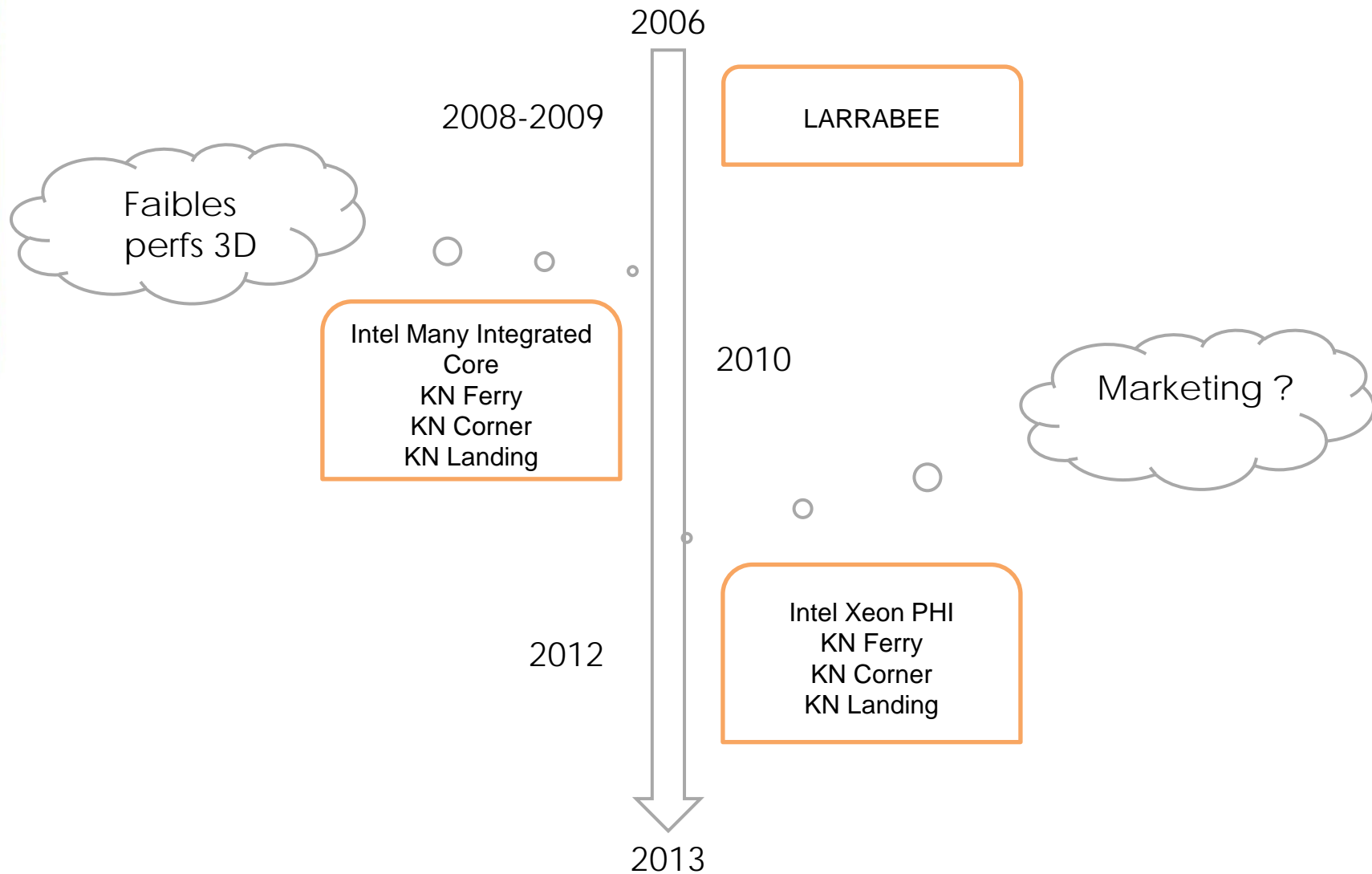


# Retour d'expérience sur le produit Xeon PHI

---

## Présentation du produit

# Présentation du produit Intel *Historique*



# Présentation du produit Intel

## Les caractéristiques

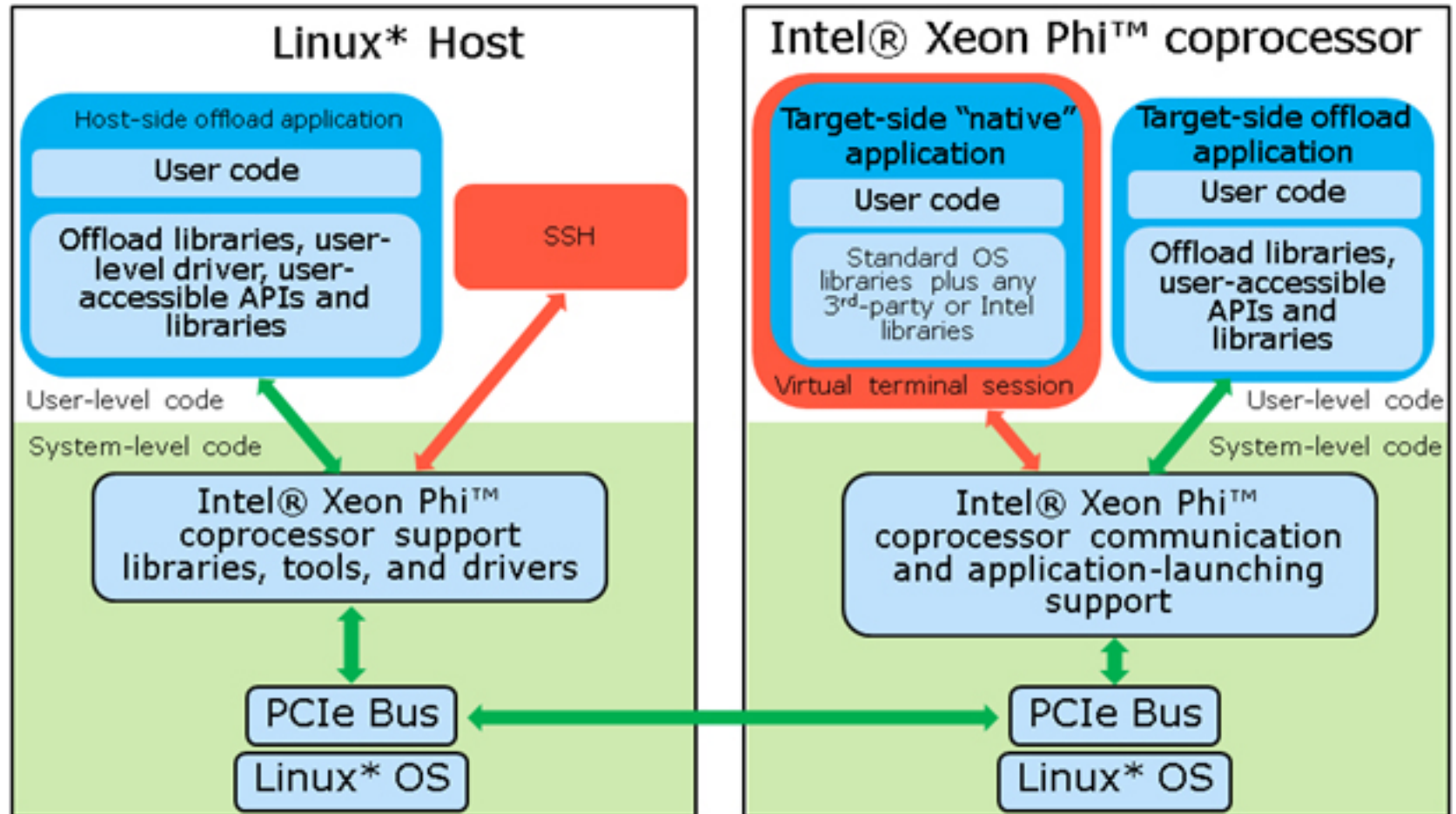
Intitulé	Caractéristique
<b>Processeur</b>	60 cores 1.053 GHz / 240 Threads
<b>Gflops en crête</b>	Jusqu'à 1 TeraFlops
<b>Mémoire vive</b>	8 Go avec une bande-passante de 320 GB/s
<b>Slot du KNC</b>	PCIe x16 Gen 2
<b>Cache mémoire</b>	32 KB de L1 et 512 KB L2 (per core)
<b>Système d'exploitation</b>	Linux avec IP addressable
<b>Instructions X86</b>	512 bits
<b>TDP</b>	225 W
<b>Host OS</b>	Red Hat Entreprise Linux 6.x et Suse Linux 12+






# Retour d'expérience sur le produit Xeon Phi

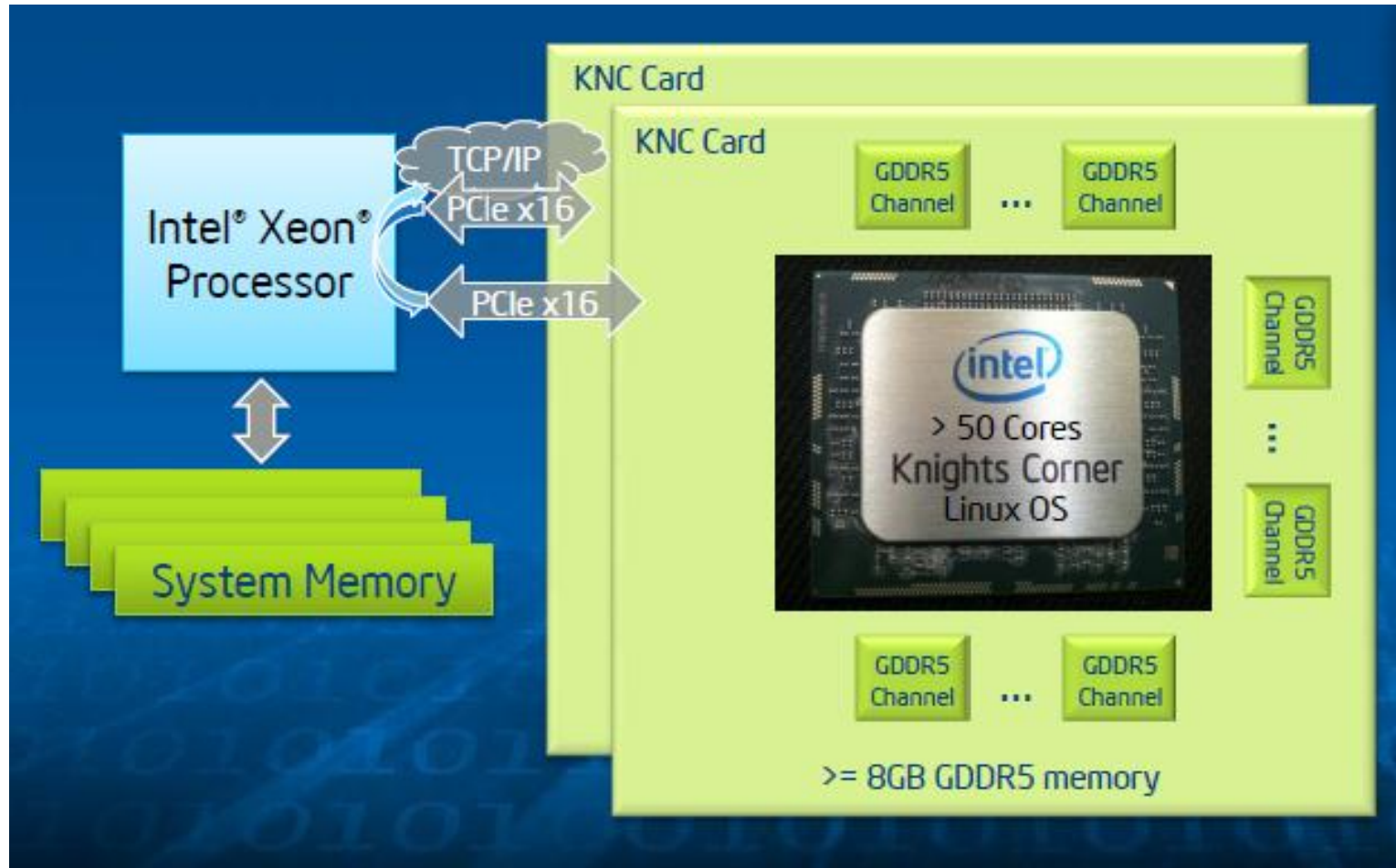
## Architecture du Xeon Phi



 Pas de support Windows. (Beta version prévu Q2-Q3 2013)

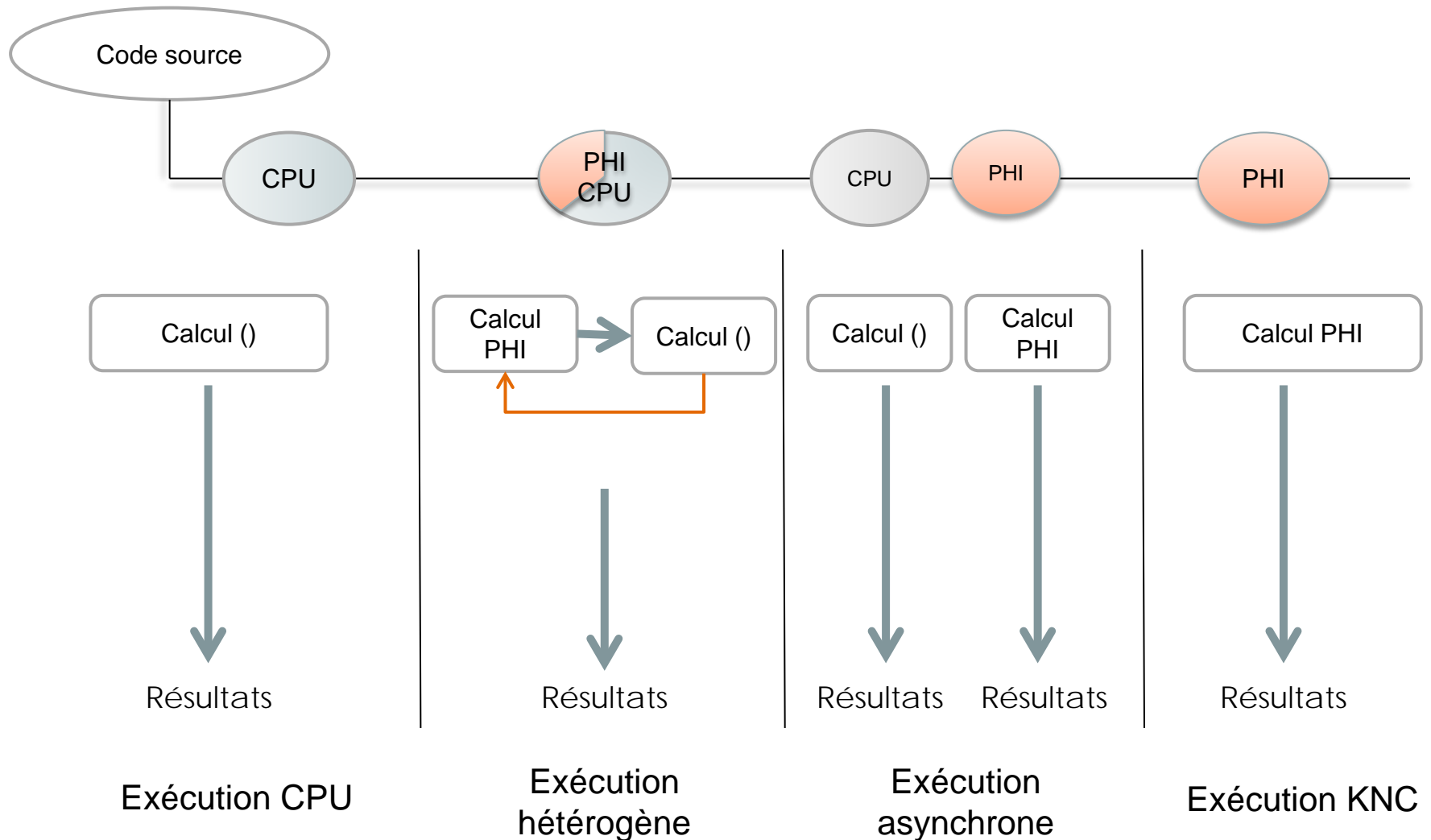
# Retour d'expérience sur le produit Xeon PHI

## Architecture du Xeon PHI



# Retour d'expérience sur le produit Xeon PHI

## Concept de programmation



# Retour d'expérience sur le produit Xeon PHI

## Concept de programmation

- Le code source reste en C++ / Fortran
- Ajout d'un langage Intel offload directive (LEO)

```
int size = 45 * 1024 * 1024;
float *a = (float *)posix_memalign(size *
sizeof(float), 64);

#pragma offload target(mic:0) inout(a:
length(size) alloc_if(1) free_if(1))
{
    ComputePartTime(a);
}

printf("Results %f\n", a[100]);
```

```
int size = 45 * 1024 * 1024;
float *a = (float *)posix_memalign(size *
sizeof(float), 64);

float * a_d;

cudaMalloc(&a_d, size * sizeof(float));
cudaMemcpy(a_d, a, size * sizeof(float),
           cudaMemcpyHostToDevice);
dim3 grid = dim3(1,1,32);
dim3 block = dim3(1,1,256);

ComputePartTime<<<grid, block>>> (a_d);

cudaDeviceSynchronize();
cudaMemcpy(a, a_d, size * sizeof(float),
           cudaMemcpyDeviceToHost);
printf("Results %f\n", a[100]);
```

# Retour d'expérience sur le produit Xeon PHI

## Concept de programmation

---

```
#pragma offload attribute(push, target(mic))
int ComputePartTime(float *a)
{
  /* Compute part */
  #pragma omp parallel for
  for (int i = 0; i < N; i++)
  {
    a[i] = 1.0 / ((float) i) * PI;
  }
}
#pragma offload attribute(pop)
```

```
__global__
int ComputePartTime(float *a)
{
  /* Compute part */
  for (int i = threadIdx.x + blockIdx.x *
        gridDim.x; i < N; i += blockDim.x * gridDim.x)
  {
    a[i] = 1.0 / ((float) i) * PI;
  }
}
```

# Retour d'expérience sur le produit Xeon PHI

## Niveaux de parallélisation

---

- **Deux niveaux de parallélisation**
  1. *Multi-Threading*
    - Pthread posix
    - OpenMP
    - TBB
    - MPI
  2. *Vectorisation*
    - Auto-vectorisation (Le compilateur)
    - Intrinsèques
    - Assembleur (???)



### Conseils d'utilisation

1. Vectoriser d'abord, Multi-Threader ensuite.
2. Bien considérer la profondeur des boucles d'itérations.
3. Vectoriser de manière **bottom-up** dans un code existant.
4. Vectoriser **Top-down** dans un nouveau code.
5. Limiter la vectorisation là où le Multithread sera implémenté.

## Retour d'expérience sur le produit Xeon PHI

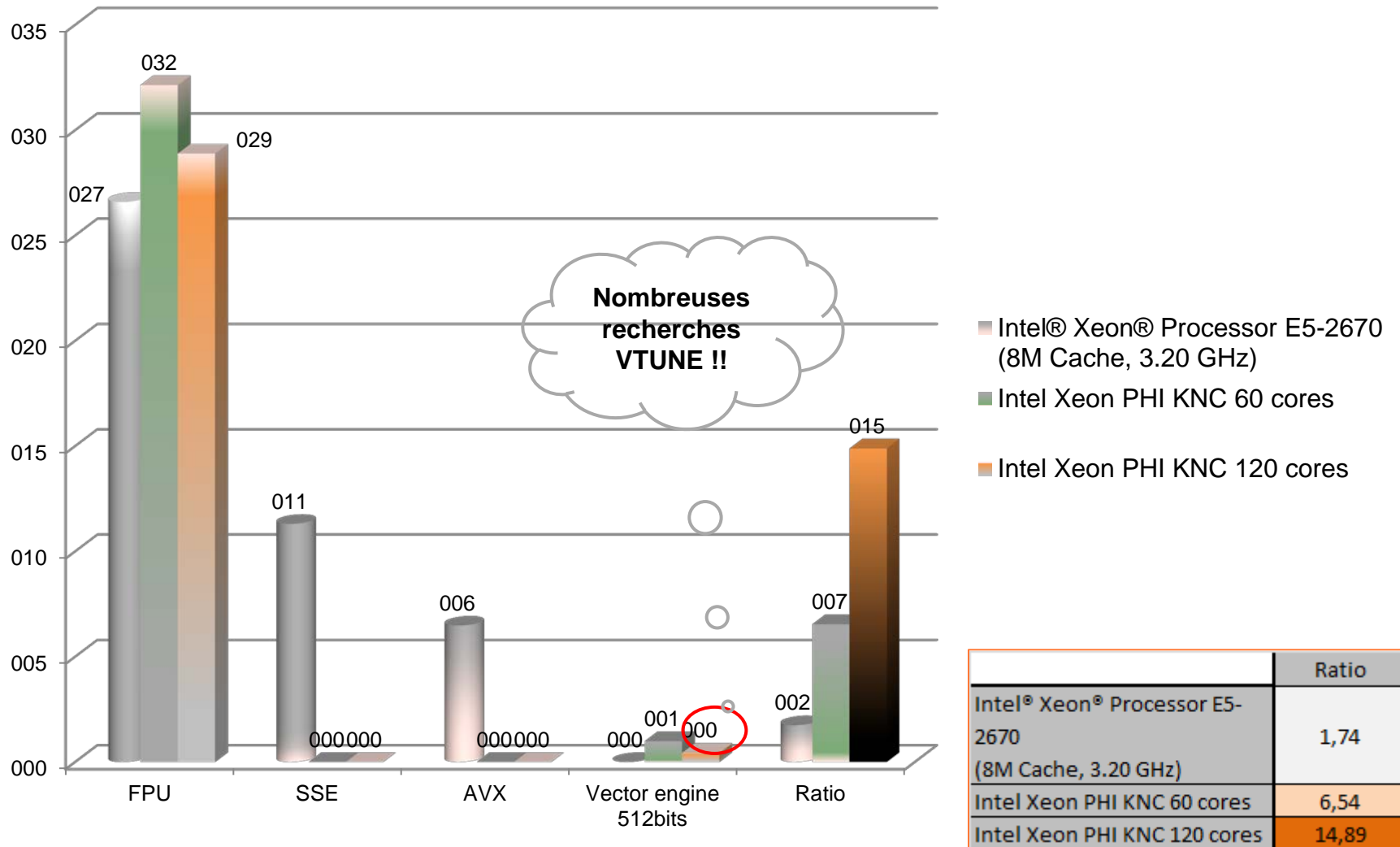
### Exemple d'exécution pour un client

---

- Simulation de risque d'exposition dans une banque...
- 10 000 Simulations, 980 CashFlows et 80 pas de temps
- Soit autant d'itérations « **séquentielles** » pouvant être traitées parallèlement :
  - 10 000 \* 80 interpolations linéaire
  - 10 000 \* 980 \* 80 exponentiels

# Retour d'expérience sur le produit Xeon PHI

## En performance (Temps en secondes)





# Retour d'expérience sur le produit Xeon PHI

## Résultats : Explications

---

### 1. Analyse des Résultats

- Le speed-up n'est pas magique.
- Plusieurs analyses avec Vtune ont été effectuées.
- Le coup des allocations n'est pas pris en compte (fausse le temps de calculs).
- Le temps de démarrage des threads OpenMP n'est pas pris en considération.
- Le temps de copie-mémoire des vecteurs n'est pas inclus pour Xeon PHI.
- Résultats un Speed-up de 15.

### 2. Changement difficile au niveau de la vectorisation des conteneurs

- Implémentation de vecteur de données.
- Implémentation d'un pool mémoire pour les calculs intermédiaires.
- Templatisation des calculs intermédiaire et de l'affectation.

ECHEC

ECHEC

OK

[1] T. L. Veldhuizen, Expression templates, C++ Report, 7 (1995), pp. 26–31. Reprinted in C++ Gems, ed. Stanley Lippman.

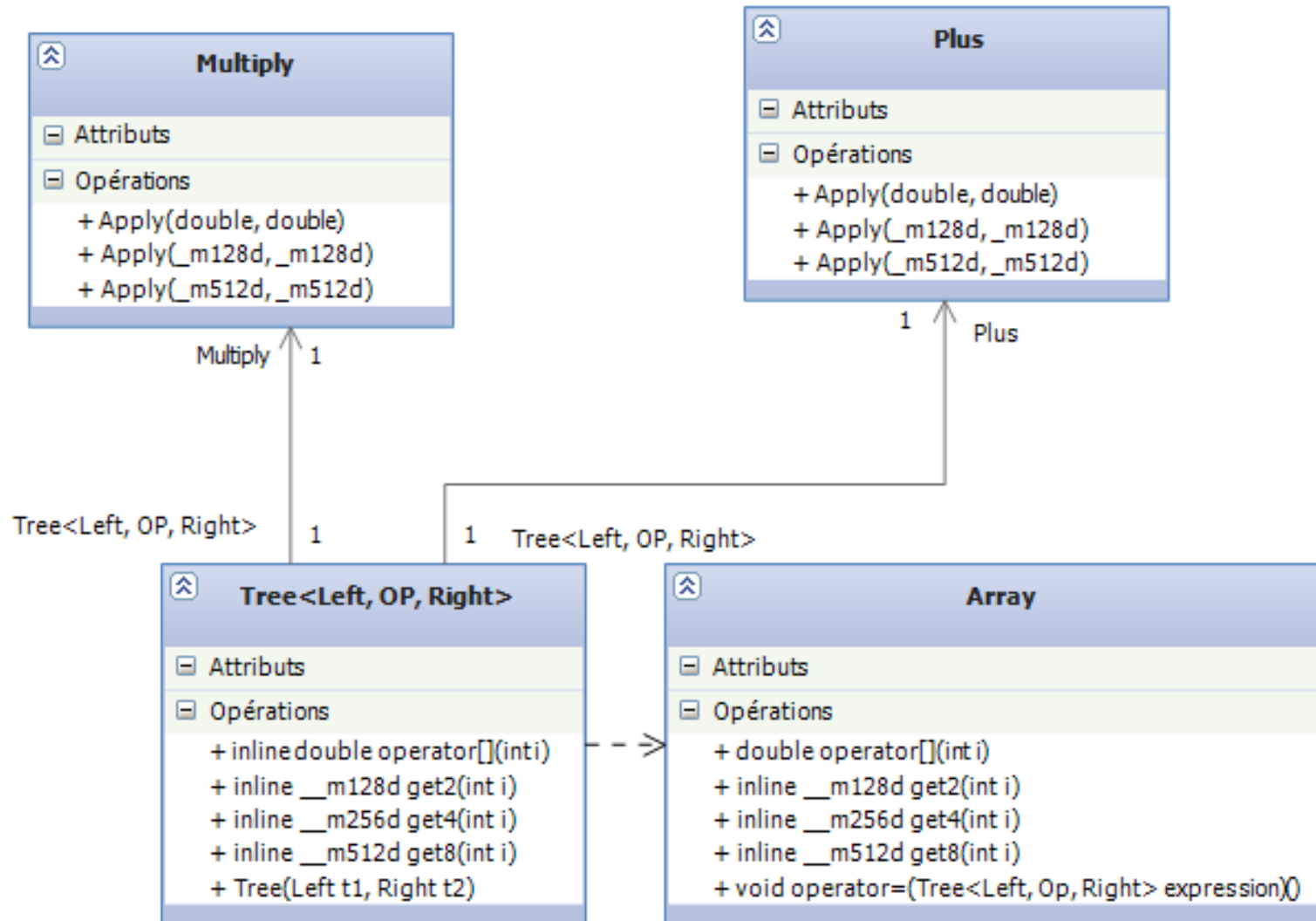
[2] T.L. Veldhuizen, Techniques for Scientific C++, technical report, 1999.

# Retour d'expérience sur le produit Xeon PHI

---

- Avec l'aide d'une méthode de Todd Veldhuizen, Expression Templates, C++ Report 7(5), June 1995, 26{31. Reprinted in C++ Gems, ed. Stanley Lippman.  
<http://oonumerics.org/blitz/papers/>
- <http://www.cs.indiana.edu/pub/techreports/TR542.pdf>

# Retour d'expérience sur le produit Xeon PHI



# Retour d'expérience sur le produit Xeon PHI

## Simple test de vectorisation sur SSE

```

Array A(a, NB);
Array B(b, NB);
Array C(c, NB);
Array D(d, NB);

D = A * B + C * B * A + A * B;

for (int i=0; i < NB; ++i)
    cout << D[i] << " ";
cout << endl;

```

```

Array C(c, NB);
Array D(d, NB);

D = A * B + C * B * A + A * B;
000000013F5F11CB movaps xmm0,xmmword ptr [rsi]
000000013F5F11CE movaps xmm3,xmmword ptr [r15]
000000013F5F11D2 mulpd xmm3,xmm0
000000013F5F11D6 movaps xmm1,xmmword ptr [rbp]
000000013F5F11DA movaps xmm2,xmm1
000000013F5F11DD mulpd xmm2,xmm0
000000013F5F11E1 mulpd xmm3,xmm1
000000013F5F11E5 addpd xmm3,xmm2
000000013F5F11E9 addpd xmm3,xmm2

```

# Retour d'expérience sur le produit Xeon PHI

## Meme test de vectorisation sur MIC

```

Array A(a, NB);
Array B(b, NB);
Array C(c, NB);
Array D(d, NB);

```

```
D = A * B + C + B
```

```

for (int i=0; i <
      cout << [
cout << endl;

```

0x1350	238	vmovapd (%r13), %k0, %zmm11
0x1357	238	inc %edx
0x1359	238	vmovapd (%rbx), %k0, %zmm0
0x135f	238	cmp \$0x989680, %edx
0x1365	238	vfmadd213pd (%rbx), %zmm11, %k0, %zmm0
0x136b	238	mov %al, %al
0x136d	238	vaddpd (%r12), %zmm0, %k0, %zmm1
0x1374	238	mov %al, %al
0x1376	238	vaddpd (%r13), %zmm1, %k0, %zmm2
0x137d	238	vfmadd132pd (%r12), %zmm2, %k0, %zmm11
0x1384	238	nop
0x1385	238	vmovapd %zmm11, %k0, (%rax)
0x138b	238	vmovapd 0x40(%r13), %k0, %zmm6
0x1392	238	vmovapd 0x40(%rbx), %k0, %zmm3
0x1399	238	vfmadd213pd 0x40(%rbx), %zmm6, %k0, %zmm3
0x13a0	238	mov %al, %al
0x13a2	238	vaddpd 0x40(%r12), %zmm3, %k0, %zmm4
0x13aa	238	mov %al, %al
0x13ac	238	vaddpd 0x40(%r13), %zmm4, %k0, %zmm5
0x13b3	238	vfmadd132pd 0x40(%r12), %zmm5, %k0, %zmm6
0x13bb	238	nop

# Retour d'expérience sur le produit Xeon PHI

---

- **Les pièges à éviter :**
  1. Eviter les Threads locals storage
  2. Bloquer l'auto-vectorisation lors de l'utilisation des fonctions intrinsèques
    1. Pragma novector
    2. -fno-vec sur le fichier compilé
  3. Faites du pré-calcul sur la carte lorsque cela est possible.
  4. Aligner automatiquement les zones mémoires à 512 bits.
    1. `_mm_malloc(size, 64)`
  5. Vérifier qu'il n'y a pas de cache missed pendant l'exécution.
- Quelques commandes :
  - `MIC_ENV_PREFIX=MIC MIC_OMP_NUM_THREADS=240 \`  
`MIC_KMP_AFFINITY=granularity=thread,verbose,scatter ./MON_BINAIRE`
  - `H_TRACE=4 ./MON_BINAIRE`

## Retour d'expérience sur le produit Xeon PHI

### *Avantage du langage*

---

- Tous les outils et bibliothèque d'Intel.
- La MKL disponible sur Xeon PHI.
- Le compilateur Intel Composer 2013 (le code OFFLOAD est automatiquement reconnu et exécuter sur une carte si elle est présente.
- Un système Unix sur la carte Xeon PHI.
- L'intercommunication des cartes.
- Une mémoire virtuelle (possibilité de faire un SWAP mémoire avec la RAM CPU.
- L'allocation sur la carte est automatique; Les pointeurs mémoire sur CPU sont automatiquement associés à celui de la carte.

## Retour d'expérience sur le produit Xeon PHI

### *Inconvénient du langage*

---

- Une allocation de page mémoire qui reste chère.
- Conséquence : l'automatisation de l'association mémoire devient un problème.
- Un système de garbage complexe à mettre en place.
- La complexité à passer toute une structure lors d'un appel offload.
- Désassocier une mémoire CPU d'une mémoire Xeon PHI.
- Les intrinsèques sont parfois incompatibles avec l'auto vectorisations.
- Le gather-scatter coute très chère.
- Un système de cache mémoire trop restrictif. Thanks to the prefetcher



# Retour d'expérience sur le produit Xeon PHI

## *Pour résumer*

---

- Tests sur des cartes en beta test. Une nouvelle version verra le jour très prochainement.
- Une carte de calcul satisfaisante pour ses débuts.
- Le kernel de la carte KNC reste à améliorer (merci la communauté).
- Carte massivement multi-cœur prometteuse.

## *Le mot de la fin...avant les questions*

---

- ANEO continuera à tester ce produit. Et tous les autres qui arriveront par la suite.



“My desire to inflict pain on the compilers is large. They've been tormenting me for the last 8 years. Now's my chance to strike back!”

-- Scott Haney

*Le mot de la fin...avant les questions*

---

Des questions ?

Des timides ?

[ddubuc@aneo.fr](mailto:ddubuc@aneo.fr)