Parallélisme(s)

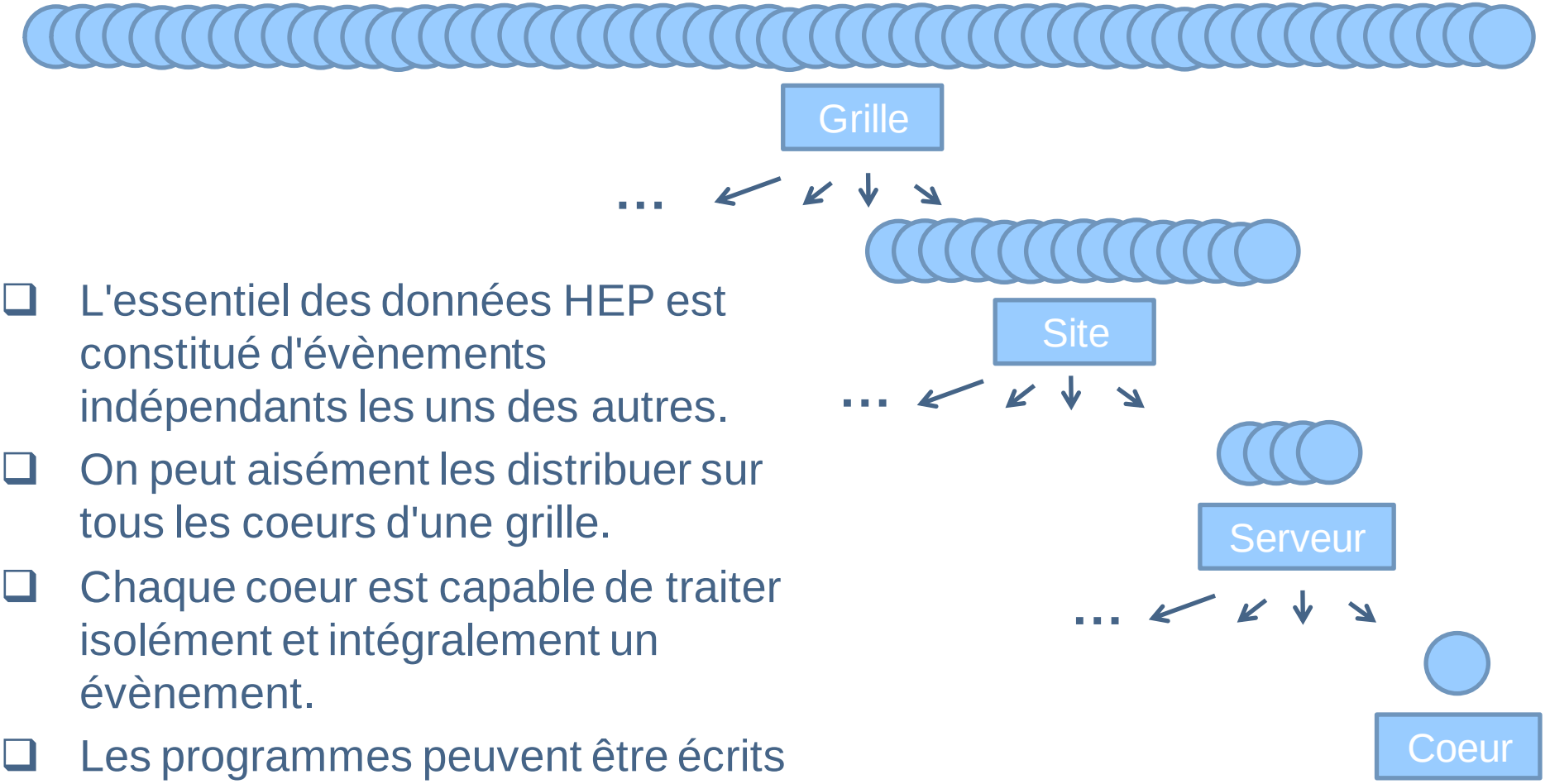
Service Informatique
David Chamont & Co

Palaiseau, France

Plan

1. Modèle de calcul actuel : la distribution des évènements.
2. Evolution matérielle :
    1. **instructions vectorielles,**
    2. **multiplication des coeurs,**
    3. **augmentation des caches.**
3. Les réponses logicielles :
    1. **techniques de vectorisation,**
    2. **multi-threading (mémoire partagée),**
    3. **calcul sur coprocesseur,**
    4. **multi-noeuds (mémoire distribuée).**
4. Projet P2IO GridCL

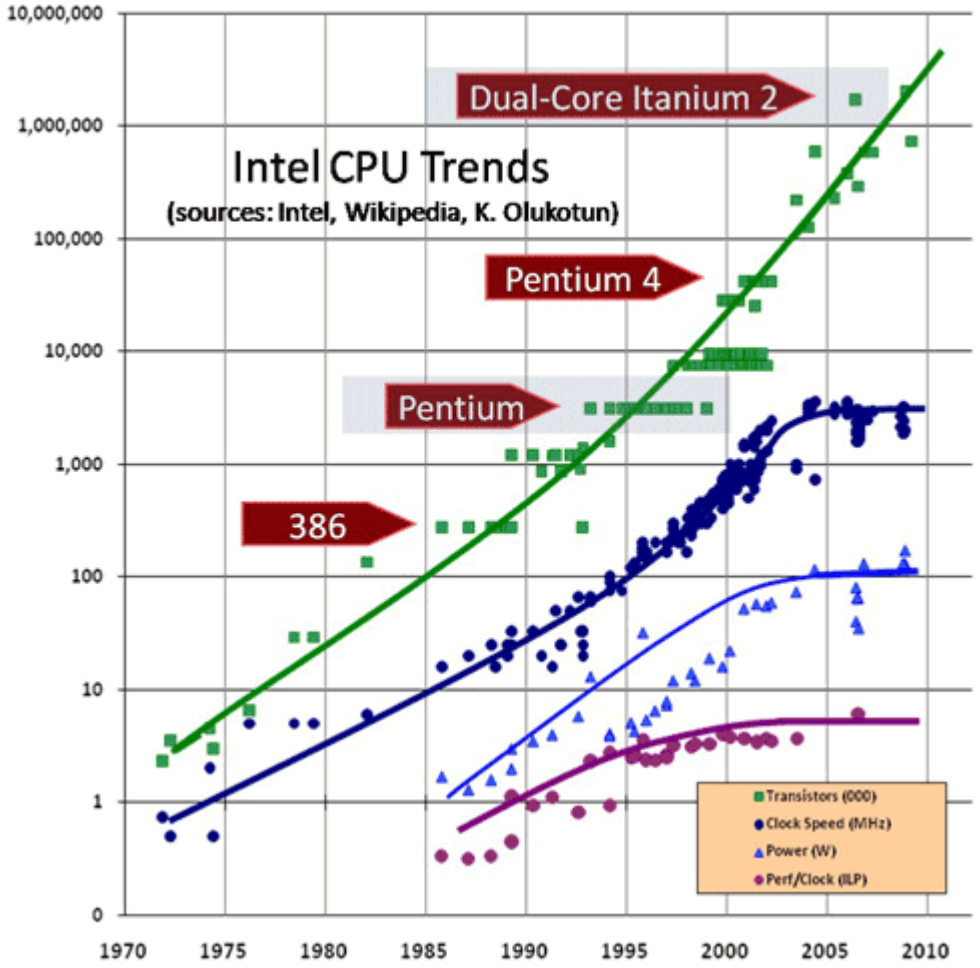
Parallélisme "embarrassant"



- L'essentiel des données HEP est constitué d'évènements indépendants les uns des autres.
- On peut aisément les distribuer sur tous les coeurs d'une grille.
- Chaque coeur est capable de traiter isolément et intégralement un évènement.
- Les programmes peuvent être écrits de façon séquentielle. Youpi !

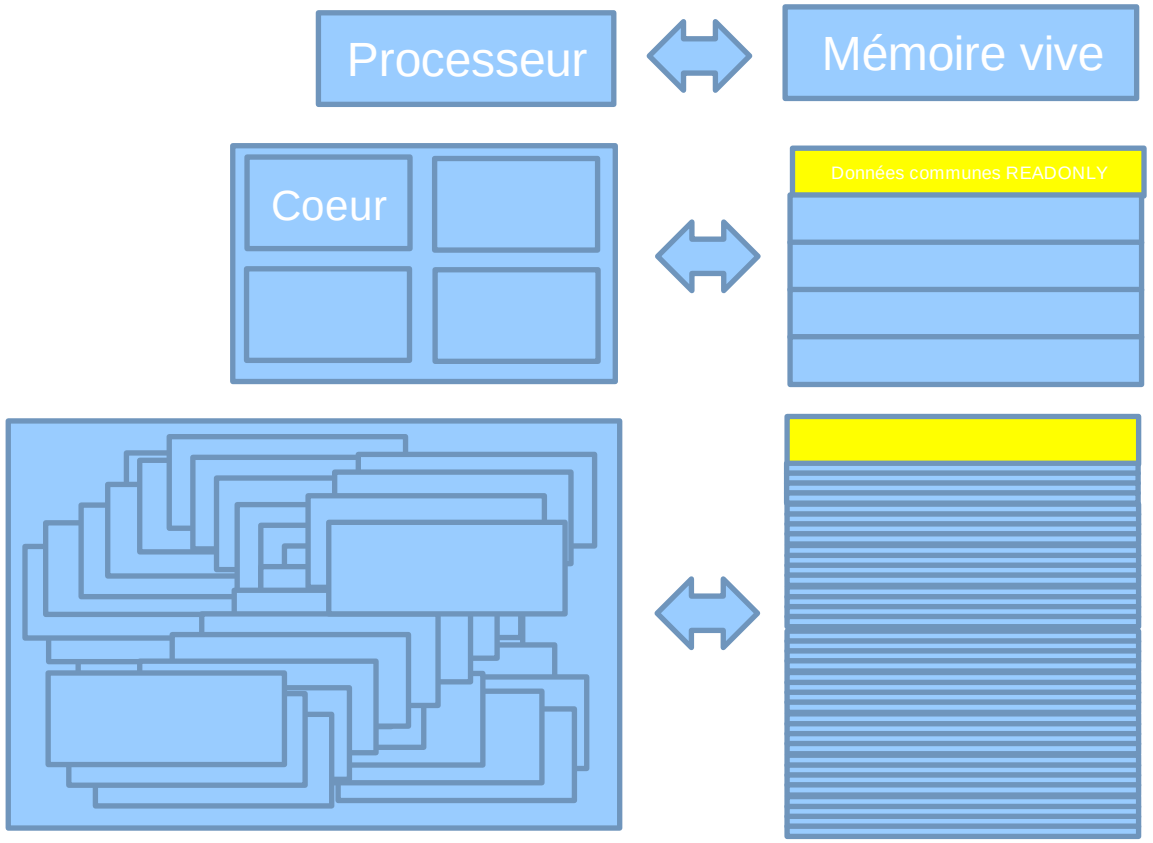
Evolution matérielle

- Fréquence d'horloge et puissance électrique stagnent.
- Le nombre de transistors continue sa croissance :
  - **Multiplication des coeurs**
  - **Ajout de capacités de calcul vectoriel au sein des coeurs : SSE, AVX.**
  - **Augmentation des caches.**



Pénurie de mémoire

- Hier
- Aujourd'hui
  - **multi-coeurs**
- Demain (HEP)
  - **many-coeurs**

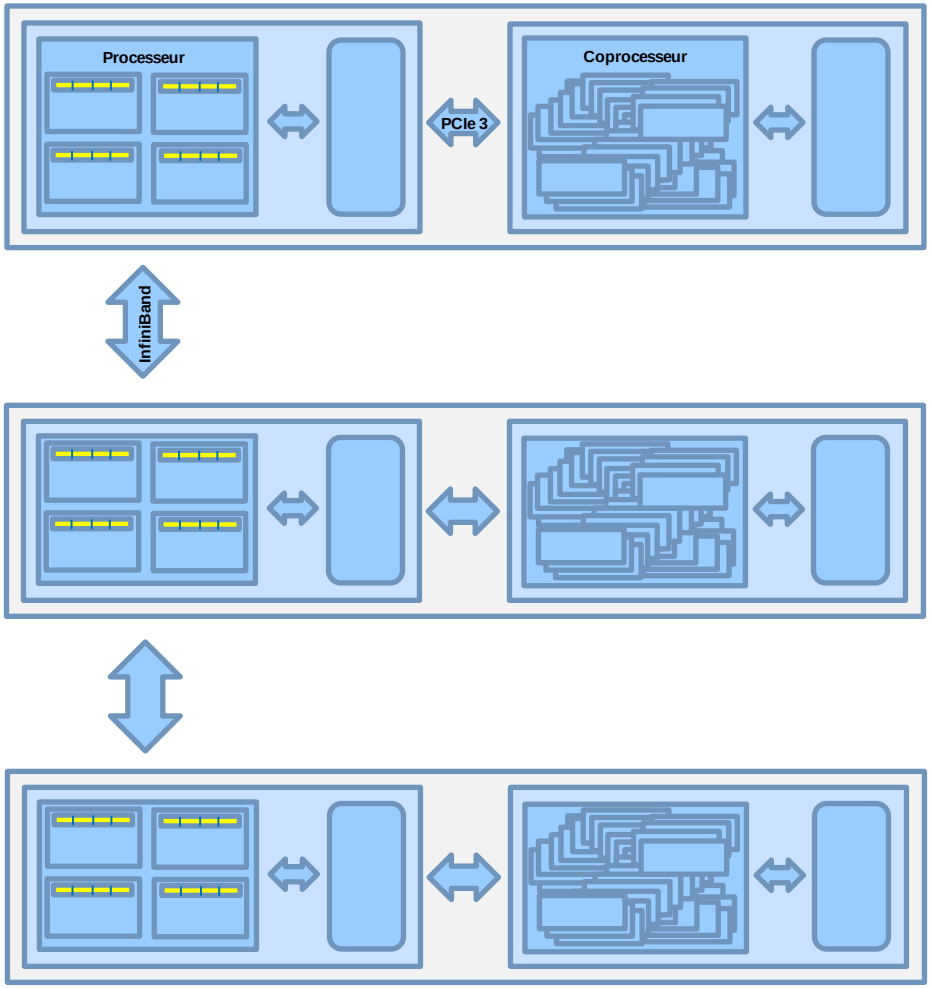


- Un cœur ne peut plus traiter l'intégralité d'un évènement
- Il faut morceler ce traitement en tâches… qui ne sont plus indépendantes …

Parallélismes

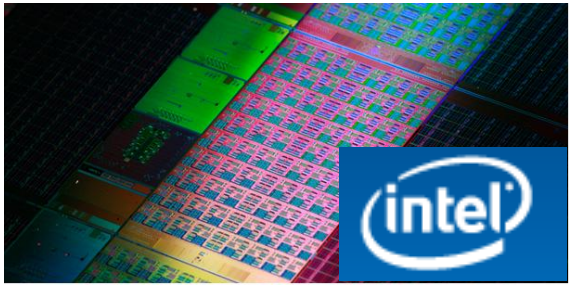
- Vectoriel
  - **Instructions SSE et AVX, au sein des cœurs du processeur.**
- Multi-thread / multi-coeurs
  - **Mémoire partagée**
  - **OpenMP**
- Coprocesseurs many-coeurs
  - **CUDA, OpenCL**
- Multi-(co)processeurs
  - **Mémoire distribuée**
  - **MPI**

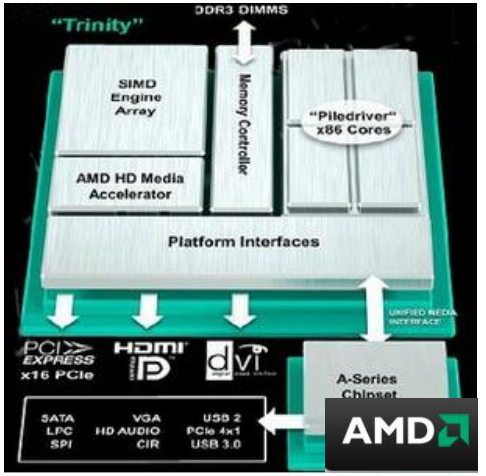
➢ Programmation hybride.



Trois approches many-coeurs

- NVIDIA : architecture CUDA
  - **Toujours plus de coeurs dans le GPU.**
  - **Dialogue direct entre GPUs : GPUDirect.**
  - **M2090 : 512 coeurs, 6 GB.**
  - **K20 : 2880 cœurs, 12 GB ?**
- INTEL : architecture MIC
  - **Permettre de faire tourner les applications actuelles sans modifications**
  - **Xeon Phi : >50 coeurs ?**
- AMD : architecture FUSION
  - **Rachat d'ATI en 2006.**
  - **Depuis, rapprochement entre CPU et GPU => APU (Accelerated Processing Units)**



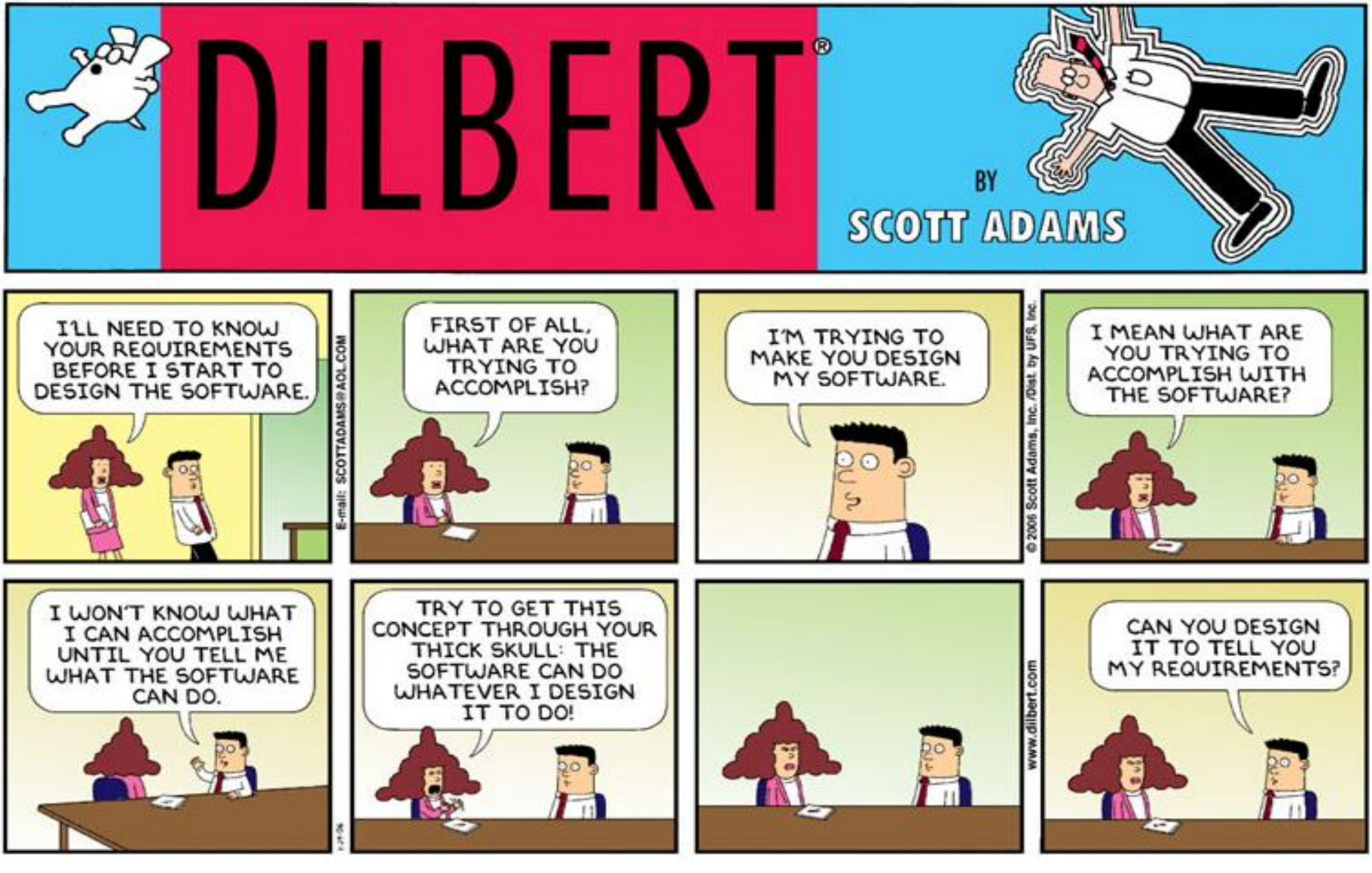


Projet P2IO GridCL

- **Objectif** : Evaluer l'intérêt d'utiliser du matériel **many-coeurs** au sein d'une **grille de calcul**, en s'appuyant sur le standard **OpenCL**, et pour les applications des laboratoires du labex P2IO, notamment la physique des hautes énergies.
- Une région riches en compétences et en moyens de calcul intensif
  - **OpenGPU, Teratec, Maison de la simulation, …**
  - **2 des 3 centers nationaux financés par GENCI :TGCC et IDRIS.**
- Partenaires et applications pilotes
  - **LLR : CMS Ions lourds Tracking et Fitting**
  - **LAL : ATLAS Fitting**
  - **IPNO : ALICE Tracking**
  - **IRFU, IMNC, IAS : expertise CUDA, astro, hGATE**
- Plate-forme de test à construire
  - **2 cartes bi-proc ivy-bridge x 2 K20, + infiniband**
  - **2 cartes bi-proc ivy-bridge x 2 Xeon Phi, + infiniband**
  - **2 cartes bi-proc ivy-bridge x 2 AMD, + infiniband**

Conclusions

- La multiplication des cœurs remet en cause notre modèle de calcul.
- Les différentes formes de parallélisme vont impacter votre pratique du développement.
- Préparons nous ! Démontons les objets… mais pas partout…
- Il y a d'ores et déjà des techniques simples qui peuvent facilement accélérer certains calculs jusqu'à 4 fois (vectorisation).


DILBERT®
BY
SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc. /Dist. by UFS, Inc.
www.dilbert.com

Extrait de code OpenMP

```
#pragma omp parallel for reduction(+: s)

    for ( int i = 0; i < n; i++)
      {  s += x[i] ; }
```

Extrait de code OpenCL

```
kernel void  dotprod( global const foat *a, global const foat *b, global foat *c)
      { int myid = get_global_id(0)  ; c[myid] = a[myid] * b[myid] ; }
```

Extrait de code MPI

```
for (d=1; d<ntasks; d++) {
      rows = (d <= extra) ? avrow+1 : avrow;
      printf(“ sending %d rows to task %d\n”, rows, dest);
      MPI_Send(&ofset, 1, MPI_INT, d, mtype, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, d, mtype, MPI_COMM_WORLD);
      MPI_Send(&a[ofset][0], rows*NCA, MPI_DOUBLE, d, mtype, MPI_COMM_WORLD);
      MPI_Send(&b, NCA*NCB, MPI_DOUBLE, d, mtype, MPI_COMM_WORLD);
      ofset = ofset + rows; }
```