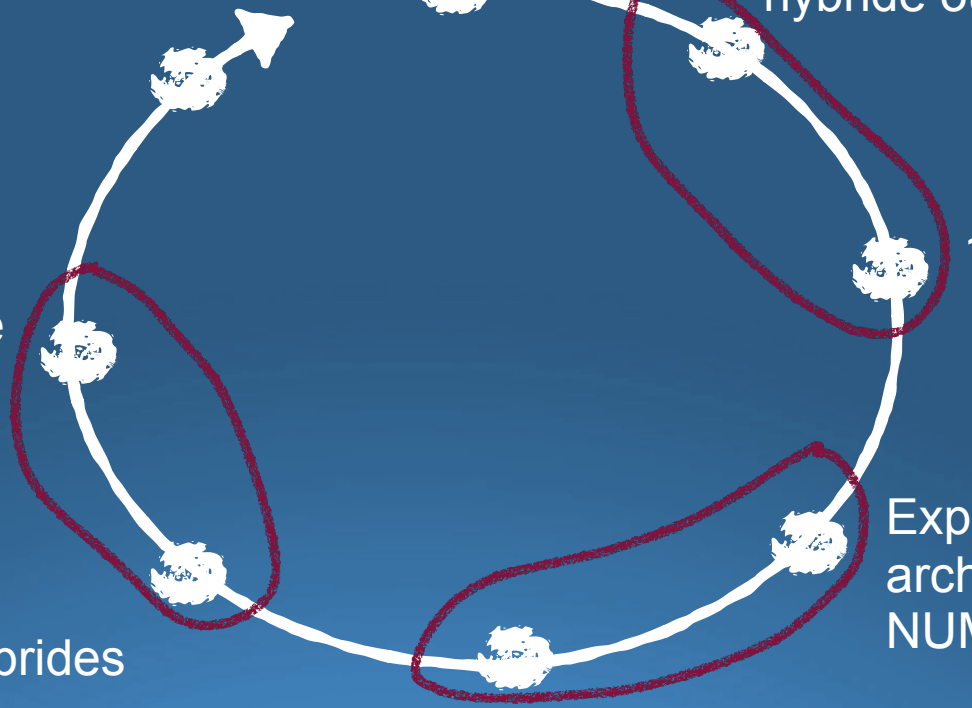

# Programmation parallèles des architectures modernes de calcul: pourquoi ? et comment ?

thierry.gautier@inrialpes.fr
CR INRIA, EPI AVALON

# Agenda



Introduction

Rappel: architecture hybride ou NUMA

1 Challenge !

Exploitation des architectures NUMA

Gestion de l'affinité et évaluation

Exploitation des architectures hybrides

Evaluation de performance sur une architecture hybride à 8 GPUs

Bilan & Perspectives

# Pourquoi la « programmation parallèle »

- **Bénéficier des progrès technologiques**
  - la fréquence des processeurs stagne depuis environ 10 ans
  - le nombre de transistors augmente
  - la parallélisation automatique fonctionne dans un cadre limité

➡ **Gain en temps de calcul**
  - problème plus gros avec une échéance de résultat
  - plusieurs problèmes simultanément
  - être plus efficace

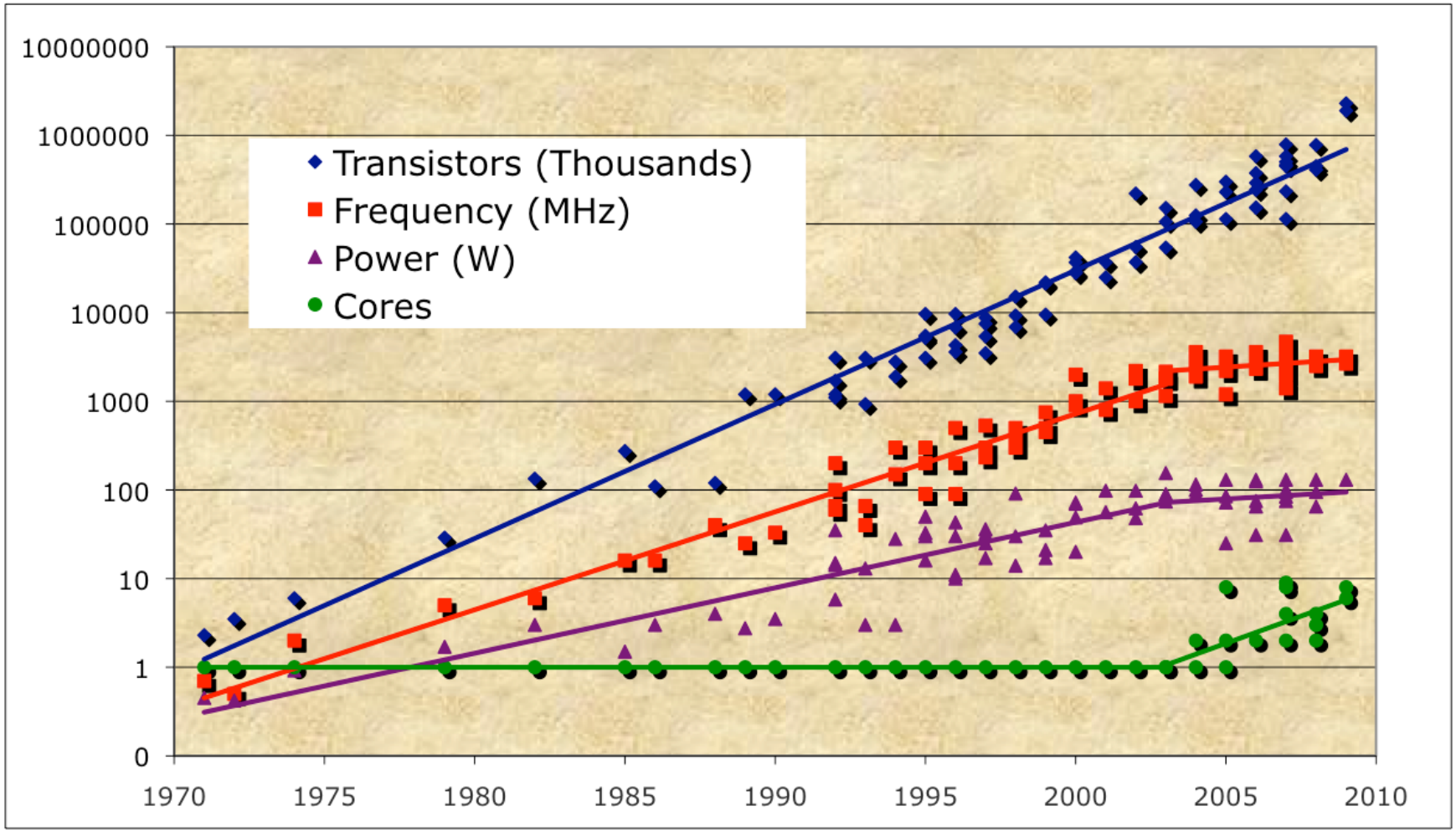
➡ **Permet de traiter des problèmes plus importants**
  - utilisation de la mémoire de plusieurs processeurs

- **Si votre application n'a pas été parallélisé, vous n'avez sans doute plus le choix**

# Top 500: https://www.top500.org

| Rank | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC<br>National Supercomputing Center in Wuxi<br>China | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 2 | **Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT<br>National Super Computer Center in Guangzhou<br>China | 3,120,000 | 33,862.7 | 54,902.4 | 17,808 |
| 3 | **Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc.<br>Swiss National Supercomputing Centre (CSCS)<br>Switzerland | 361,760 | 19,590.0 | 25,326.3 | 2,272 |
| 4 | **Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler<br>Japan Agency for Marine-Earth Science and Technology<br>Japan | 19,860,000 | 19,135.8 | 28,192.0 | 1,350 |
| 5 | **Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc.<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 560,640 | 17,590.0 | 27,112.5 | 8,209 |
| 6 | **Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM<br>DOE/NNSA/LLNL<br>United States | 1,572,864 | 17,173.2 | 20,132.7 | 7,890 |
| 7 | **Trinity** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc.<br>DOE/NNSA/LANL/SNL<br>United States | 979,968 | 14,137.3 | 43,902.6 | 3,844 |
| 8 | **Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc.<br>DOE/SC/LBNL/NERSC<br>United States | 622,336 | 14,014.7 | 27,880.7 | 3,939 |
| 9 | **Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu<br>Joint Center for Advanced High Performance Computing<br>Japan | 556,104 | 13,554.6 | 24,913.5 | 2,719 |
| 10 | K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu<br>RIKEN Advanced Institute for Computational Science (AICS)<br>Japan | 705,024 | 10,510.0 | 11,280.4 | 12,660 |

# Tendance




Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

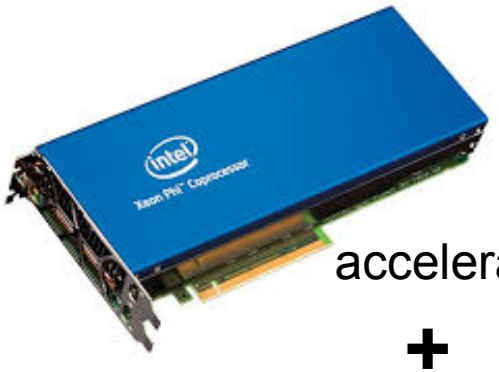
# Many-{node,core} to Multi-{node,core}


processor

+

memory

+





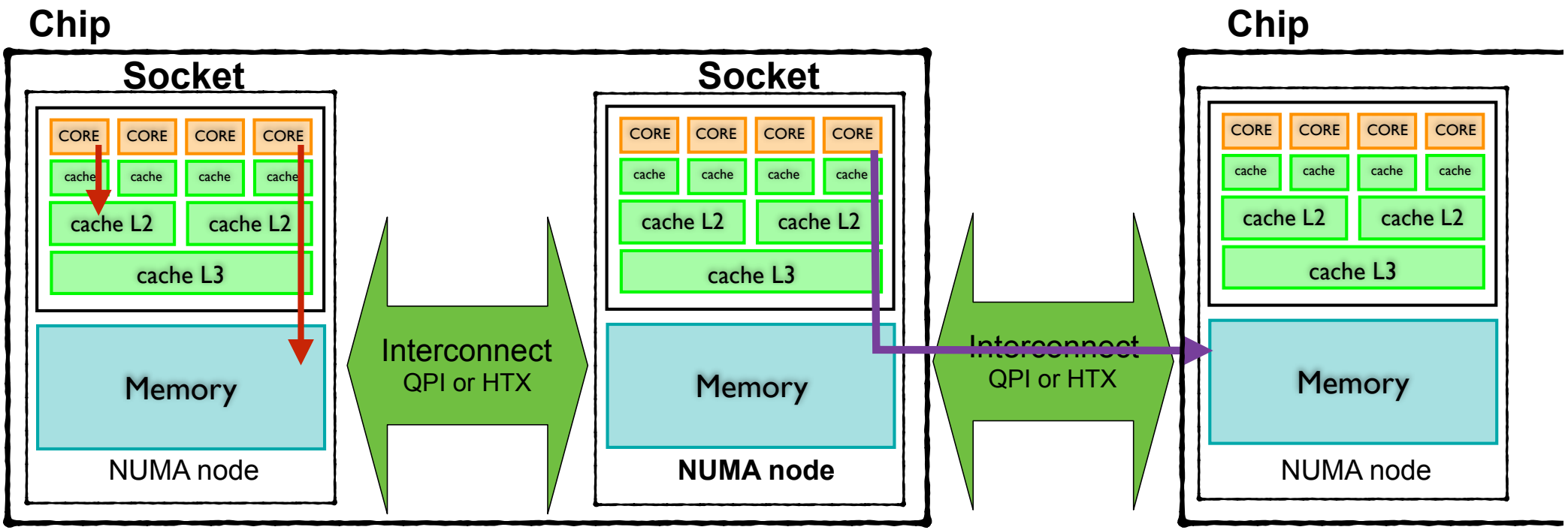
accelerator

+

+ network
+ I/O
+ …

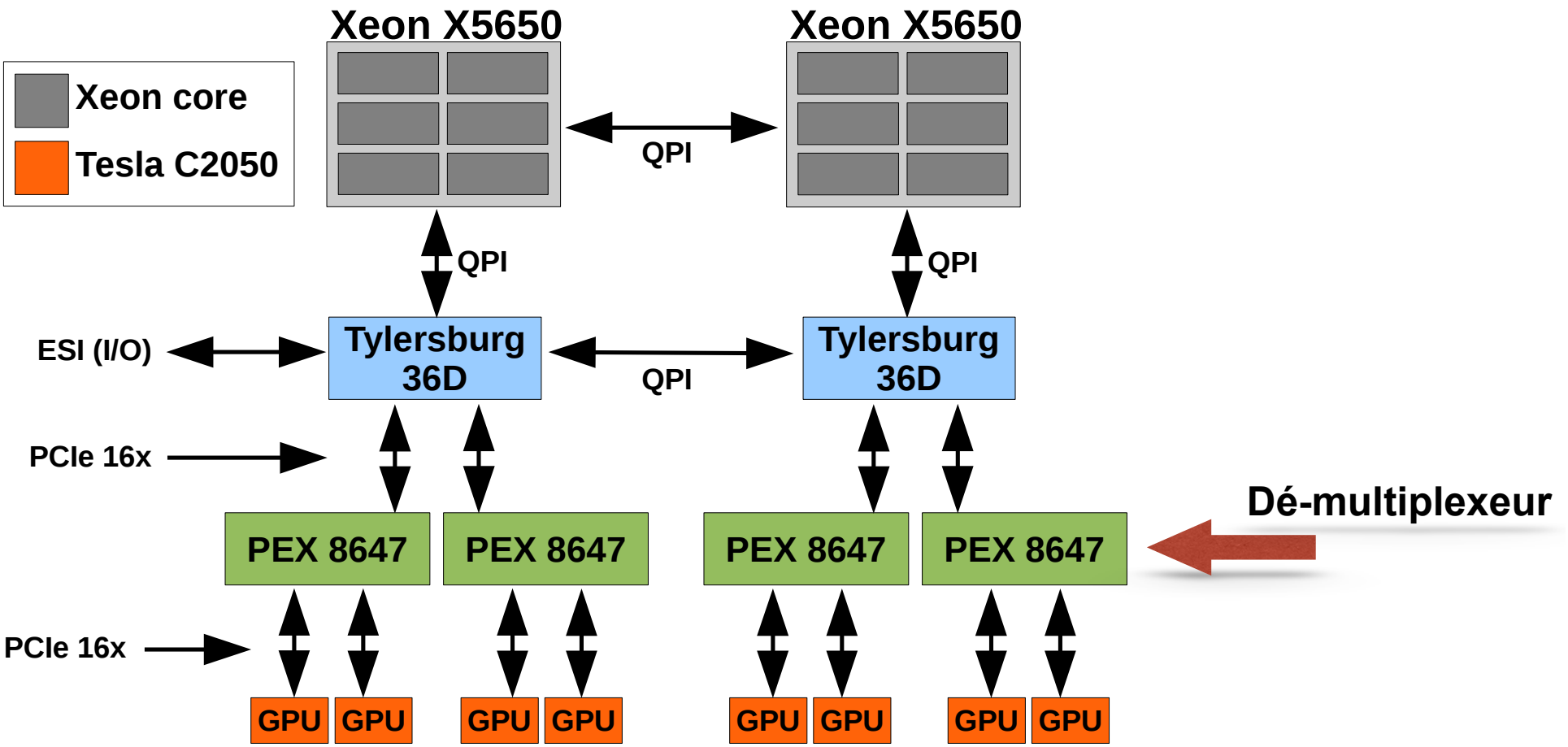

# Architecture NUMA

- NUMA = Non Uniform Memory Access
  - ≠ PRAM mostly due to Non Uniform Memory Access



# NUMA hybride multi-CPUs/multi-GPUs

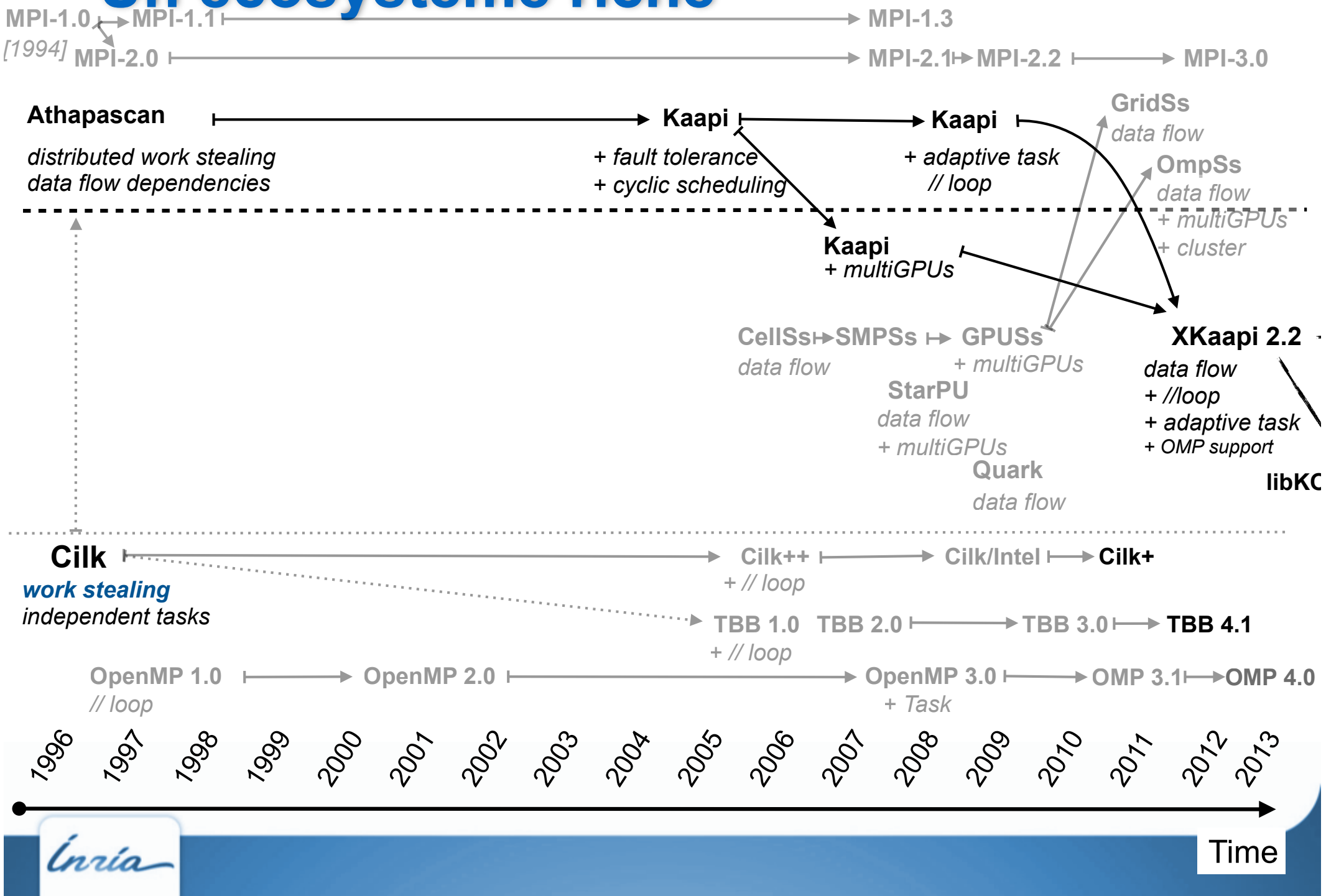
- idgraph [G5K/Digitalis, 2011] 8 GPUs et 12 CPUs



# Caractéristique de ces architectures

- **NUMA**
  - **le coût d'exécution dépend de l'emplacement du calcul et des données**
- **Calcul hybride CPU / GPU**
  - **choix du placement du calcul**
  - **gestion de communication CPU/GPU ou GPU/GPU (~mémoire uniforme)**
- **Plusieurs modèles de programmation parallèle**
  - **NUMA:**
    - parallélisme entre cores
    - parallélisme entre les instructions + vectoriel (AVX)
  - **GPU**
    - idem mais avec une autre terminologie [Jonathan]

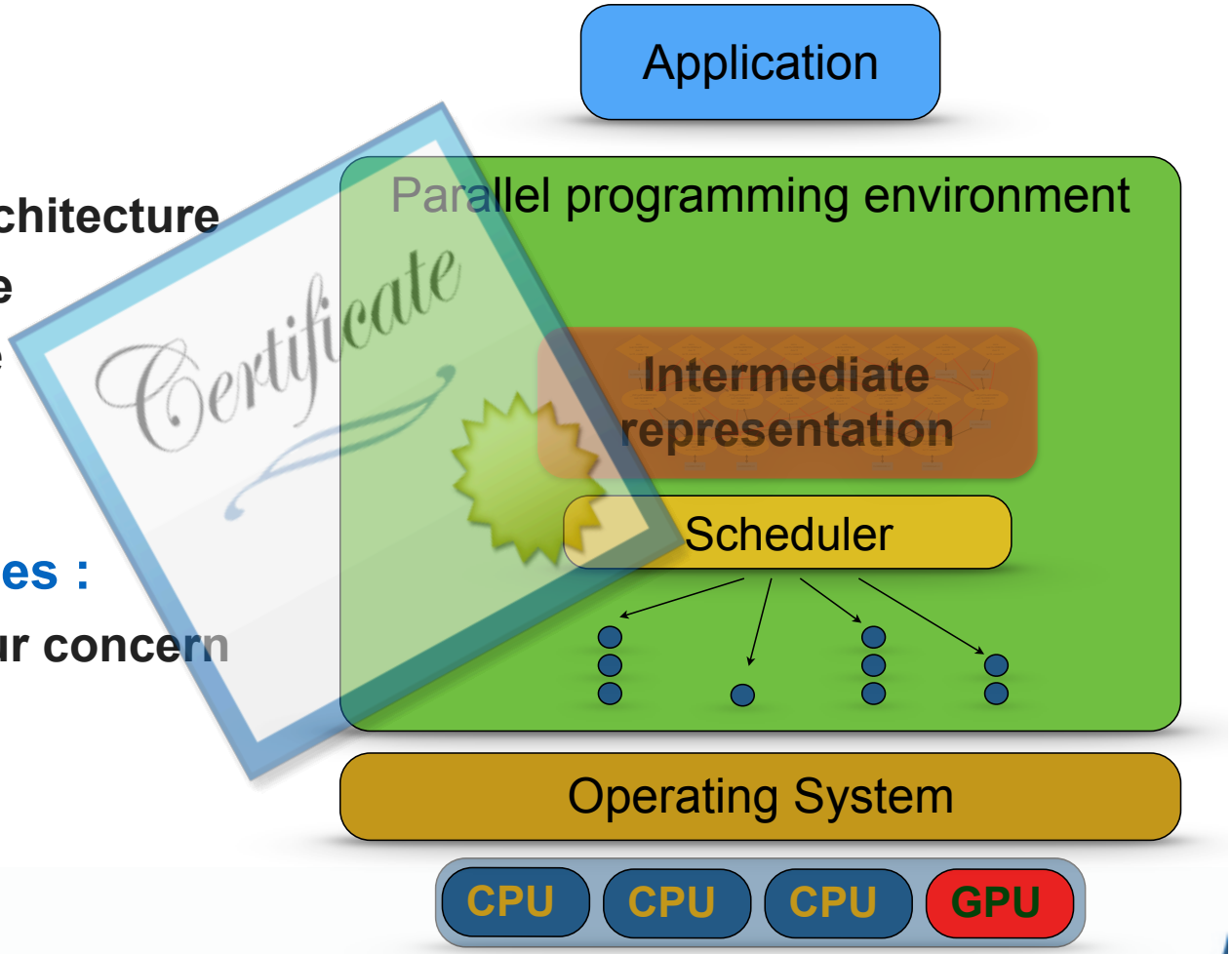
# Un écosystème riche



# Challenge: portabilité des performances !

- **Portabilité des performances**
  - **sur plusieurs générations d'architecture**
  - **avec de l'équilibrage de charge**
    - OS jitter, application irrégulière

- **Approche proposée en 2 étapes :**
  1. **algorithme parallèle ☞ its your concern**
     - communications
  2. **scheduling as a plugin**

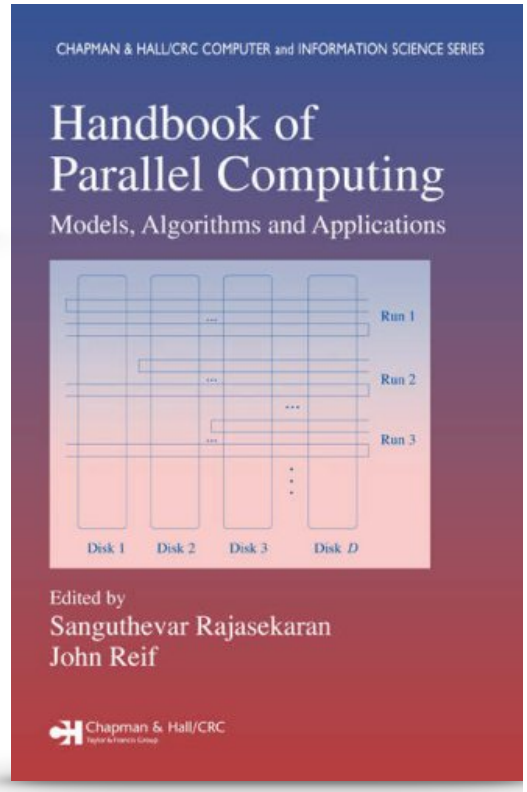


# Comment programmer en parallèle ?

## 1. Parallélisation de l'algorithme de calcul

- décomposition du problème en sous problèmes indépendants
- un « bon algorithme parallèle »
  - efficace (~ même nombre d'opérations qu'en séquentiel)
  - très parallèle afin d'exploiter tout le parallélisme disponible

$T_p$ , $T_a$ , $W$




- **Loi d'Amdahl (pessimiste)**
  - s: fraction du programme exécuté en séquentiel
  - (1-s): fraction du programme exécuté en parallèle
  - p nombre de cœurs
  - Alors le speedup est borné par : $Speedup(p) = \frac{T_{séquentiel}}{T_p}$

- **Loi de Gustafson (optimiste)** $\leq \frac{1}{s + \frac{1-s}{p}}$

# Comment exécuter ces programmes ?

## 2. Réguler la charge de travail sur les cœurs

- **étape 1 => degré de parallélisme important**
  - portabilité, « parallel slackness »
- **problème du re-pliage des activités dues aux décompositions en sous problème sur des ressources limitées**
  - problème d'ordonnancement compliqué
  - approche « offline »
    - un modèle d'application : classiquement un graphe de tâche
    - un modèle de performance : connaître le temps d'exécution d'une tâche sur une ressource
    - un algorithme qui calcule une date et un site d'exécution pour chaque tâche afin d'optimiser une fonction objectif (temps d'exécution, mémoire, énergie)
  - approche « online »
    - prédiction de temps ou temps d'exécution réels non connu à l'avance
    - Vol de travail, avec une bonne probabilité :

$$T_p = O\left(\frac{T_1}{p} + T_\infty\right)$$

# Programmation des architectures NUMA

- **Multi/many core**
- **Top-down classification**
  - **network**
    - MPI, PGAS (UPC), X10
  - **multi-core**
    - OpenMP, Cilk, Intel TBB, XKaapi, StarPU…
  - **accelerator**
    - Cuda, OpenCL, OpenACC, OpenMP (4.0)
  - **vector / SIMD unit**
    - compiler, extended type, OpenMP (4.0)

## OpenMP 4.x

**Permanent Members of the ARB:**
- **AMD** (Greg Stoner)
- **ARM** (Chris Adeniyi-Jones)
- **Cray** (Luiz DeRose)
- **Fujitsu** (Eiji Yamanaka)
- **HP** (Sujoy Saraswati)
- **IBM** (Kelvin Li)
- **Intel** (Xinmin Tian)
- **Micron** (Kirby Collins)
- **NEC** (Kazuhiro Kusano)
- **NVIDIA** (Jeff Larkin)
- **Oracle Corporation** (Nawal Copty)
- **Red Hat** (Matt Newsome)
- **Texas Instruments** (Andy Fritsch)

**Auxiliary Members of the ARB:**
- **Argonne National Laboratory** (Kalyan Kumaran)
- **ASC/Lawrence Livermore National Laboratory** (Bronis R. de Supinski)
- **Barcelona Supercomputing Center** (Xavier Martorell)
- **cOMPunity** (Barbara Chapman/Yonghong Yan)
- **Edinburgh Parallel Computing Centre (EPCC)** (Mark Bull)
- **Los Alamos National Laboratory** (David Montoya)
- **Lawrence Berkeley National Laboratory** (Alice Koniges/Helen He)
- **NASA** (Henry Jin)
- **Oak Ridge National Laboratory** (Oscar Hernandez)
- **RWTH Aachen University** (Dieter an Mey)
- **Sandia National Laboratory** (Stephen Olivier)
- **Texas Advanced Computing Center** (Kent Milfeld)
- **University of Houston** (Barbara Chapman/Deepak Eachempati)

**[+2016]**
- **INRIA Storm (O. Aumage)**

# Zoom sur deux modèles de programmation par tâche

- **Thème de recherche actif**
  - **Old Age: Cilk, Athapascan, Jade, Sam, Sisal, Multilisp, Fx-Fortran…**
  - **OpenMP-3 and 4.x, OMPss, StarPU, Kaapi & XKaapi, Swan, Qthread, Quark, X10, IntelTBB, Microsoft PPL, HPX…**

- **De bonnes propriétés des modèles par tâche**
  - **Composabilité**
  - **Portabilité des codes = *scheduler as a plugin***
    - code = déclaration des tâches + dépendances
    - scheduler restant difficile à prouver

- **Zoom**
  - **Multi-CPU NUMA = OpenMP + libKOMP**
  - **Multi-CPUs/GPUs = Kaapi**

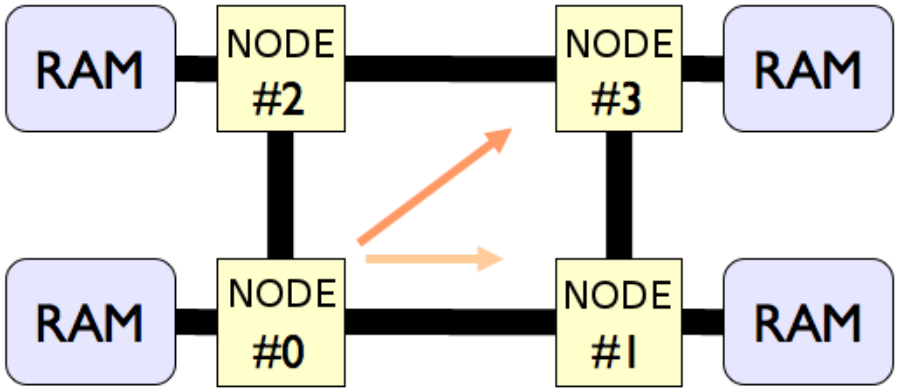
# OpenMP + libKOMP

**Prise en compte des aspects NUMA**
**Travail effectué avec F. Broquedis, UGA, LIG équipe CORSE et P. Virouleau, PhD.**

# Architecture NUMA

- **Coût des calculs dépendant de :**
  - **localisation des données**
  - **charges des liens d'interconnexion des mémoires**

- **Exemple :**




| Access to... | Local node | Neighbor node | Opposite node |
|---|---|---|---|
| Read | 83 ns | 98 ns (x1.18) | 117 ns (x1.41) |
| Write | 142 ns | 177 ns (x1.25) | 208 ns (x1.46) |

© P. Virouleau

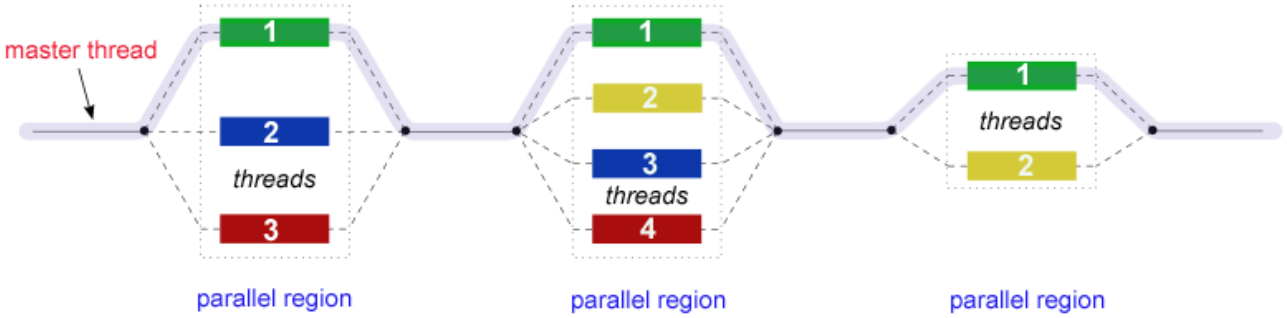
# Programmation par tâche & NUMA

- **Impact sur l'algorithme**
  - **a minima distribution des données sur les bancs NUMA**
    - outil système : numactl
    - initialisation = une tâche + first touch policy de l'OS
      - le core d'exécution allouera les pages touchées
- **Comment mieux gérer la localité ?**
  - **le scheduler a une représentation du programme**
    - graphe de tâches
  - **quelle politique de scheduling ?**
  - **impact sur le runtime ?**

# OpenMP

- **Basé sur des directives de compilation**
  - le compilateur générer des appels à des fonctions d'un runtime pour gérer quasiment tout le parallélisme
  - C,C++, Fortran

- **Modèle d'exécution**
  - séquence de régions parallèles / régions séquentielles

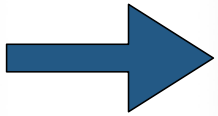



- **Des directives pour:**
  - distribuer le travail entre threads (work sharing constructs)
  - task (independent and dependent)
  - synchronisation

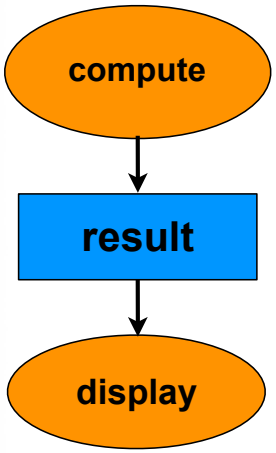
# Création de tâche

- création de tâche == annotation OpenMP
- Ici tâches avec dépendance

```
void main()
{
  /* data result is produced */
  compute( size, result );

   /* data result is consumed */
  display( size, result );
}
```



```
void main()
{
  #pragma omp parallel
  #pragma omp master
  {
  #pragma omp task depend(out: result)
    compute( size, result );

  #pragma omp task depend(in: result)
    display( size, result );
  }
}
```



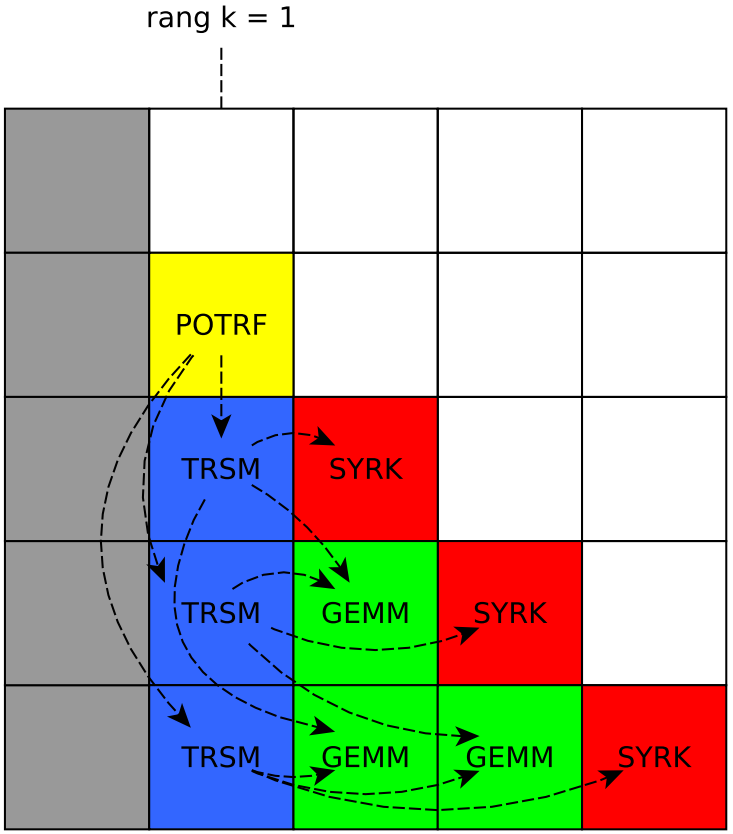

- Compile with GCC, ICC, CLANG et bien d'autres, IBM, PGI...

- Ok, Easy ! Let us introduce Cholesky matrix factorization

# Factorisation de Cholesky

- **Factorisation d'une matrice A symétrique définie positive**
  - **A = L * L$^t$**
  - **Résolution de système d'équations en calcul scientifique**

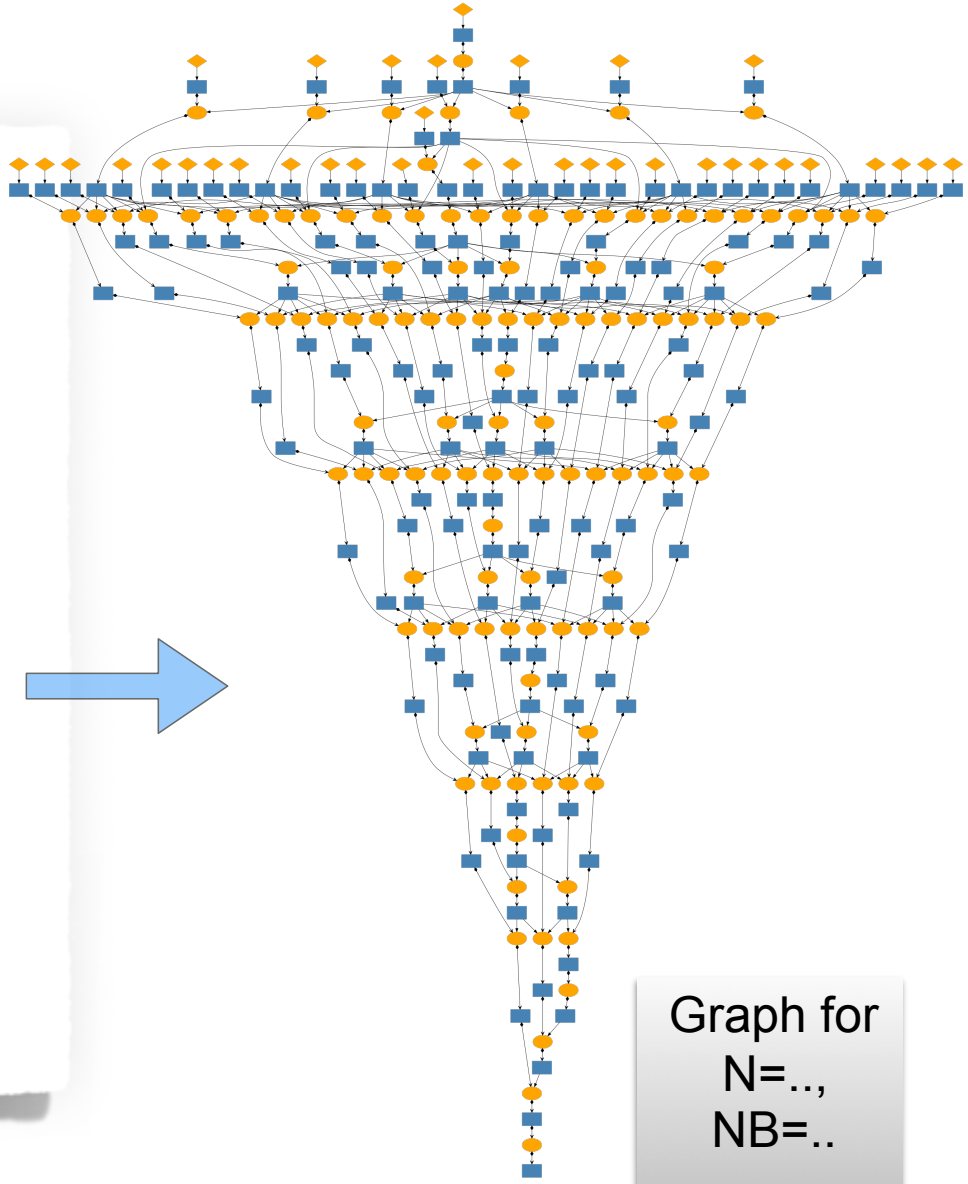
- **Plusieurs algorithmes**
  - **ici algorithme par bloc, « à la PLASMA »**
  - **4 noyaux principaux**
    - dpotrf
    - trsm
    - syrk
    - dgemm




© P. Virouleau

# OpenMP program

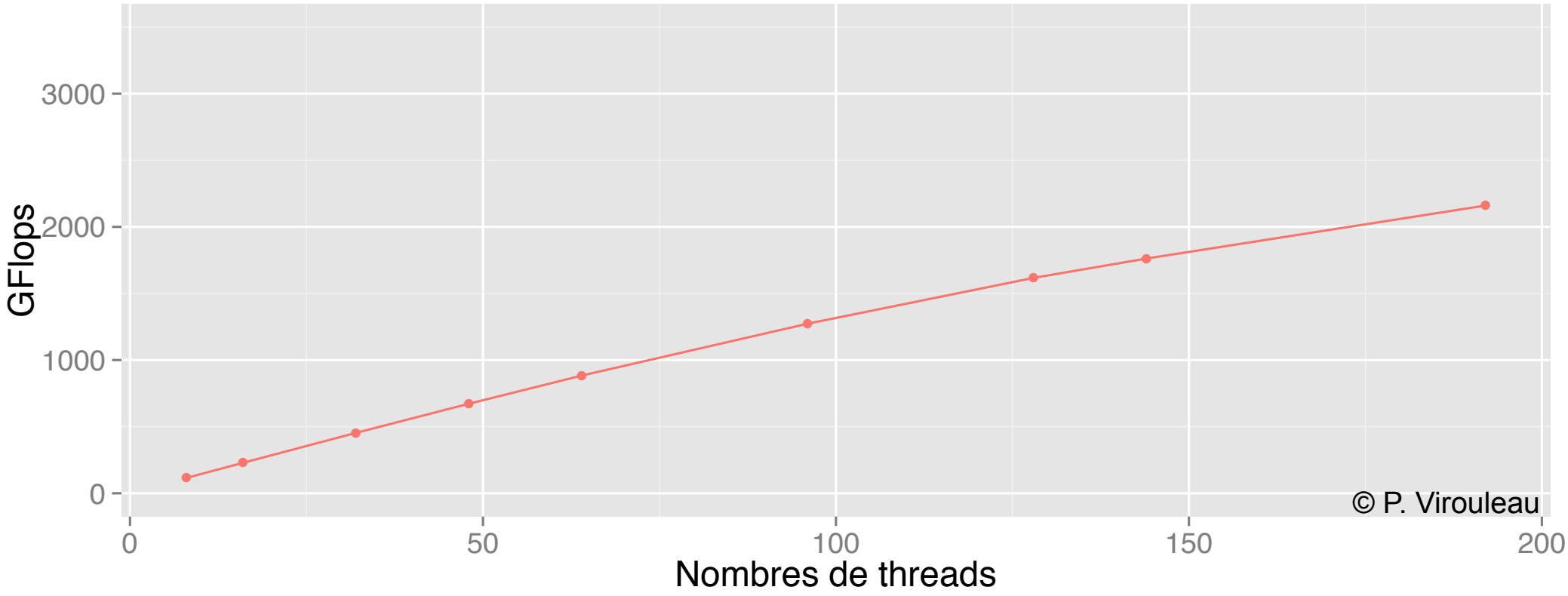
```
#include <cblas.h>
#include <clapack.h>

void Cholesky( int N, double A[N][N], size_t NB )
{
  for (size_t k=0; k < N; k += NB)
  {
#pragma omp task depend(inout: A[k:NB][k:NB]) shared(A)
    clapack_dpotrf( CblasRowMajor, CblasLower, NB, &A[k*N+k], N );

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: A[k:NB][k:NB]) \
                 depend(inout: A[m:NB][k:NB]) shared(A)
      cblas_dtrsm ( CblasRowMajor, CblasLeft, CblasLower, CblasNoTrans, CblasUnit,
        NB, NB, 1., &A[k*N+k], N, &A[m*N+k], N );
    }

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: A[m:NB][k:NB]) \
                 depend(inout: A[m:NB][m:NB]) shared(A)
     cblas_dsyrk ( CblasRowMajor, CblasLower, CblasNoTrans,
        NB, NB, -1.0, &A[m*N+k], N, 1.0, &A[m*N+m], N );

      for (size_t n=k+NB; n < m; n += NB)
      {
#pragma omp task depend(in: A[m:NB][k:NB], A[n:NB][k:NB])\
                 depend(inout: A[m:NB][n:NB]) shared(A)
        cblas_dgemm ( CblasRowMajor, CblasNoTrans, CblasTrans,
          NB, NB, NB, -1.0, &A[m*N+k], N, &A[n*N+k], N, 1.0, &A[m*N+n], N );
      }
    }
  }
#pragma omp taskwait
}
```



Graph for N=.., NB=..

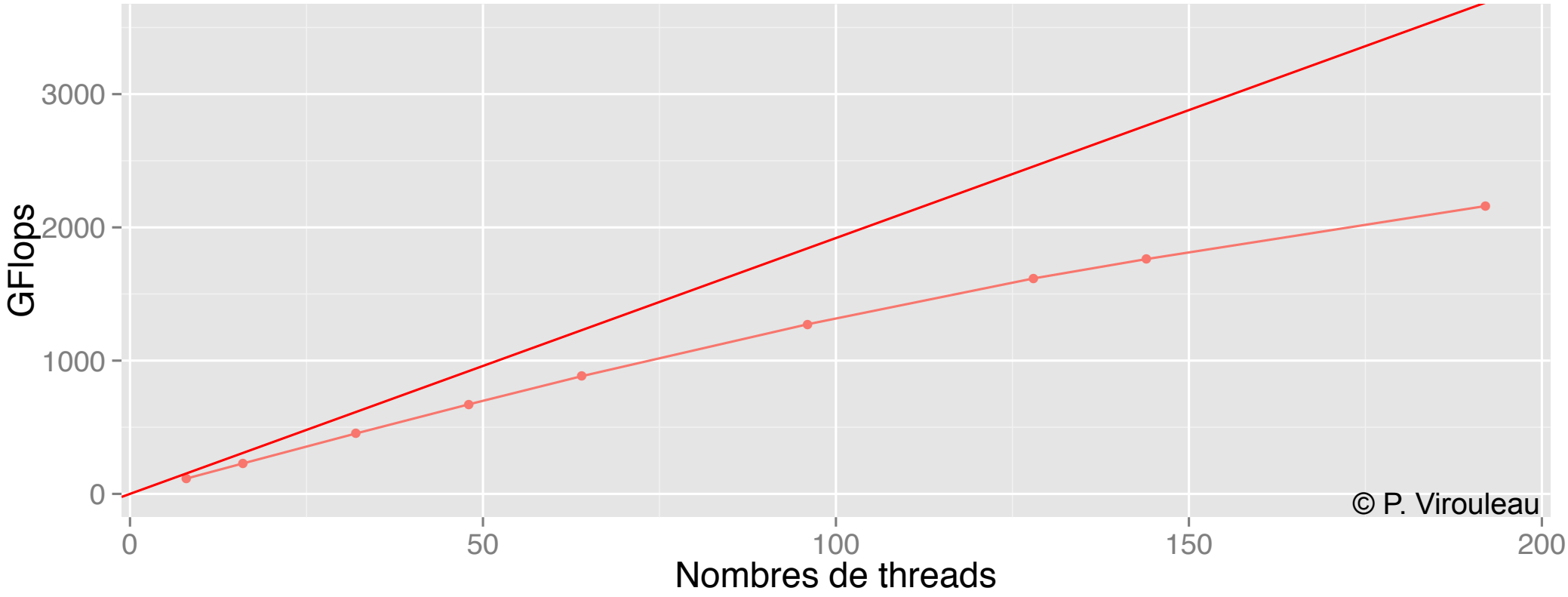
# Performances ?

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- GCC 6.3



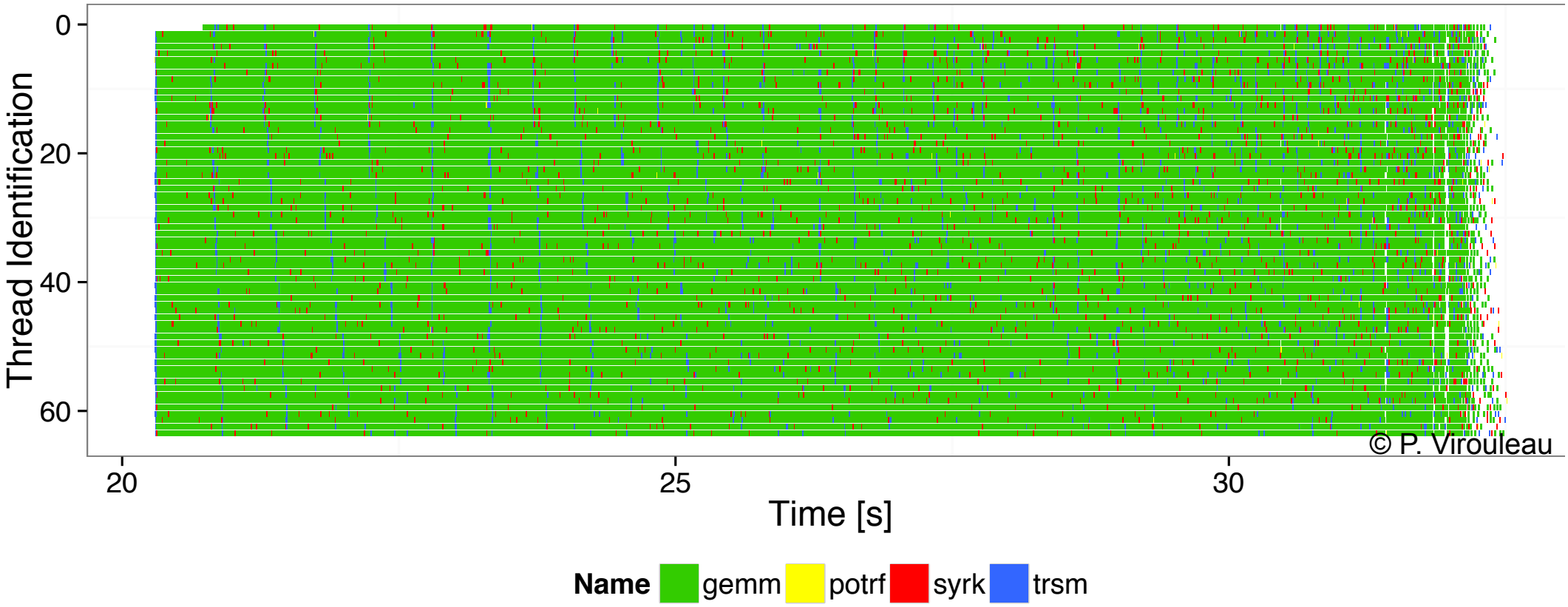
# Performances ?

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- GCC 6.3



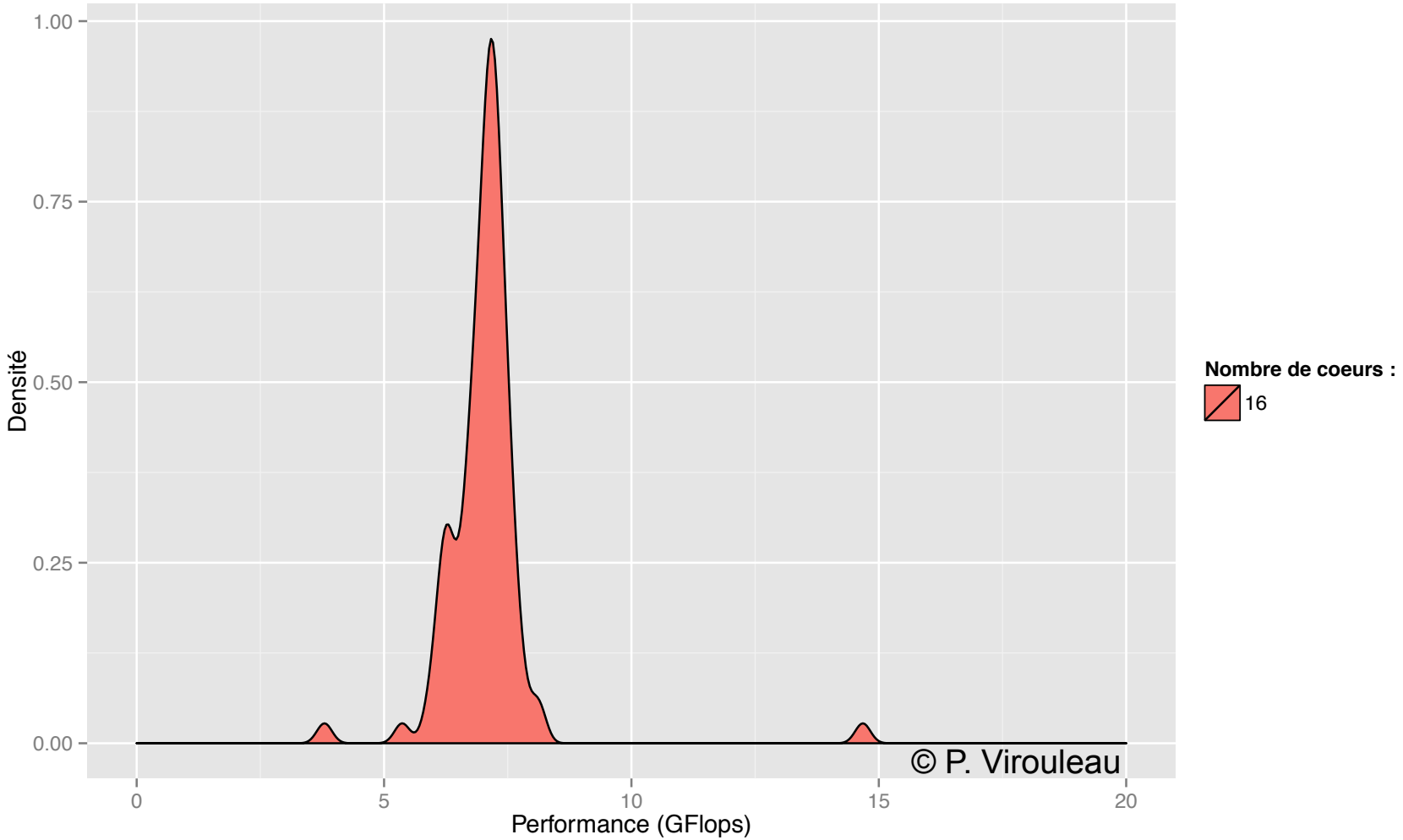
# Gantt

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- GCC 6.3
- 64 cores



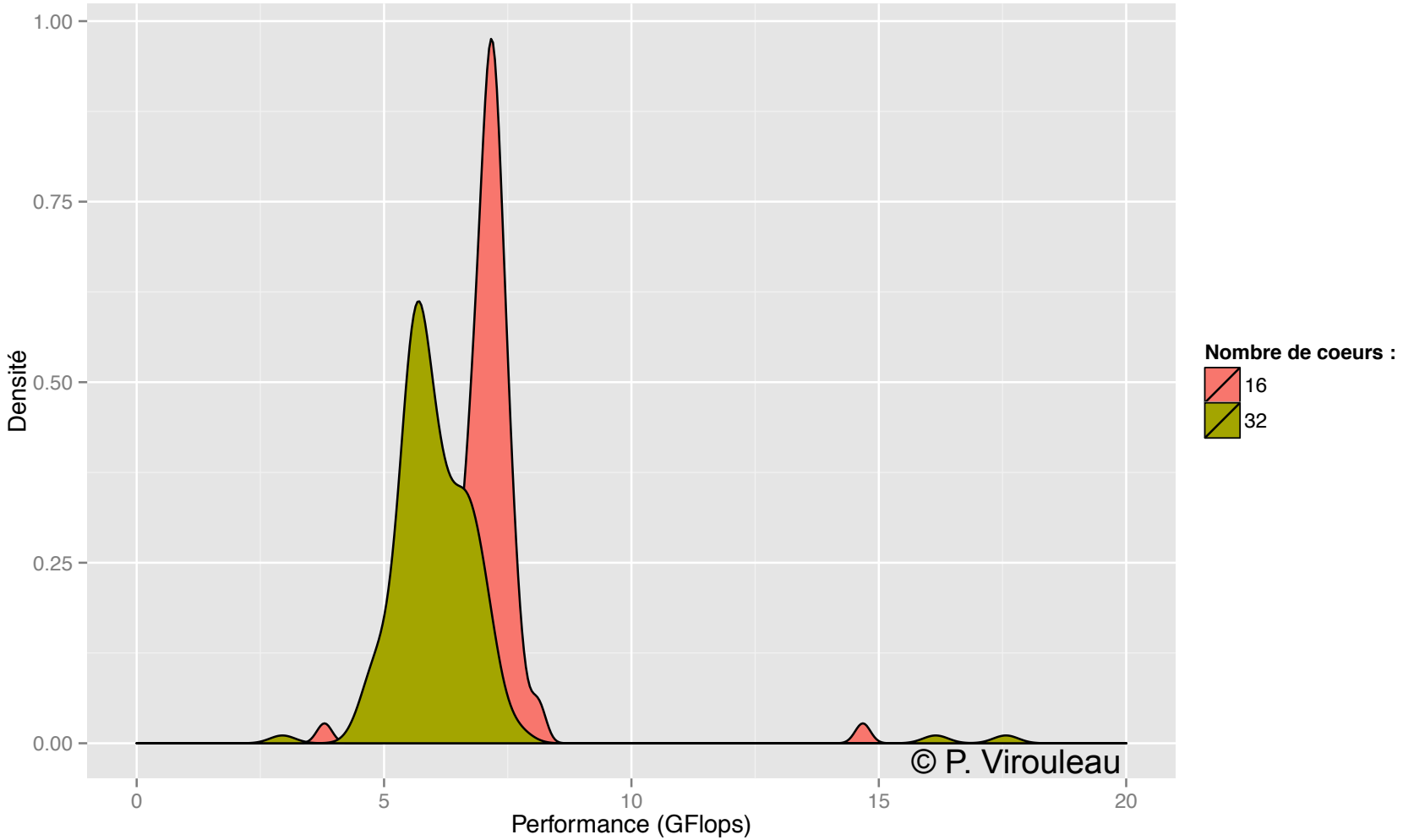
# Distribution des performances

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- Distribution des performances des kernels ‘DPOTRF’



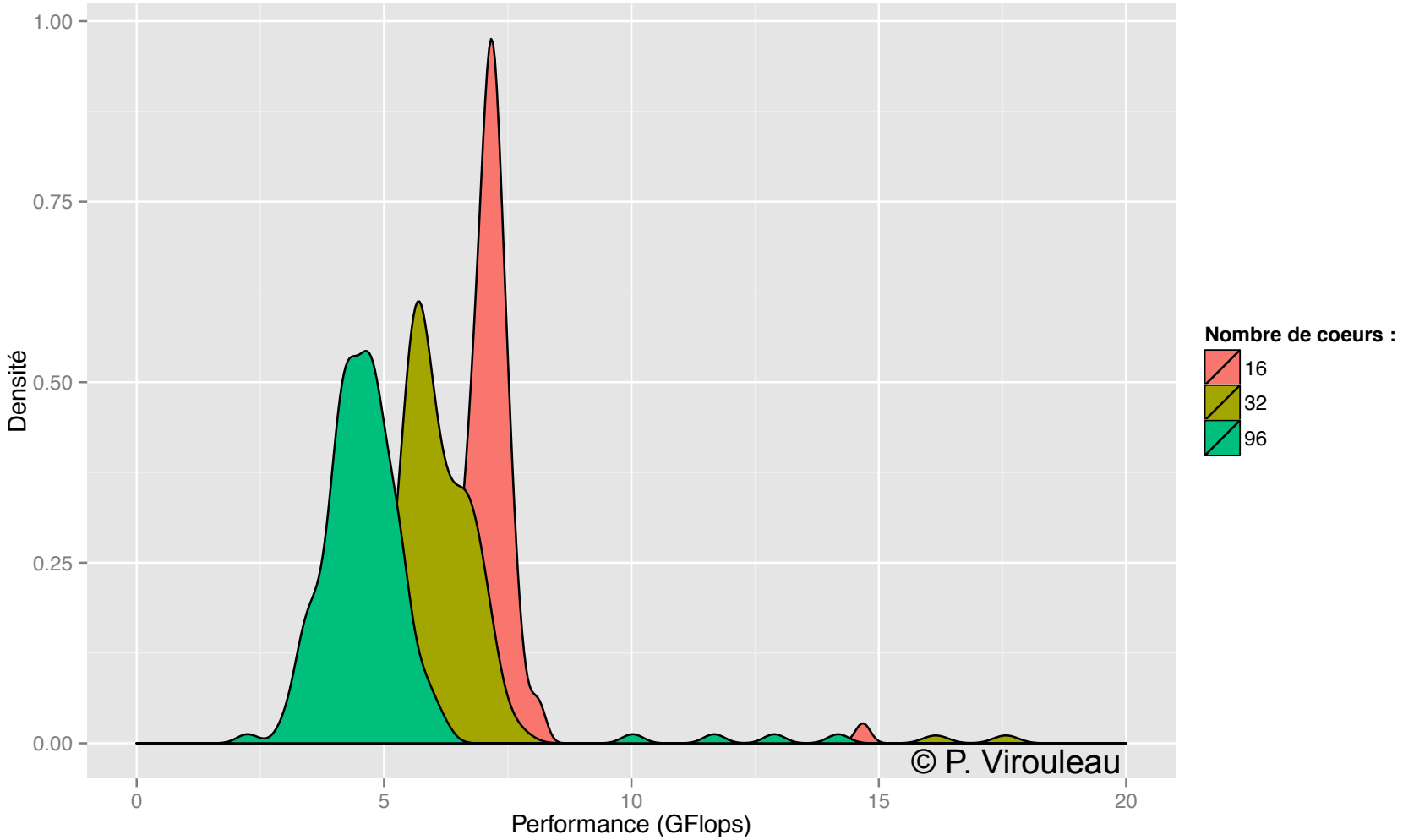
# Distribution des performances

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- Distribution des performances des kernels ‘DPOTRF’



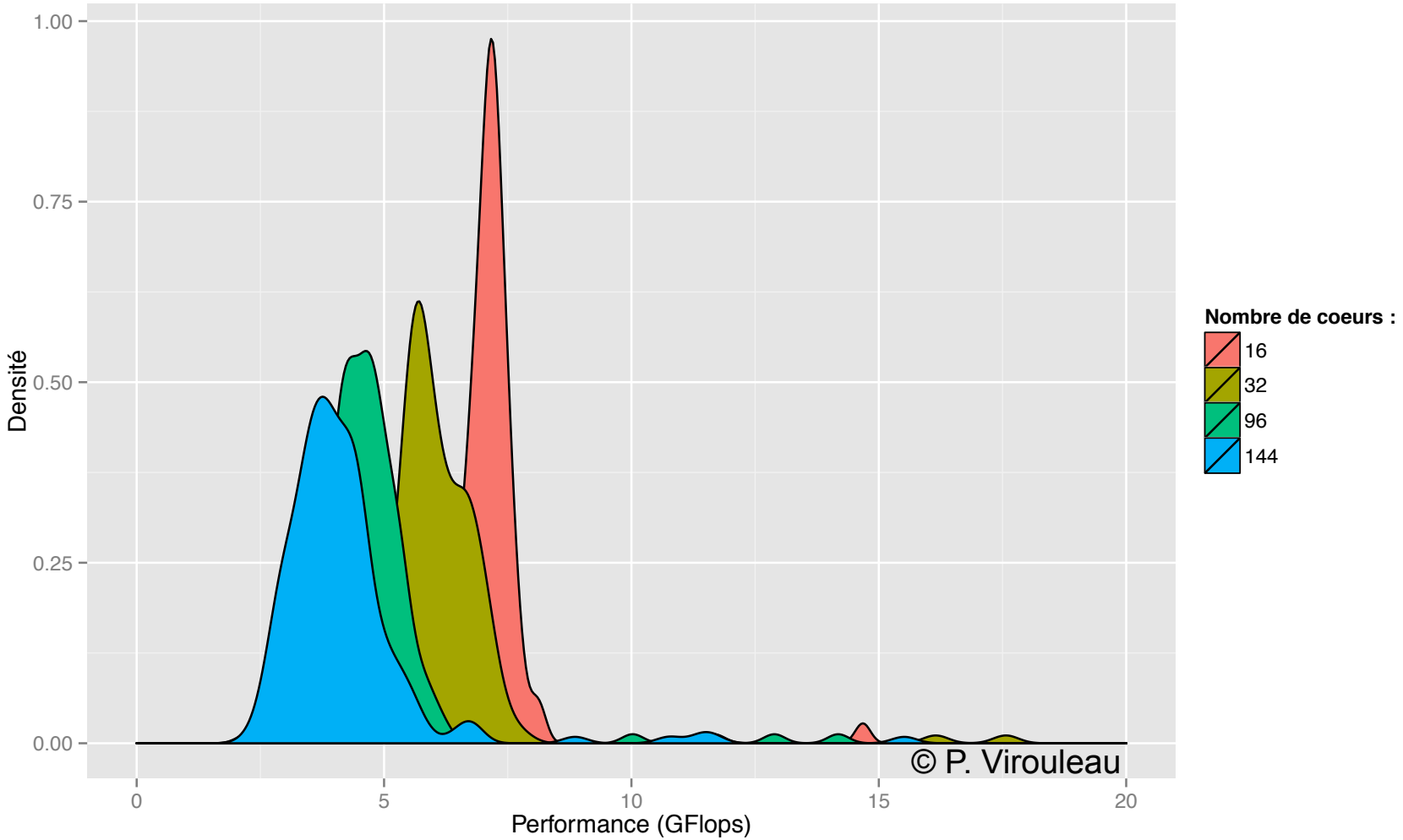
# Distribution des performances

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- Distribution des performances des kernels ‘DPOTRF’



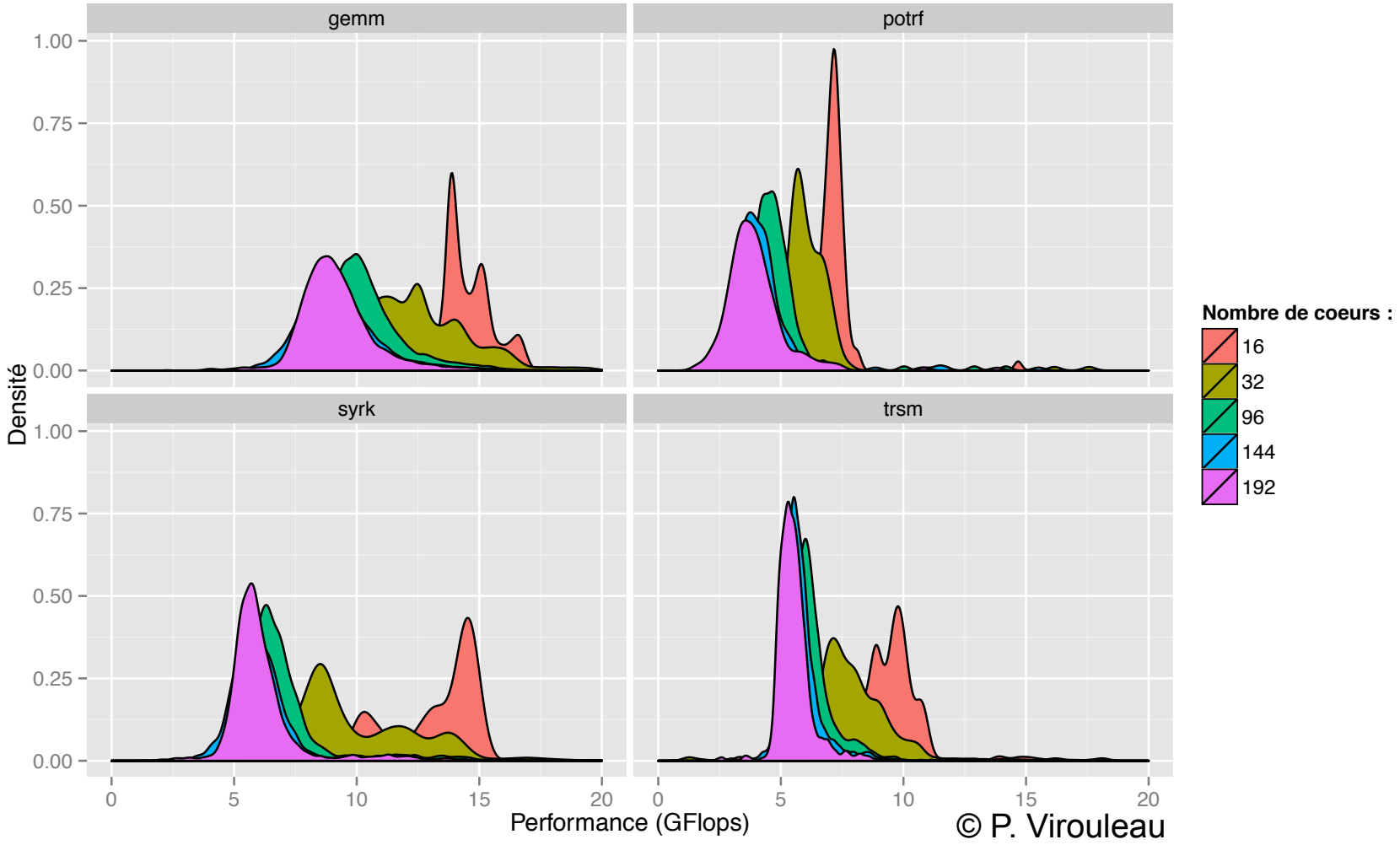
# Distribution des performances

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- Distribution des performances des kernels ‘DPOTRF’



# Distribution des performances

- idchire, UV3000, 192 cores, 24 bancs NUMA
- Matrice N=32768, BS=512, OpenBLAS
- Tous les kernels



# Explication : *work inflation*

- idchire, UV3000, 192 cores, 24 bancs NUMA

- Coût des calculs dépendant de :
  - localisation des données
  - charges des liens mémoires induits par l'activités de tous les cœurs actifs

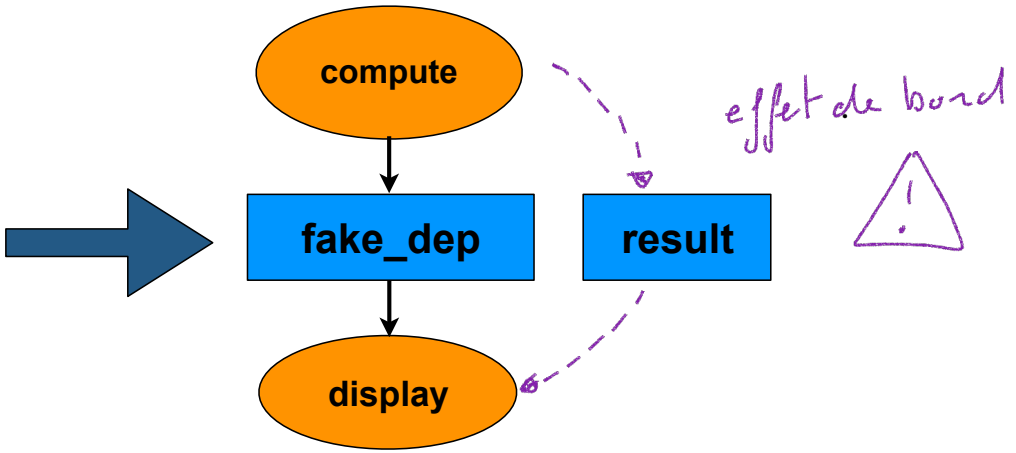
- Réduire le « work inflation »
  - une bonne distribution des données
  - des accès plus locaux

© P. Virouleau

# Amélioration de la gestion de la localité

- **Objectif: contrôler le placement des tâches / localisations des données**
- **OpenMP**
  - **description des dépendances entre tâches ≠ description des accès mémoires**

```
void main()
{
 #pragma omp parallel
 #pragma omp master
 {
   int fake_dep;
 #pragma omp task depend(out: fake_dep)
   compute( size, result );

 #pragma omp task depend(in: fake_dep)
   display( size, result );
 }
}
```




- **Ajout d'une clause affinité**

```
…
   int fake_dep;
 #pragma omp task depend(out: fake_dep) \
                 affinity(data: result)
   compute( size, result );

 #pragma omp task depend(in: fake_dep) \
                 affinity(data: result)
   display( size, result );
…
```

# Types d'affinité considérés

```
#pragma omp task affinity( [data | thread | node | data]: <expr> [, strict])
<structured block>
```

| | expr |
|---|---|
| **data** | adresse mémoire |
| **thread** | indice du thread dans la région parallèle |
| **node** | indice du i-ème nœud NUMA utilisé par les threads de la région parallèle |

# Implémentation

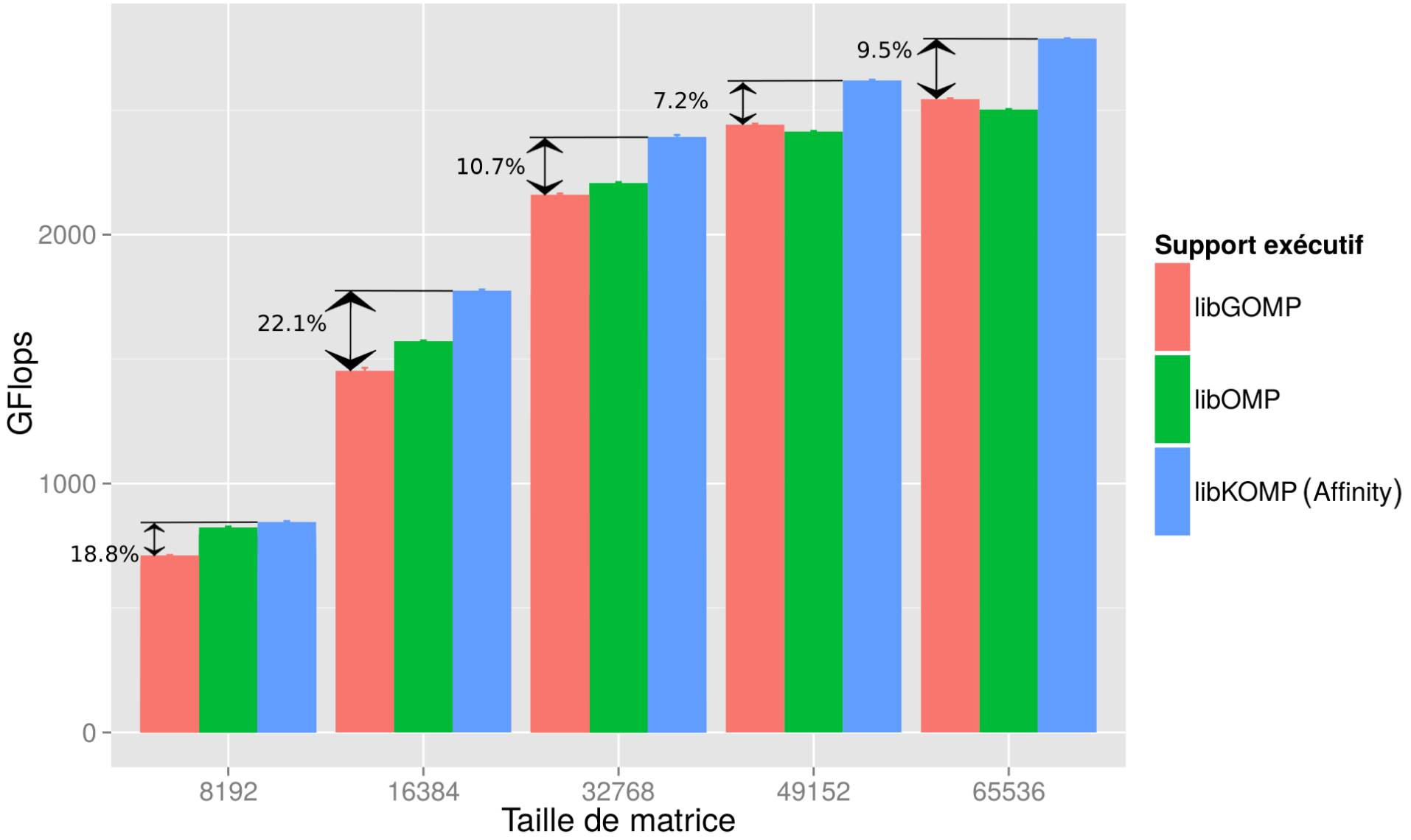
- **Intégration dans XKaapi (v0)**
  - **validation de l'approche**

- **Intégration dans le runtime LLVM OpenMP : libKOMP**
  - **définition des structures de queues hiérarchiques**
    - ajout d'une queue de tâches prêtes par nœud NUMA

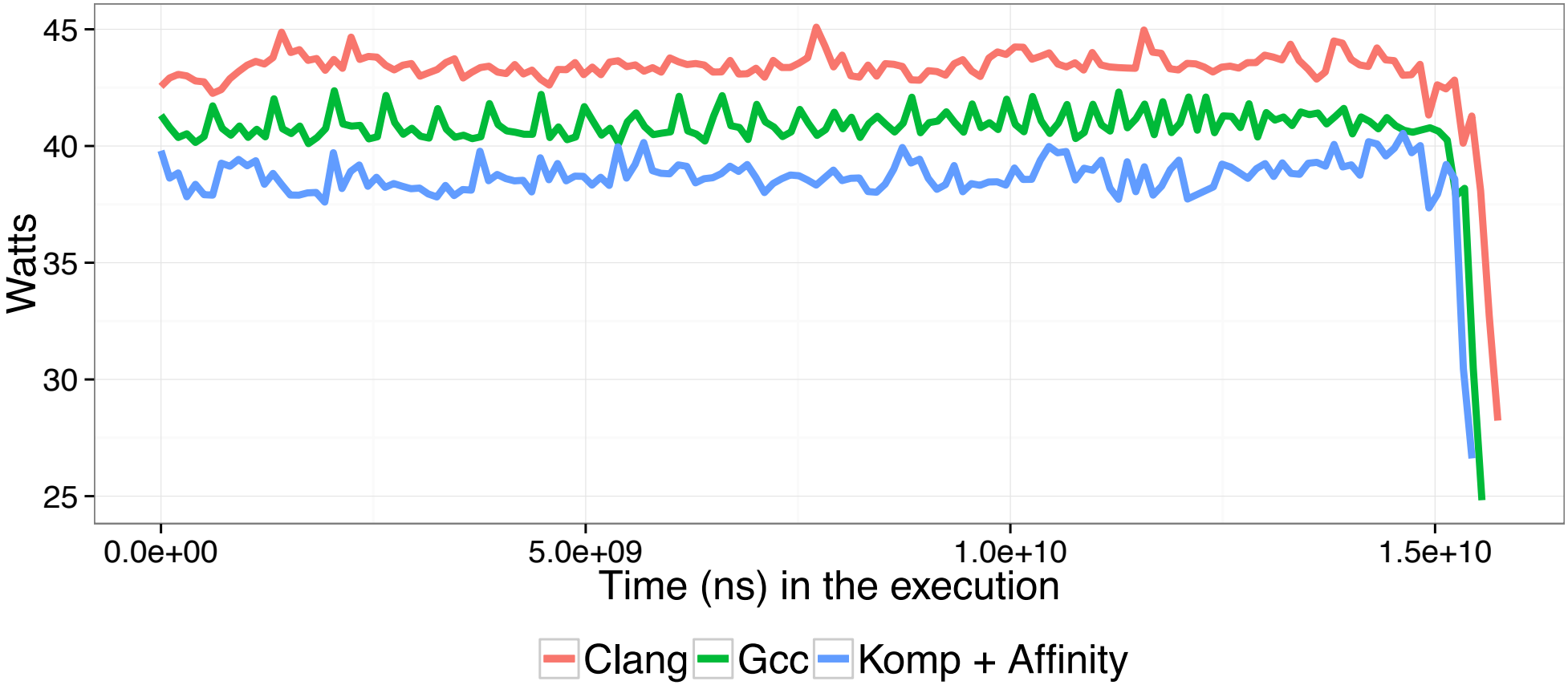
- **Très simple car ordonnancement = vol de travail**
  - **détournement de la fonction qui pousse sur la bonne queue une tâche activée**
    - si contrainte d'affinité : calcul du nœud NUMA contenant la donnée et « enqueue »
    - sinon on pousse la tâche dans la queue locale

- **Moins simple**
  - **évaluation et construction d'une bonne heuristique parmi les choix possibles**
  - **PhD P. Virouleau**

# Gain en performance (Cholesky)



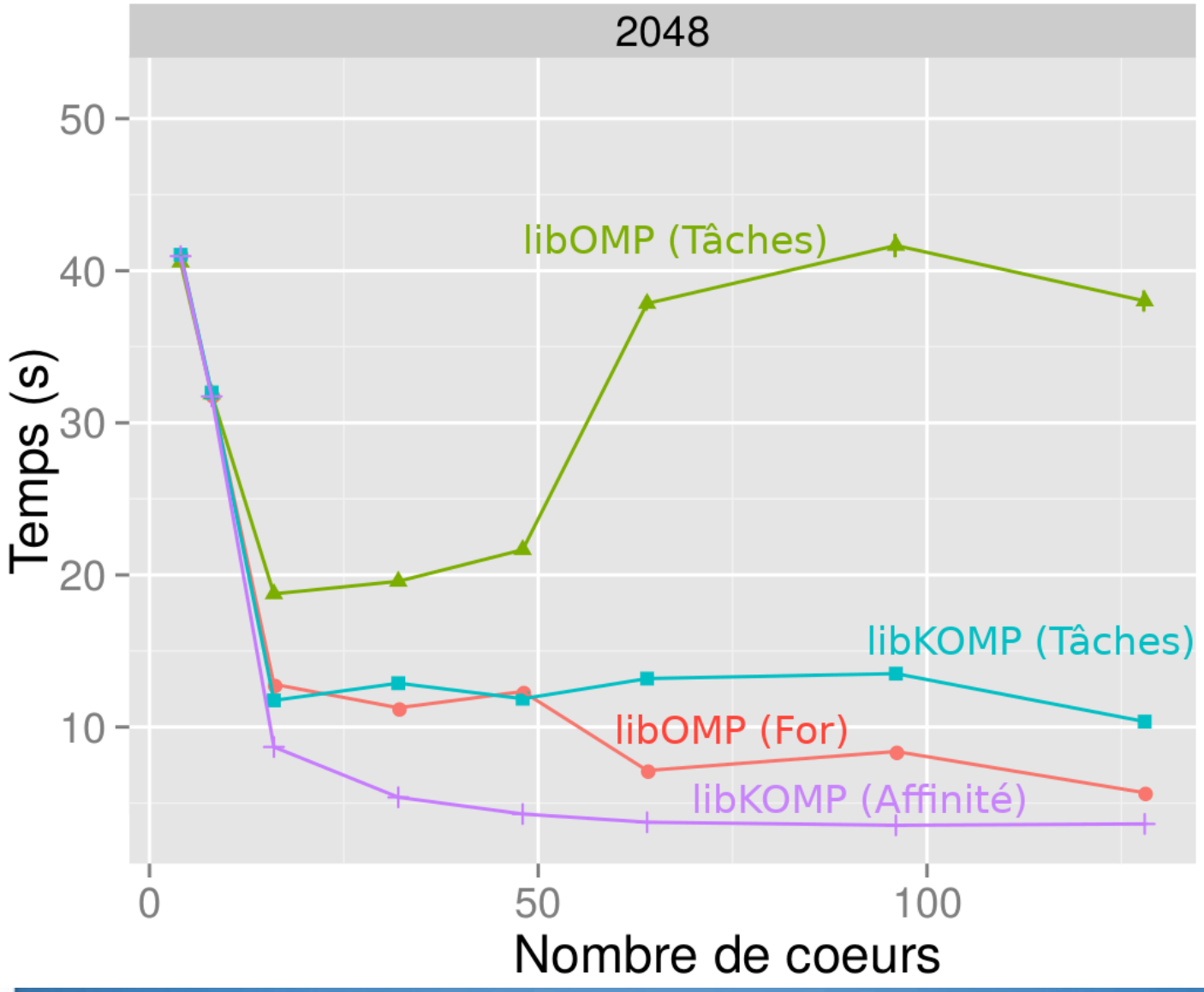
# Impact sur l'énergie 'DRAM'




- ~ 10% de gain en énergie consommée par la DRAM

# Gain sur des applications itérative

- 2D stencil, see Kastors benchmark suite http://kastors.gforge.inria.fr



# Bilan

- **Programmation par tâche**
  - **souplesse car « séparation of concerns »**
    - programme = description d'un enchainement de tâches avec dépendances
    - scheduler = placement des tâches sur les ressources

- **Programmation sur architecture NUMA**
  - **vital : distribution des données**
    - numactl
    - tâches + affinité

- **Extension OpenMP**
  - **présentée comme une *extension de clause* => extension du compilateur (CLANG)**
  - **mais possible en tant qu'appel à des fonctions de runtime OpenMP (« omp.h »)**
    - omp_set_task_affinity( type, expr, strictness)
    - portable avec ICC, GCC, CLANG, iFort et GFortran !

# libKOMP : Runtime modifié

- **Basé sur le runtime LLVM OpenMP, fork du runtime d'Intel**
- **libKOMP: https://gitlab.inria.fr/openmp/libkomp**
  - **T.H.E Queue *à la Cilk***
  - **Protocole d'agrégation des requêtes de vol à la XKaapi**
  - **Optimisation pour la gestion des dépendances**
    - hash table dynamiquement adaptable
    - gestion de l'affinité
  - **Extension: affinité, écriture concurrente**

  - **module OMPT de capture de la trace d'exécution**
    - enregistrement des événements OpenMP (début-fin de tâche, …)
    - enregistrement de compteurs de performance PAPI
    - R comme language de script pour le traitement et l'affichage

# Questions ?

- Pause ?

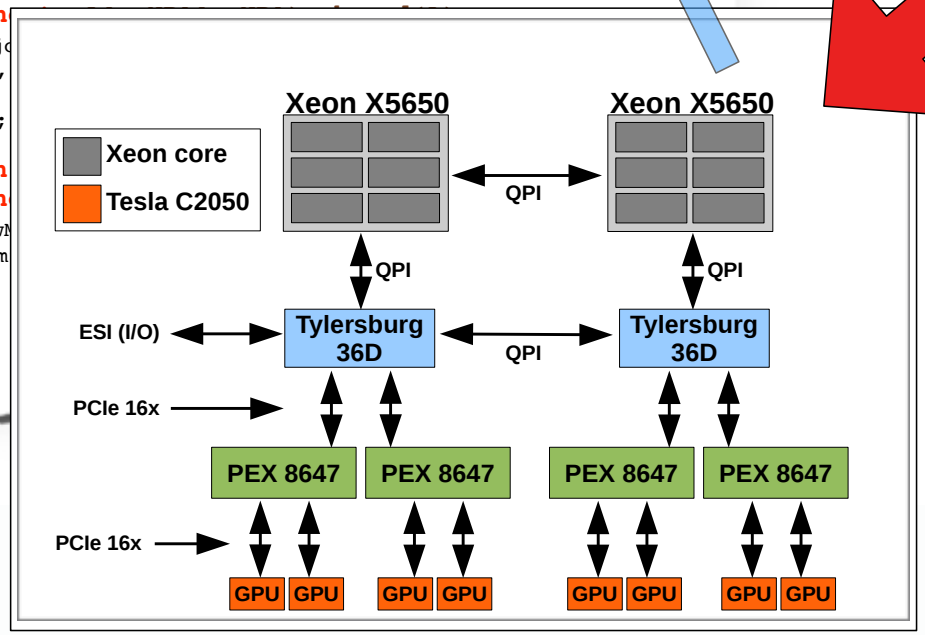
# Kaapi

## Un modèle de calcul sur multi-GPUs et multi-CPUs
## Travail effectué avec J. Lima, UFSM, Brazil

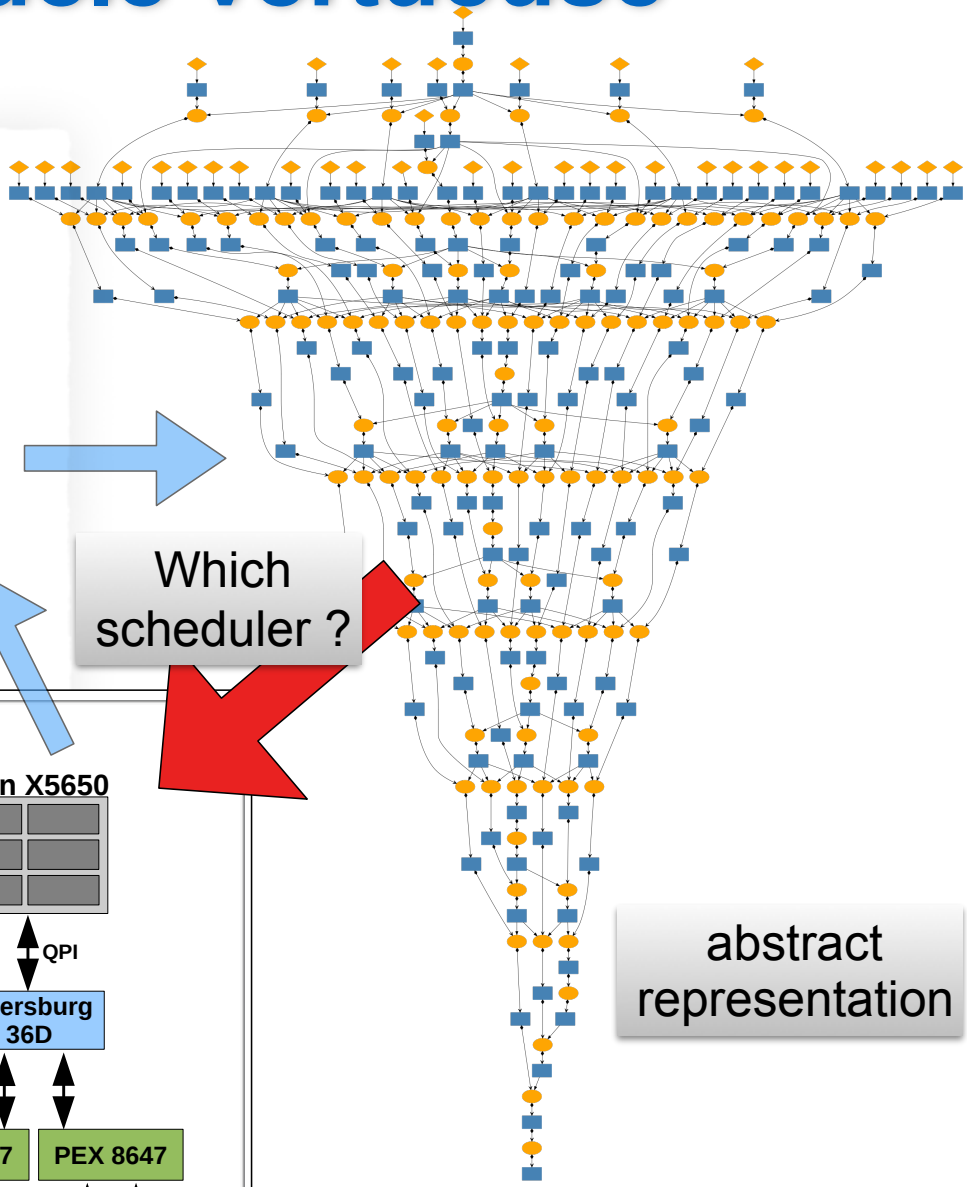
# Réalisation de la boucle vertueuse

Application:
how to program it ?

```
#include <cblas.h>
#include <clapack.h>

void Cholesky( int N, double A[N][N], size_t NB )
{
  for (size_t k=0; k < N; k += NB)
  {
#pragma omp task depend(inout: A[k:NB][k:NB]) shared(A)
    clapack_dpotrf( CblasRowMajor, CblasLower, NB, &A[k*N+k], N );

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: A[k:NB][k:NB]) \
                 depend(inout: A[m:NB][k:NB]) shared(A)
      cblas_dtrsm ( CblasRowMajor, CblasLeft, CblasLower, CblasNoTrans, CblasUnit,
        NB, NB, 1., &A[k*N+k], N, &A[m*N+k], N );
    }

    for (size_t m=k+ NB; m < N; m += NB)
    {
#pragma omp task depend(in: A[m:NB][k:NB]) \
                 depend(in
     cblas_dsyrk ( CblasRowMajo
        NB, NB, -1.0, &A[m*N+k],

      for (size_t n=k+NB; n < m;
      {
#pragma omp task depend(in
                 depend(in
        cblas_dgemm ( CblasRowM
          NB, NB, NB, -1.0, &A[m
      }
    }
  }
#pragma omp taskwait
}
```




Which scheduler ?



abstract representation

# Objectif

- **Représentation de l'exécution = graphe de flot de données**
  - **graphe orienté bi-parti, nœuds « tâche » et nœuds « donnée »**

- **Scheduler du graphe de tâches**
  - **choix en cours d'exécution des ressources utilisées**
    - meilleure adaptation de la charge de calcul en cours d'exécution

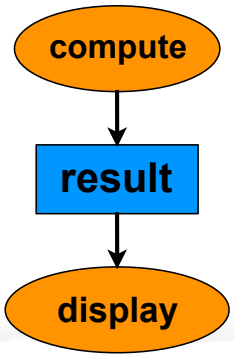
  **➡ une même tâche doit avoir plusieurs implémentations CPU ouGPU**

- **Comment écrire un tel programme ?**
  - **[question posée en 2008]: OpenMP ? Non car pas de dépendances entre tâche**
  - **en 2018 ? OpenMP : non car pas possible d'avoir facilement plusieurs implémentation par tâches + effet de bord non décrit. Extension des targets envisageable.**

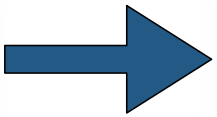
  **➡ Kaapi [Europar 2010, IPDPS 2013, PARCO 2015]**

# Création de tâche en KAAPI

- **3 concepts fondamentaux**
  - **Mode d'accès : lecture (R), écriture (W), écriture concurrente (CW), postponed (P)**
  - **Signature d'une tâche ~ prototype + modes accès aux paramètres formels**
  - **Implémentation d'une tâche : CPU ou GPU (non exclusif)**
    - hypothèse pour la portabilité : il existe toujours une implémentation valide
- **Tâche**
  - **pas d'effet de bord**
  - **description des accès à la mémoire par les *modes d'accès***
- **Création de tâche == spawn (à la Cilk !)**
  - **opération non bloquante**
- **Exemple**




```
void main()
{
  /* data result is produced */
  compute( size, result );

   /* data result is consumed */
  display( size, result );
}
```



```
void main()
{
...
  ka::Spawn<TaskCompute>()( size, &result );

  ka::Spawn<TaskDisplay>()( size, &result );
  ka::Sync()
}
```

# If you know…

| | | Kaapi equiv. |
|---|---|---|
| OpenMP | #pragma omp task<br>F(arg) | ka::Spawn<F>()(arg)<br>+ déclaration de F |
| | #pragma omp task depend(…)<br>F(arg) | ka::Spawn<F>()(arg)<br>+ déclaration de F |
| | #pragma omp taskwait | ka::Sync()<br>Mais sémantique + forte |
| Cilk | cilk_spawn F(arg) | ka::Spawn<F>()(arg) |
| | cilk_sync | ka::Sync() |

# Définition des tâches

Signatures

```
#include <kaapi++h>

struct TaskCompute : public ka::Task<2>::Signature<
  int,                          /* size */
  ka::W<ka::array<1,int> >  /* result W == WRITE*/
> {};

struct TaskDisplay : public ka::Task<2>::Signature<
  int,                          /* size */
  ka::R<ka::array<1,int> >  /* R == READ access to result */
> {};
```

```
template<>
struct TaskBodyCPU<TaskCompute> {
  void operator() ( int size, ka::pointer_w<ka::array<1,double> > result )
  {
    int i;
    for (i=0; i<size; ++i)
     result[i] = (double)(i*i);
  }
};

template<>
struct TaskBodyCPU<TaskDisplay> {
  void operator() ( int size, ka::pointer_r<ka::array<1,double> > result )
  {
    int i;
    printf("Result is:");
    for (i=0; i<size; ++i)
      printf(" %f", result[i]);
    printf("\n");
  }
};
```

Implémentation **CPU**

# Définition des tâches

Implémentation **GPU**

```
template<>
struct TaskBodyGPU<TaskCompute> {
  void operator() ( ka::gpuStream stream,
                    int size, ka::pointer_w<ka::array<1,double> > result )
  {
    float* const d_R = result->ptr();
    dim3 threads(…);
    dim3 grid(…);
    squareKernel<<<grid, threads, 0, stream>>>(size, d_R);

};
```

# Restriction

- Soit e un paramètre effectif passé par référence == ka::pointer_XX
- Soit f un paramètre formel de type ka::pointer_YY
- Seuls les cas suivants sont autorisés

| effective \ formal | pointer_r | pointer_rw | pointer_w | pointer_cw |
|---|---|---|---|---|
| pointer | yes | yes | yes | yes |
| pointer_r | yes | no | no | no |
| pointer_rw | no | no | no | no |
| pointer_w | no | no | no | no |
| pointer_cw | no | no | no | yes |

# Pourquoi ?

- **Conserver une sémantique lexicographique**
  - identification des écrivains des valeurs lues
- **Déterminisme**
  - Une donnée lue par une tâche ne sera jamais modifier par une tâche fille
- **Ordonnancement non préemptif des tâches**
  - une tâche ne s'arrêtera jamais en cours d'exécution pour attendre la production d'un paramètre

# + mode postponed

| formal \ effective | pointer_r | pointer_rp | pointer_rw | pointer_rpwp | pointer_w | pointer_wp |
|---|---|---|---|---|---|---|
| pointer | yes | yes | yes | yes | yes | yes |
| pointer_r | yes | yes | no | no | no | no |
| pointer_rp | yes | yes | no | no | no | no |
| pointer_rw | no | no | no | no | no | no |
| pointer_rpwp | yes | yes | yes | yes | yes | yes |
| pointer_w | no | no | no | no | no | no |
| pointer_wp | no | no | no | no | yes | yes |

# Multi-GPUs Cholesky Factorisation

```
for(k= 0; k < N; k+= blocsize){
  ka::Spawn<TaskPOTRF>()( A(rk,rk) );

  for(m= k+blocsize; m < N; m+= blocsize)
    ka::Spawn<TaskTRSM>()( A(rk,rk), A(rm,rk));

  for(m= k+blocsize; m < N; m+= blocsize){
    ka::Spawn<TaskSYRK>()( A(rm,rk), A(rm,rm));

    for(n= k+blocsize; n < m; n+= blocsize)
      ka::Spawn<TaskGEMM>()( A(rm,rk), A(rn,rk), A(rm,rn) );
  }
}
```

# Multi-versions des tâches

```
struct TaskSYRK: public ka::Task<2>::Signature<
  ka::R<ka::range2d<double> >,
  ka::RW<ka::range2d<double> > >{};
```

Signature

```
template <>
struct TaskBodyCPU<TaskSYRK>{
  void operator( ka::range2d_r<double> A, ka::range2d_rw<double> C )
  {
    /* CPU implementation */
    cblas_dsyrk( A->dim(0), A->dim(1), A->ptr(), A->ld(), C->ptr(), C->ld() );
  }
};
```

CPU Task

```
template <>
struct TaskBodyGPU<TaskSYRK>{
  void operator( ka::gpuStream stream,
                 ka::range2d_r<double> A, ka::range2d_rw<double> C)
  {
    /* GPU implementation */
    cublasDsyrk( kaapi_cublas_handle(stream),
                 A->dim(0), A->dim(1), A->ptr(), A->ld(), C->ptr(), C->ld() );
  }
};
```

No explicit synchronization!

GPU Task

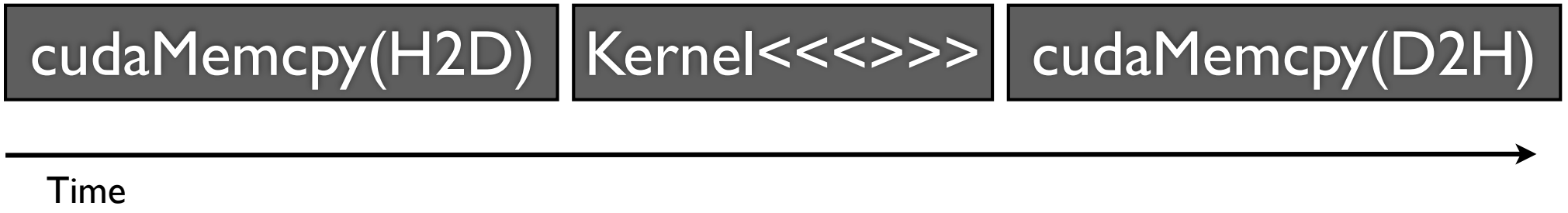
# Exécution d'une tâche sur une ressource

- **Le runtime NUMA gère**
  - **les exécutions asynchrones**
  - **ordonnancement par vol de travail**

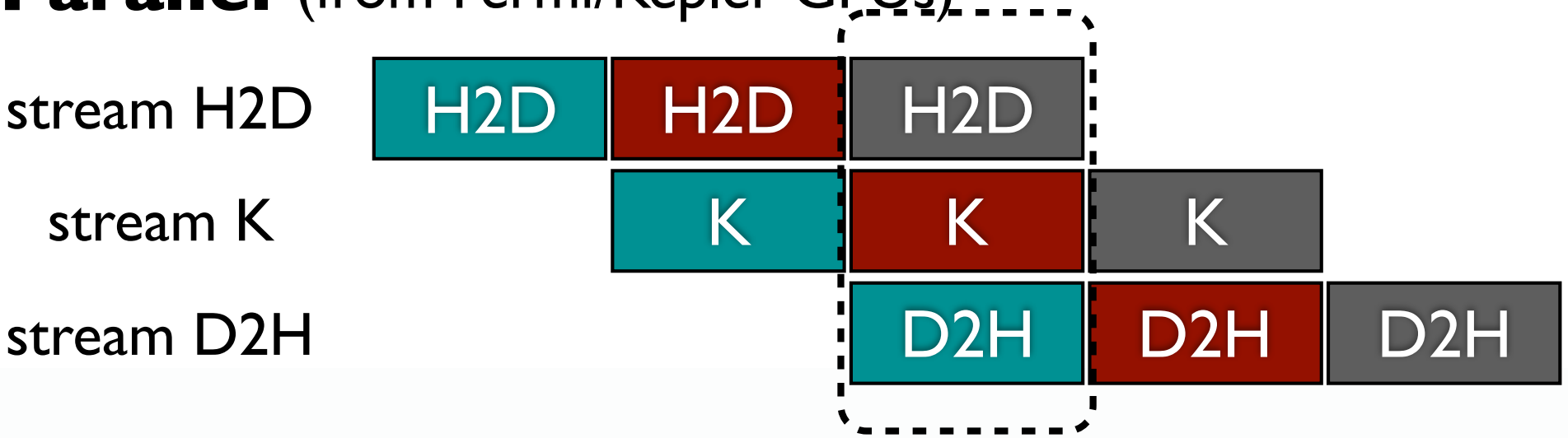
- **Les extensions nécessaires pour le multi-GPUs**
  - **gestion des communications des données entre les espaces mémoires**
    - cache logicel et gestion des réplicas
  - **gestion de la concurrence des opérations vers un GPU**
    - multi kernels, recouvrement calcul/communication

# Opérations GPU concurrentes

**Sequential**




**Parallel** (from Fermi/Kepler GPUs)

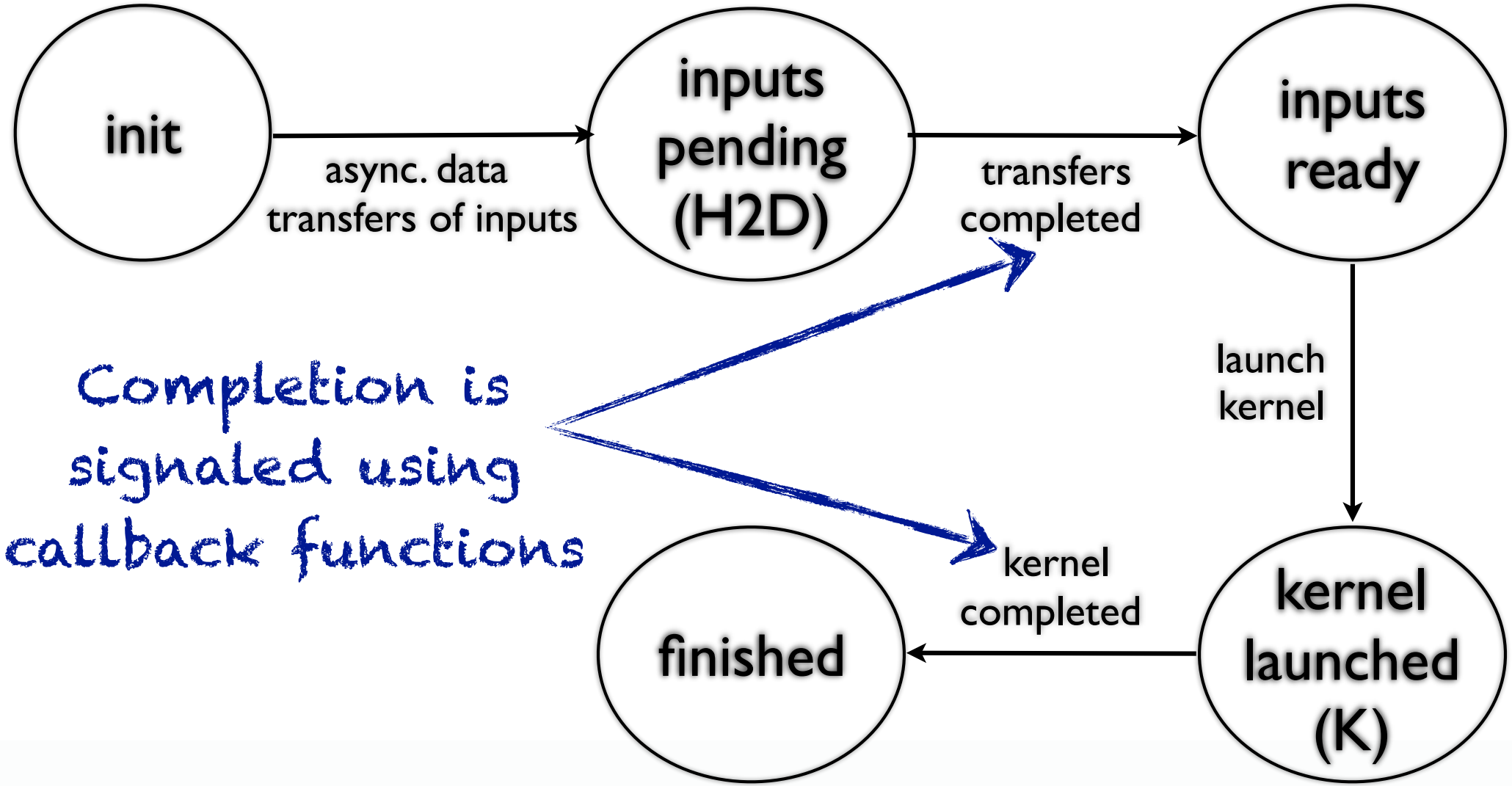


# Opérations GPU concurrentes

- **kstream**
  - **3 streams CUDA streams avec seulement des opérations asynchrones**
    - H2D == host to device transfers
    - K == kernel execution (implicitly launched by `TaskBodyGPU`)
    - D2H == device to host transfers
  - **Signalisation de la fin des opérations asynchrones par des fonctions callback**

- **Intérêt = ne pas utiliser explicitement des synchronisations**
  - **`cudaDeviceSynchronize(), cudaStreamSynchronize(), …`**

# Diagramme d'état des tâches GPU



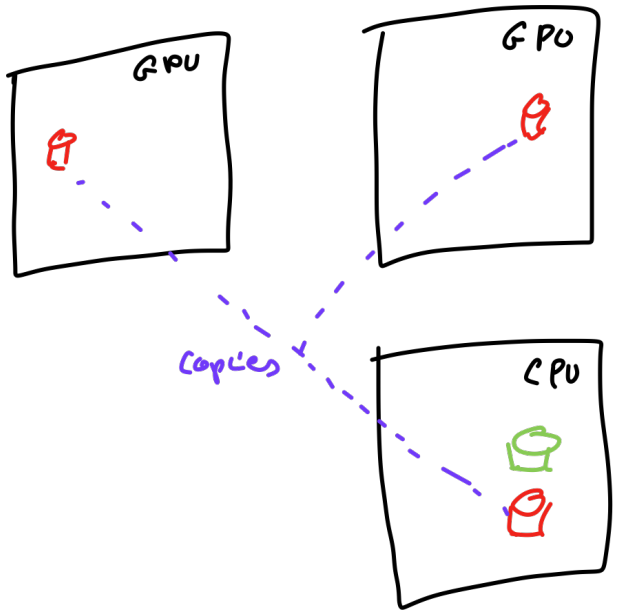
# Cache logiciel

- Gestion des réplicas entre les GPUs et la machine hôte
- Carte GPU ~ qques GBytes, mémoire hôte ~ 32GBytes -> 1TBytes

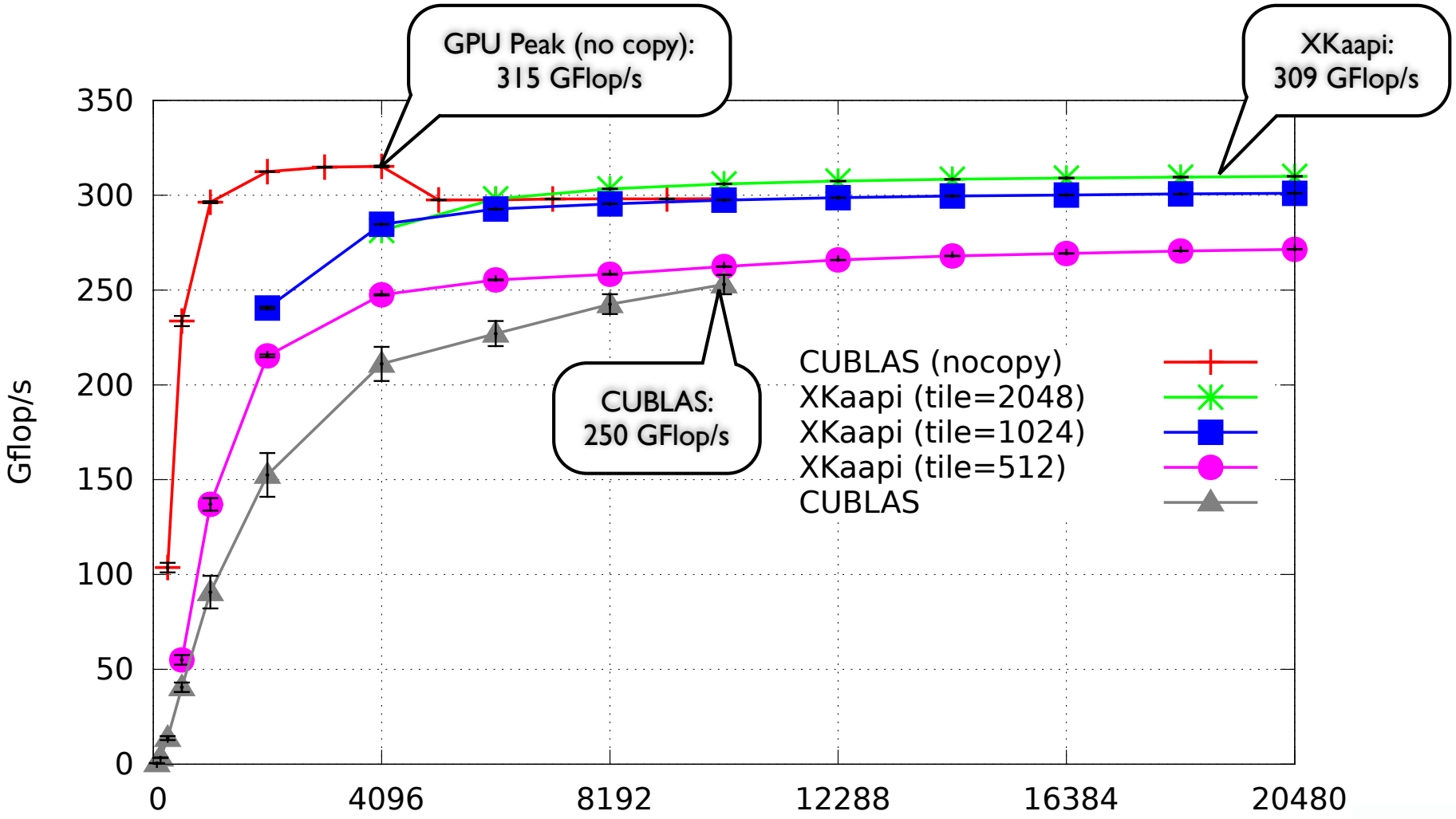
- Politique de remplacement
  - 2 queues Least Recently Used (LRU)
    - R only : éviction en premier
    - RW or W : si nécessaire

- Cohérence = write-back policy
  - Lazy strategy

- Transferts non bloquant
  - gérés par les 'kstream'



# Evaluation du Kstream sur un DGEMM

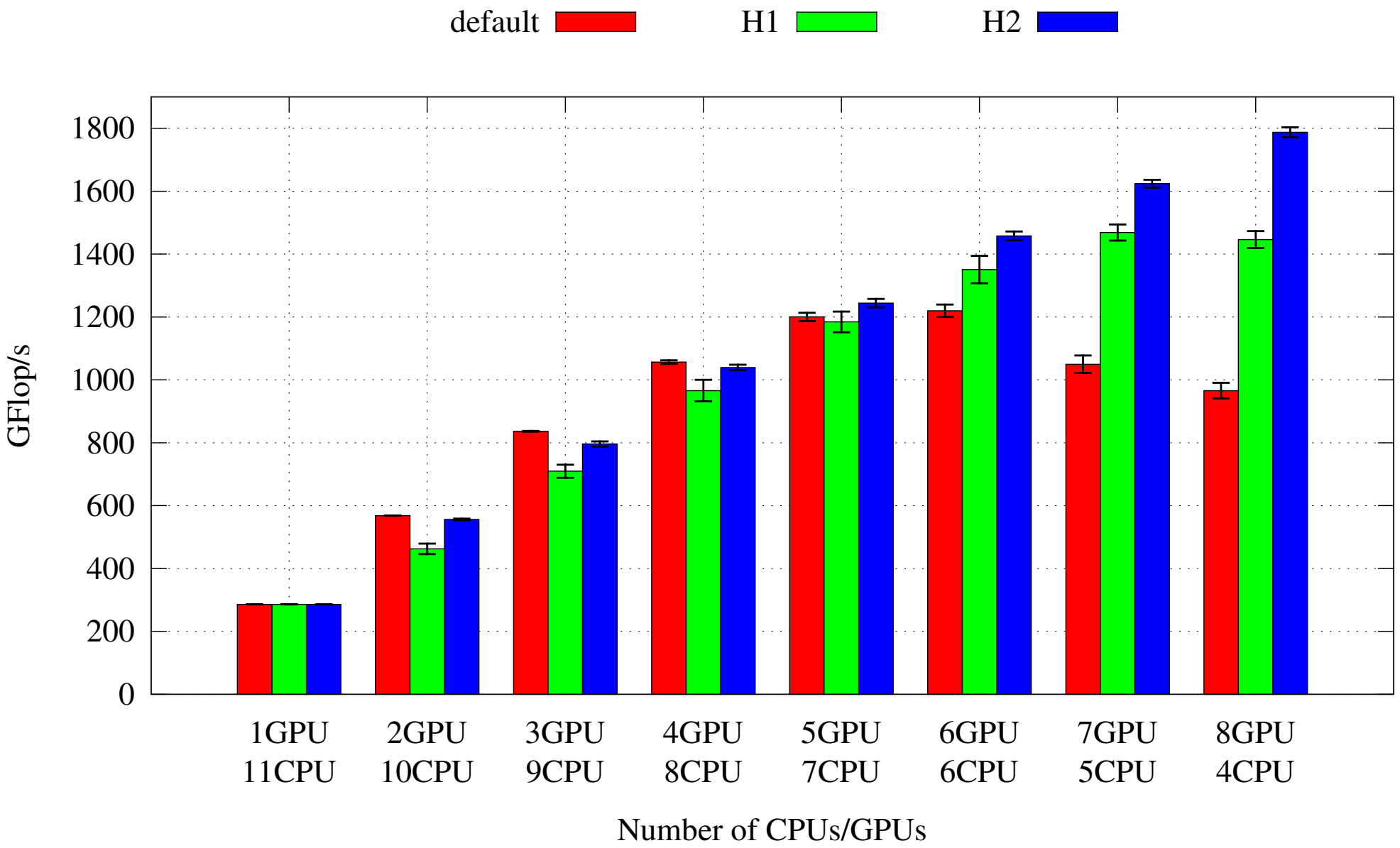



DGEMM Single-GPU Evaluation -M2050-
(Double Precision)

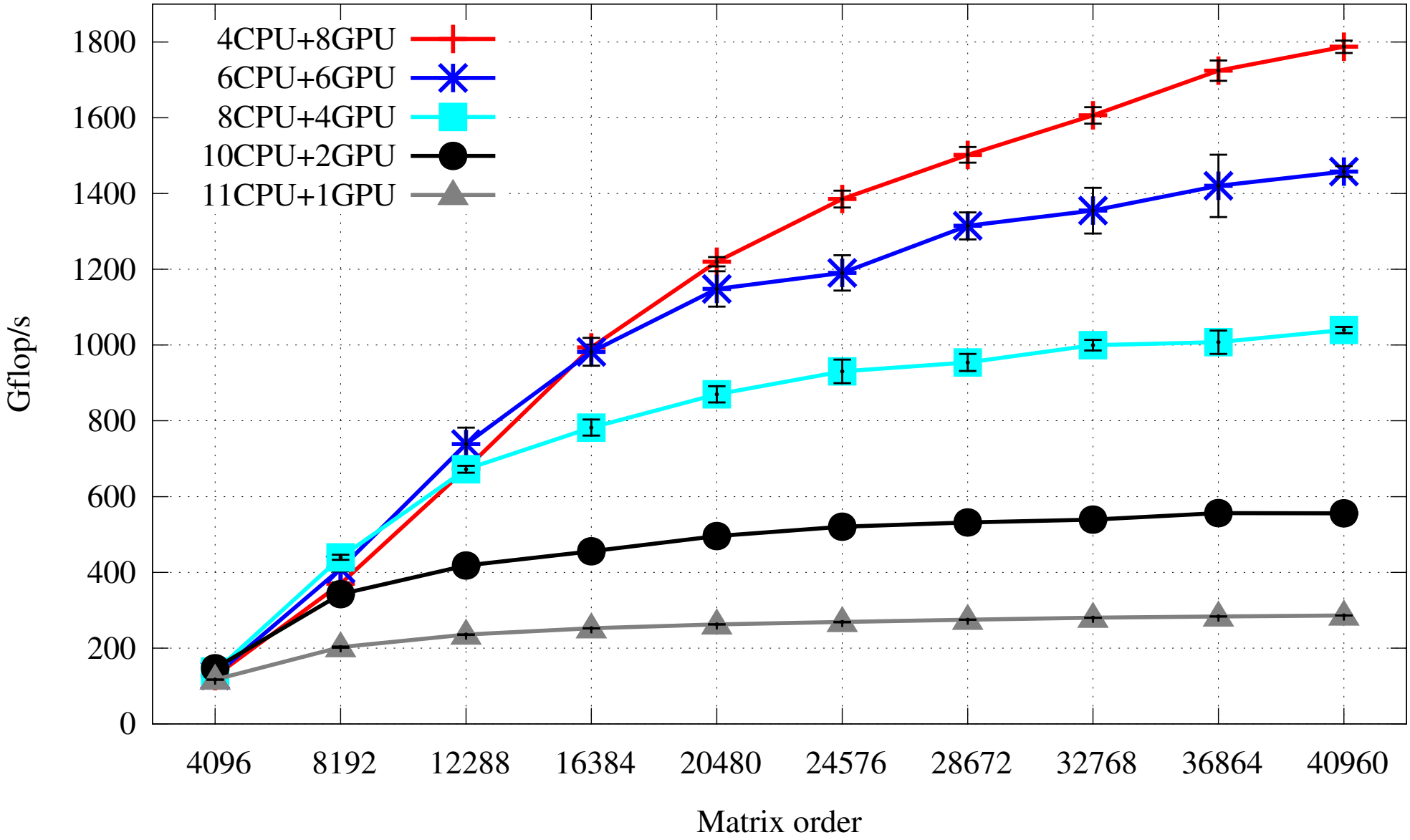
# Heuristiques de vol de travail

- 1 thread par ressource CPU ou GPU
  - 1 queue par ressource pour stocker les tâches prêtes
- default :
  - pas de gestion d'affinité données / tâches / ressources
- H1: GPU + gestion de l'affinité « en volume »
  - lorsqu'une tâche est prête elle est poussée sur la ressource qui stocke le plus de ses données (en volume)
- H2: GPU + gestion de l'affinité « OCR »
  - lorsqu'une tâche est prête elle est poussée sur la ressource qui stocke l'une de ses données en écriture
  - OCR = Owner Compute Rule

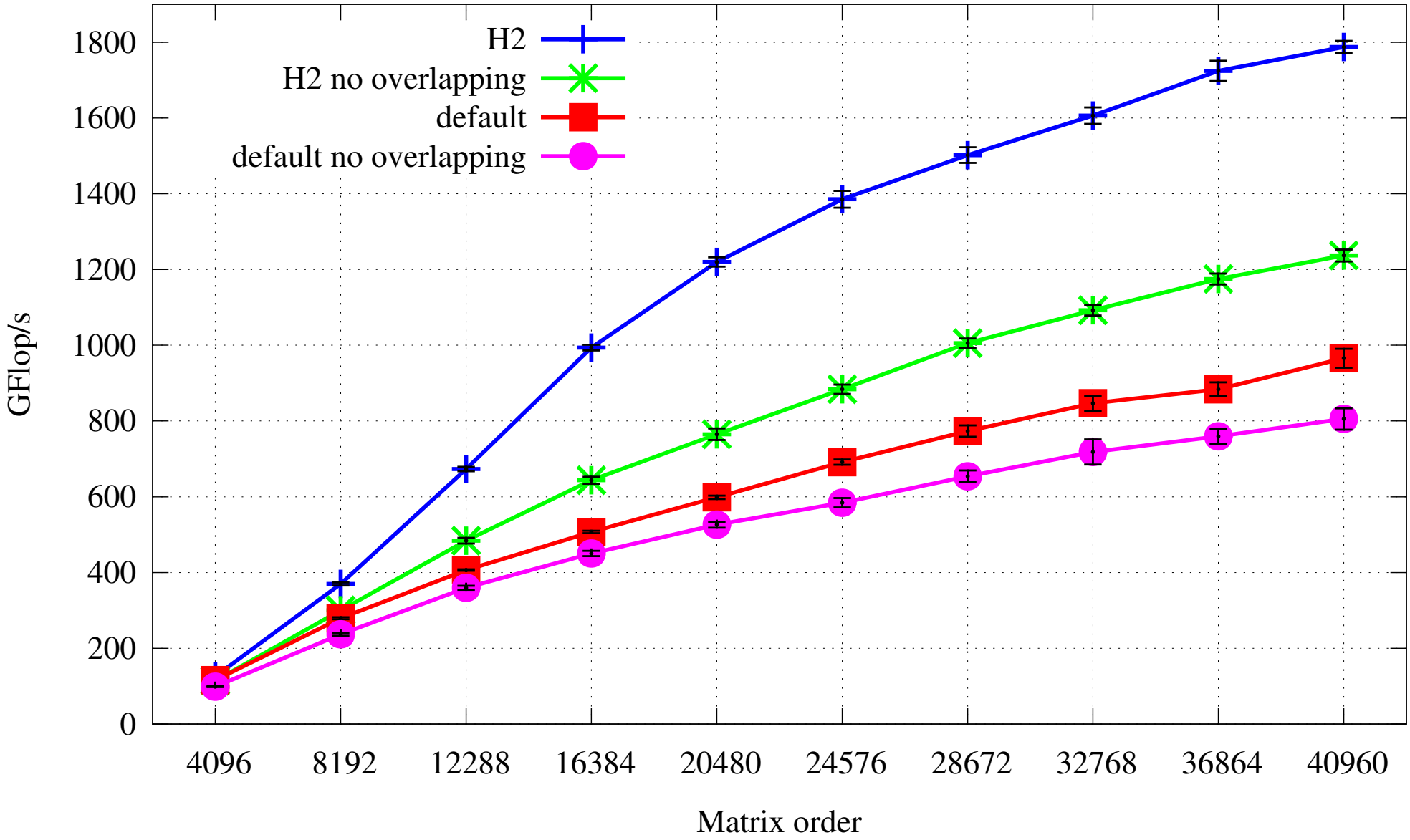
# Speedup Cholesky N=40960



# Speedup


4CPU+8GPU
6CPU+6GPU
8CPU+4GPU
10CPU+2GPU
11CPU+1GPU
Gflop/s
0
200
400
600
800
1000
1200
1400
1600
1800
4096
8192
12288
16384
20480
24576
28672
32768
36864
40960
Matrix order

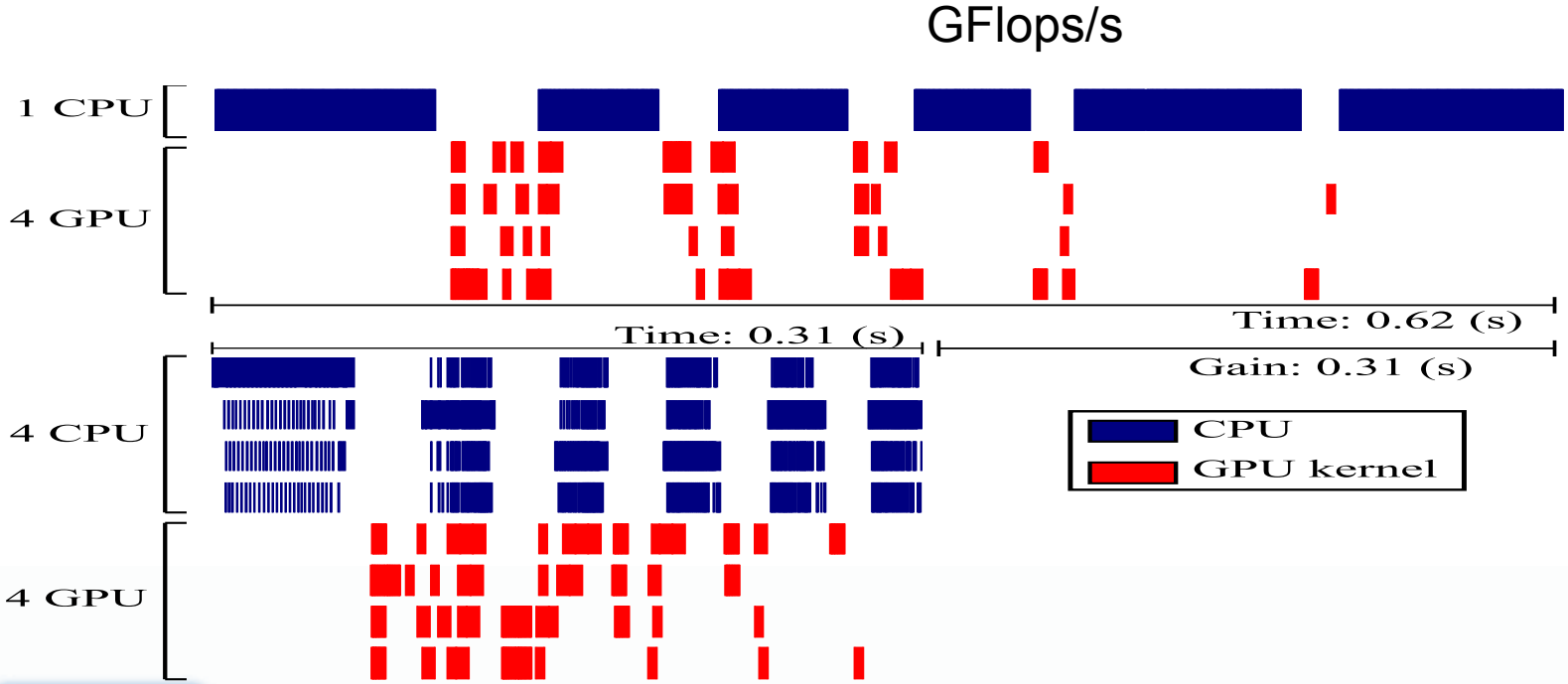
# Stratégies / Overlapping



# Réduction du chemin critique

- DPOTRF. Parallel panel factorization on x-CPUs, 4 GPUs, N=40960, BS=1024

| #CPU | Matrix dimension 4096 | 8192 | 16384 | 32768 | 40960 |
|---|---|---|---|---|---|
| 1 | 53.85 ±0.98 | 206.38 ±2.70 | 622.55 ±7.90 | 962.21 ±31.77 | 1052.58 ±20.53 |
| 4 | 115.16 ±1.02 | 391.05 ±2.64 | 755.91 ±6.89 | 1013.65 ±7.81 | 1022.45 ±37.55 |
| 8 | 138.34 ±1.06 | 439.70 ±3.38 | 782.21 ±10.51 | 999.46 ±6.90 | 1045.53 ±4.19 |



# Bilan

- **Exploitation des architectures hybrides de manière « oblivious »**
  - **un modèle de programmation par tâches avec dépendances**
    - contraintes, tâche récursive
  - **pas de modèle de performance de l'architecture**
  - **pas de modèle de performance de l'application**
  - **scabilité jusqu'à 8 GPUs !**

- **A la place**
  - **une gestion fine de l'overlapping**
  - **une gestion spécifique du cache logiciel**
  - **un vol de travail avec heuristique de gestion de l'affinité tâche/donnée/ressource**
    - similaire entre les versions NUMA et Multi-GPUs

- **Codage moins simple qu'en OpenMP**
  - **ADT INRIA K'STAR [MOAIS, RUNTIME] 2013-2015**
    - définition d'un compilateur basé sur CLANG OpenMP vers Kaapi et StarPU

# Conclusions

- **La programmation des architectures hybrides**
  - **un même environnement comme OpenMP n'est pas encore mature**
    - code avec les tâches et dépendances ? Oui => une version des PLASMA
    - code avec les targets ?
  - **un modèle à base de tâche est bien adapté pour la gestion multi-GPUs**
    - ordonnancement par vol de travail faisable, sans modèle de performance !
  - **restera complexe car actuellement plusieurs modèles de parallélisme à maitriser: CPU et GPU [Jonathan]**
    - sauf à ré-utiliser des bibliothèques… cublas etc
    - uniformisation des approches ? Modèle de tâche pour GPU [Europar 2012] ?

- **Les architectures modernes de HPC vont être hybrides pendant un temps**
  - **Non poursuite du KNL**
  - **Exploitation des plusieurs (>2) GPUs par nœuds ! 6 sur les OpenPower de Summit**

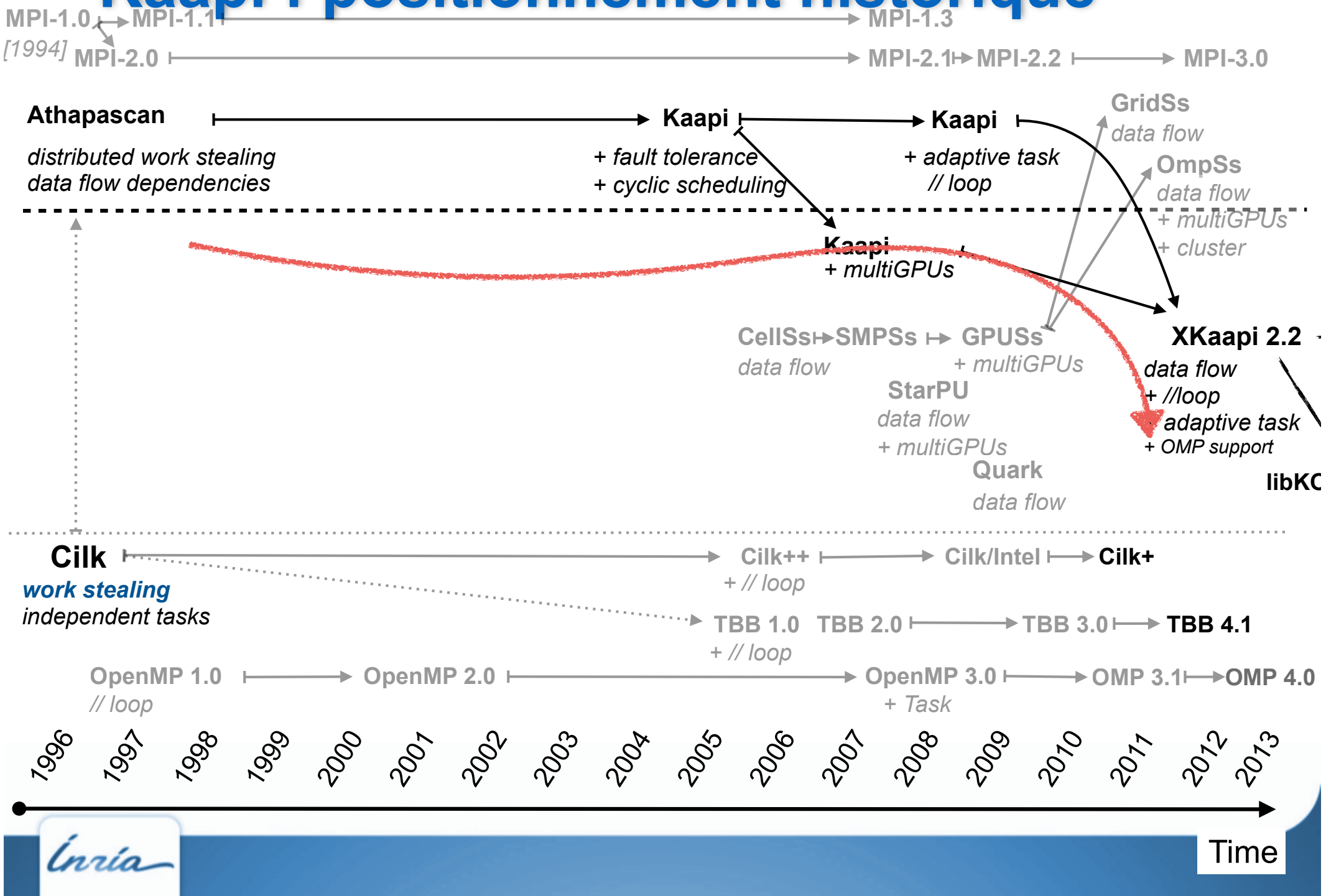
# Perspectives

- **Uniformisation de la programmation distribué, NUMA et multi-GPUs**
  - **1 même programme parallèle sur plusieurs architectures ?**
  - **tentatives Grenobloise :**
    - Cluster : Athapascan [PACT98],
    - Cluster + Grid : Kaapi [Pasco07]
    - GPU : [Europar2012]

- **Coût de maintenance des codes ?**
  - **par exemples approches à composant pour le HPC**
  - **importance à gérer les évolutions des codes et les variations d'architecture**

- **Analyse des performances automatiques ou semi-automatiques**

# Kaapi : positionnement historique



# Questions ?

# Bibliographie

## CPU and GPU


- João V. F. Lima, Thierry Gautier, Vincent Danjean, Bruno Raffin, Nicolas Maillard. Design and analysis of scheduling strategies for multi-CPU and multi-GPU architectures. Parallel Computing 44: 37-52 (2015)
- Raphaël Bleuse, Thierry Gautier, João V. F. Lima, Grégory Mounié, Denis Trystram. Scheduling Data Flow Program in XKaapi: A New Affinity Based Algorithm for Heterogeneous Architectures. Euro-Par 2014: 560-571
- Thierry Gautier, Joao Vicente Ferreira Lima, Nicolas Maillard, Bruno Raffin. XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures. In Proc. of the 27-th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Boston, USA, jun 2013.
- Thierry Gautier, Joao Vicente Ferreira Lima, Nicolas Maillard, Bruno Raffin. Locality-Aware Work Stealing on Multi-CPU and Multi-GPU Architectures. 6th Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG), Berlin, Allemagne, jan 2013.
- Julio Toss, Thierry Gautier. A New Programming Paradigm for GPGPU. EUROPAR 2012, Rhodes Island, Greece, aug 2012.
- J.V.F. Lima, Thierry Gautier, Nicolas Maillard, Vincent Danjean. Exploiting Concurrent GPU Operations for Efficient Work Stealing on Multi-GPUs. 24rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Columbia University, New York, USA, oct 2012.
- Everton Hermann, Bruno Raffin, François Faure, Thierry Gautier, Jérémie Allard. Multi-GPU and Multi-CPU Parallelization for Interactive Physics Simulations. EUROPAR 2010, Ischia Naples, Italy, aug 2010.


## Data flow programming language


- Thierry Gautier, Fabien Le Mentec, Vincent Faucher, Bruno Raffin. X-kaapi: A Multi Paradigm Runtime for Multicore Architectures. ICPP 2013: 728-735
- Marc Tchiboukdjian, Nicolas Gast, Denis Trystram: Decentralized List Scheduling CoRR abs/1107.3734: (2011). To appear.
- Thierry Gautier, Xavier Besseron, Laurent Pigeon. KAAPI: A Thread Scheduling Runtime System for Data Flow Computations on Cluster of Multi-Processors. Parallel Symbolic Computation'07 (PASCO'07), (15–23), London, Ontario, Canada, 2007.
- François Galilée, Jean-Louis Roch, Gerson Cavalheiro, Mathias Doreille. Athapascan-1: On-line Building Data Flow Graph in a Parallel Language. International Conference on Parallel Architectures and Compilation Techniques, PACT'98, :88–95, Paris, France, oct 1998.


## OpenMP & Kaapi


- Philippe Virouleau, François Broquedis, Thierry Gautier, Fabrice Rastello. Using Data Dependencies to Improve Task-Based Scheduling Strategies on NUMA Architectures. Euro-Par 2016: 531-544
- Philippe Virouleau, Adrien Roussel, François Broquedis, Thierry Gautier, Fabrice Rastello, Jean-Marc Gratien: Description, Implementation and Evaluation of an Affinity Clause for Task Directives. Proceedings of the 12th International Conference on OpenMP in a Heterogeneous World (IWOMP), Osaka, Japan, 2016
- Philippe Virouleau , Pierrick Brunet, Francois Broquedis, Nathalie Furmento, Samuel Thibault, Olivier Aumage, Thierry Gautier. Evaluation of the OpenMP Dependent Tasks with the KASTORS Benchmarks Suite. Proceedings of the 10th International Conference on OpenMP in a Heterogeneous World (IWOMP), Bahia, Brazil, sep 2014.
- Marie Durand, Francois Broquedis, Thierry Gautier, Bruno Raffin. An Efficient OpenMP Loop Scheduler for Irregular Applications on Large-Scale NUMA Machines. Proceedings of the 9th International Conference on OpenMP in a Heterogeneous World (IWOMP), 8122:141-155, Lecture Notes in Computer Science, Canberra, Australia, sep 2013.
- Francois Broquedis, Thierry Gautier, Vincent Danjean. libKOMP, an Efficient OpenMP Runtime System for Both Fork-Join and Data Flow Paradigms. IWOMP, :102-115, Rome, Italy, 2012.