Programmation à base de tâches : quels défis pour le passage à l'Exascale ?

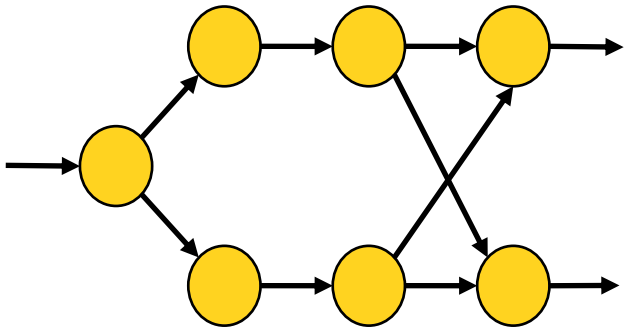
Raymond Namyst
Université de Bordeaux
Équipe Inria STORM

01

Les Mille et Une vertus de la programmation par tâches

La programmation par tâches

Découpage non-uniforme, potentiellement récursif, exécution asynchrone



Longtemps un objet d'étude pour les algorithmes d'ordonnancement …

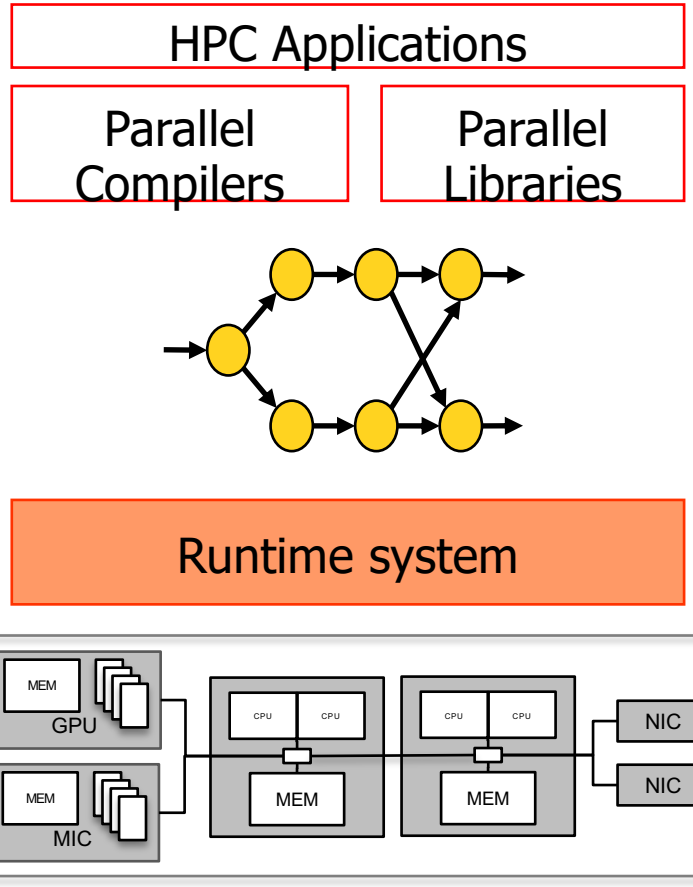
Une longue histoire

- Cilk
- Intel TBB
- Legion
- Open Community Runtime
- OMPSs
- OpenMP
- Parsec
- Quark
- StarPU
- X-Kaapi
- …

Mais une adoption récente

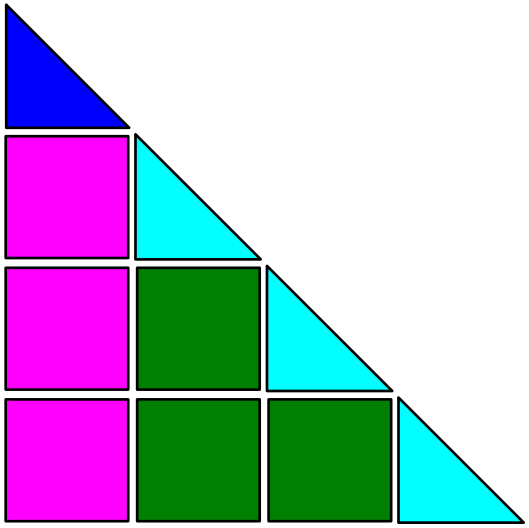
Modèle d'exécution

- Les tâches sont soumises à un support d'exécution
  - > Avec leurs dépendances
  - > « *Sequential Task Flow* » (STF)
- Le support d'exécution met en œuvre une stratégie d'ordonnancement dynamique pour répartir les tâches prêtes
  - > De multiples stratégies sont possibles
    - Vol de tâches (WS)
    - Date de fin au plus tôt (MCT)
    - Par affinité mémoire

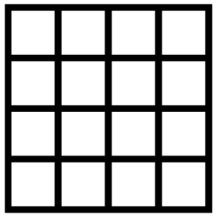


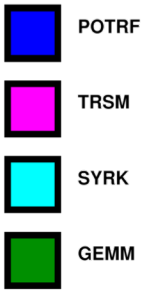
Exemple: PLASMA (2008, ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)



```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

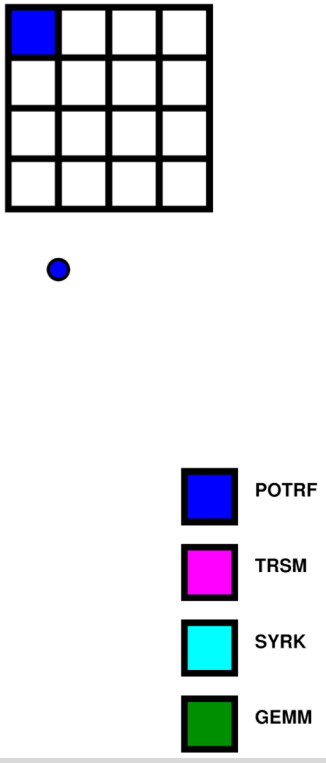




Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

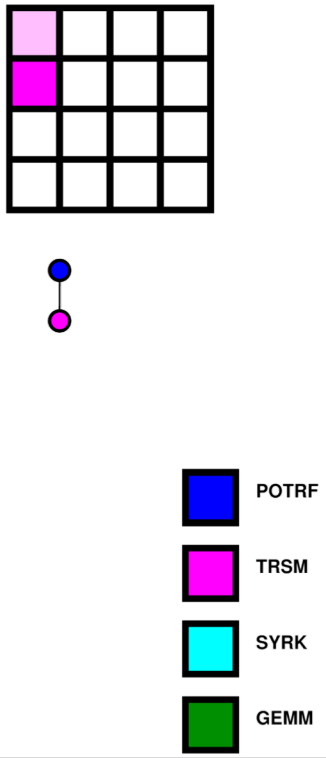
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

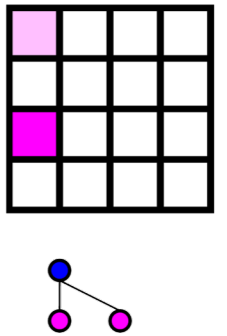
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

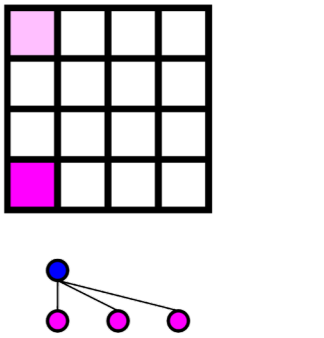
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

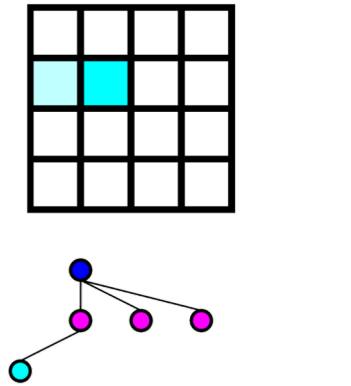
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

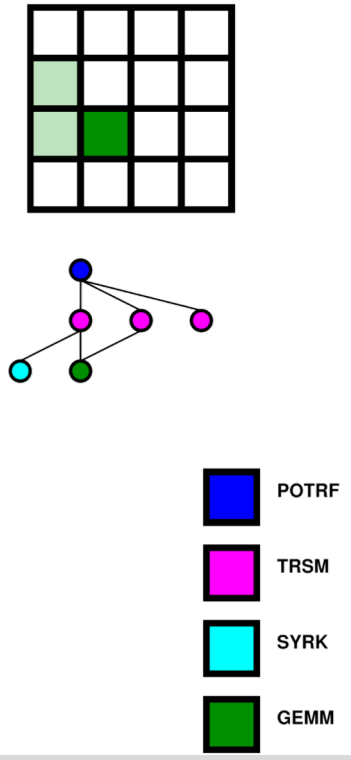
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

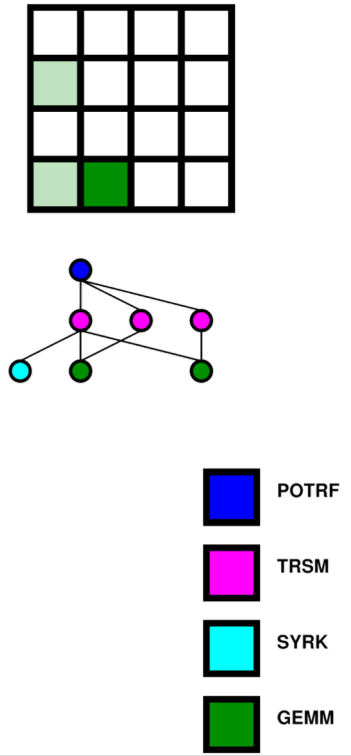
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

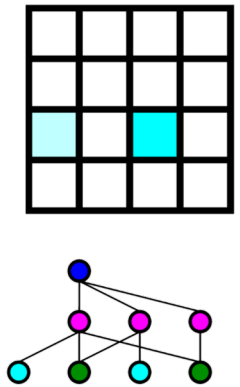
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

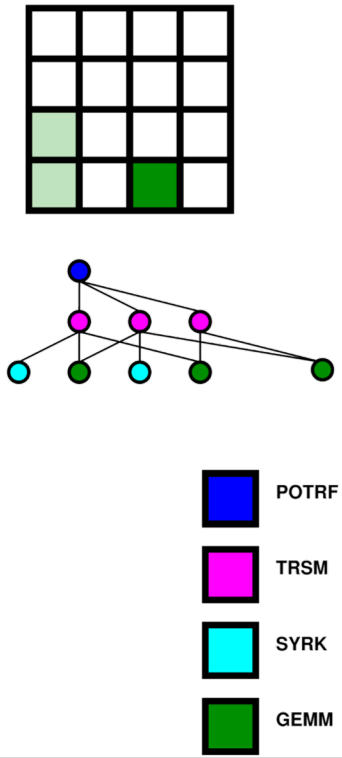
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

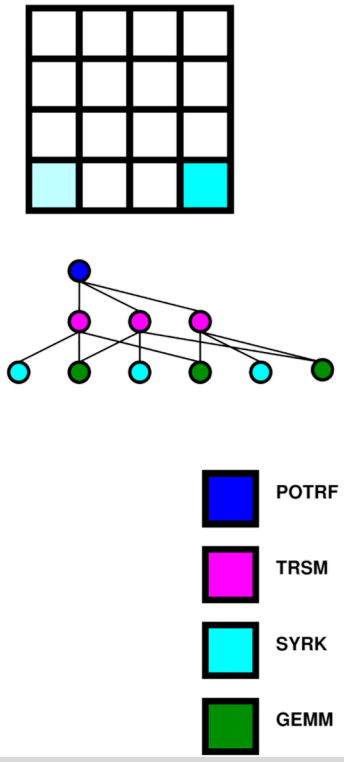
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

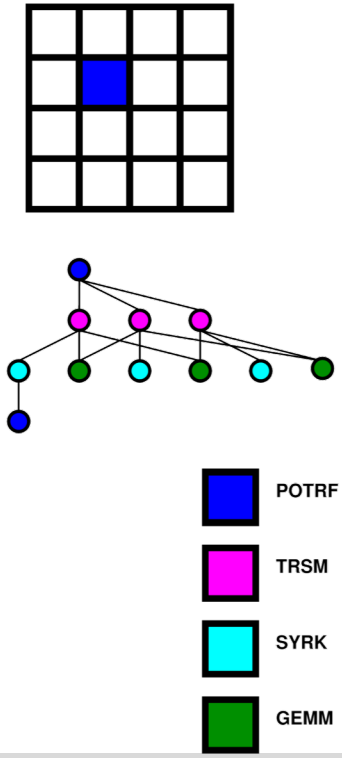
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

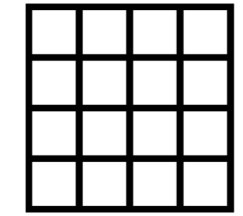
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

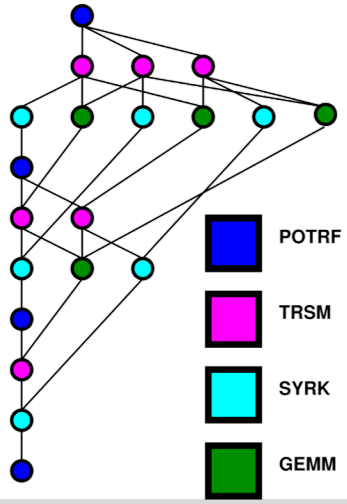


Exemple: PLASMA (ICL, Univ. Tennessee Knoxville)

Factorisation de Cholesky (A = L.LT)

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```



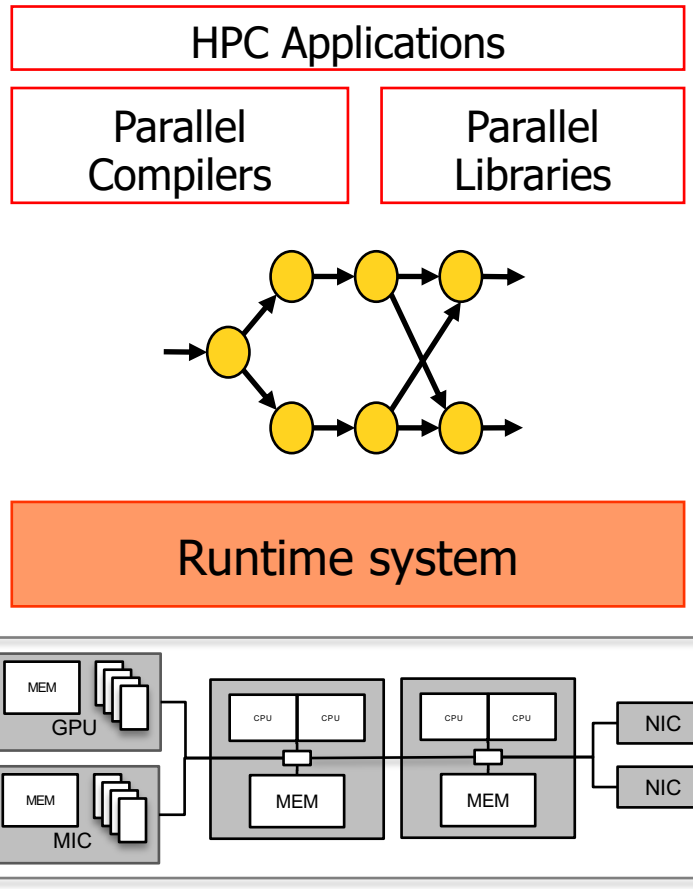


Modèle d'exécution

Pourquoi c'est mieux qu'avant ?

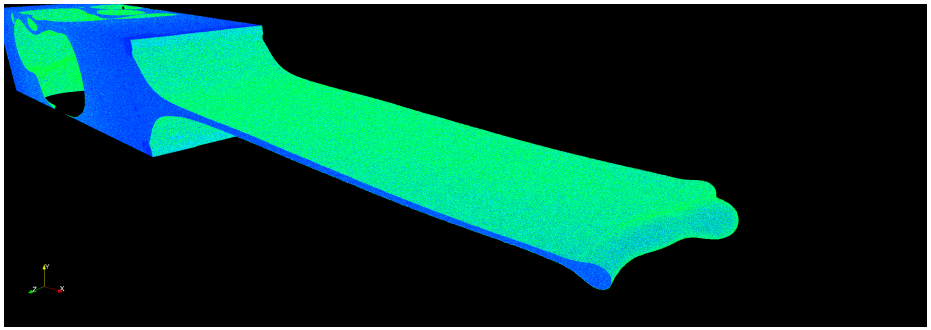
- Portabilité
  - > L'application exprime des dépendances/affinités
  - > L'ordonnanceur fait au mieux en fonction du matériel
- Efficacité
  - > Dépendances fines vs barrières de synchronisation
- Expressivité
  - > Récursivité

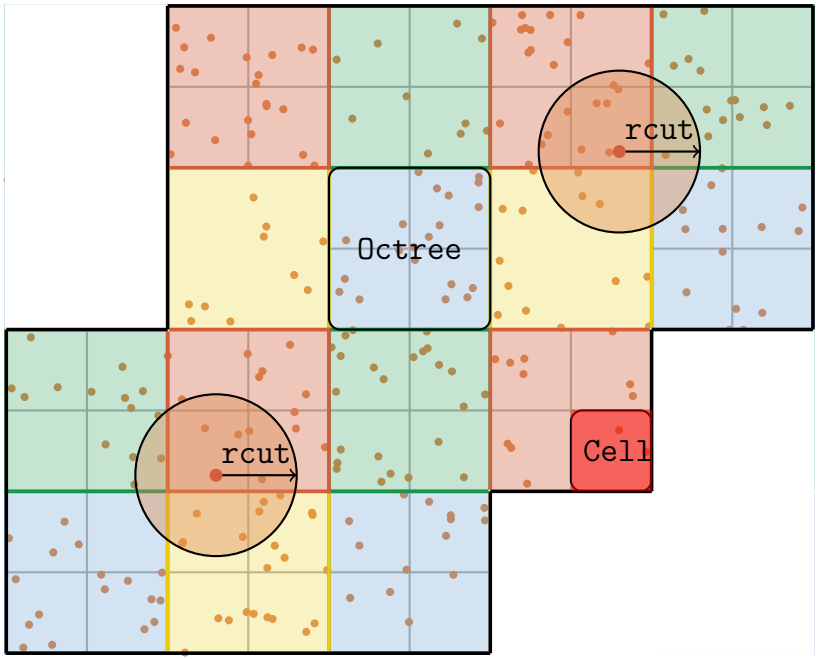
- Bémol : compréhension des performances ?



ExaStamp: Dynamique moléculaire (DPTA/CEA/DAM)

- Maillage adaptatif du domaine
- Calcul par « vagues » (couleurs)
  - > Suppression des synchronisations sur les particules
- Entrelacement des vagues grâce aux tâches OpenMP
  - > Gain ~10%



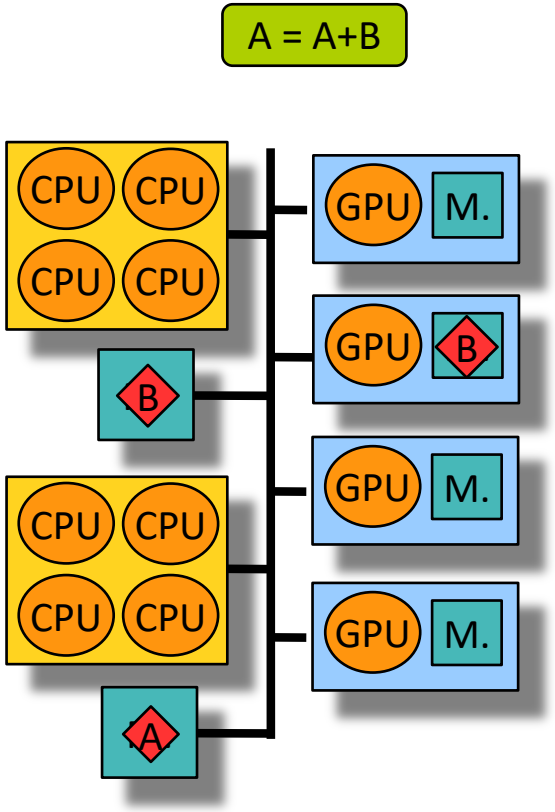


Architectures hétérogènes

Le modèle d'exécution par tâches se prête très bien aux accélérateurs

- Données en entrée et en sortie clairement identifiées
  - > Transferts CPU <-> GPU
- Possibilité de spécialiser les implémentations

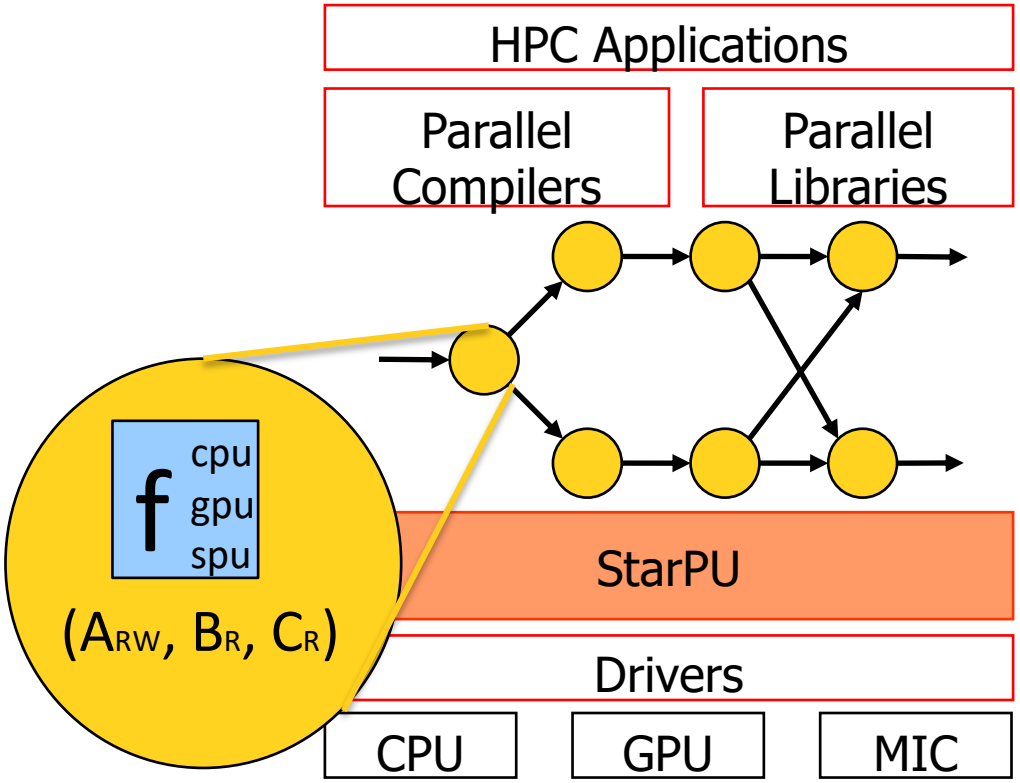
- Possibilité de laisser le support d'exécution choisir l'unité de calcul cible
  - > Machine = pool d'unités hétérogènes
  - > ≠ *offloading*



Architectures hétérogènes

Exemple de support d'exécution: StarPU (2008)

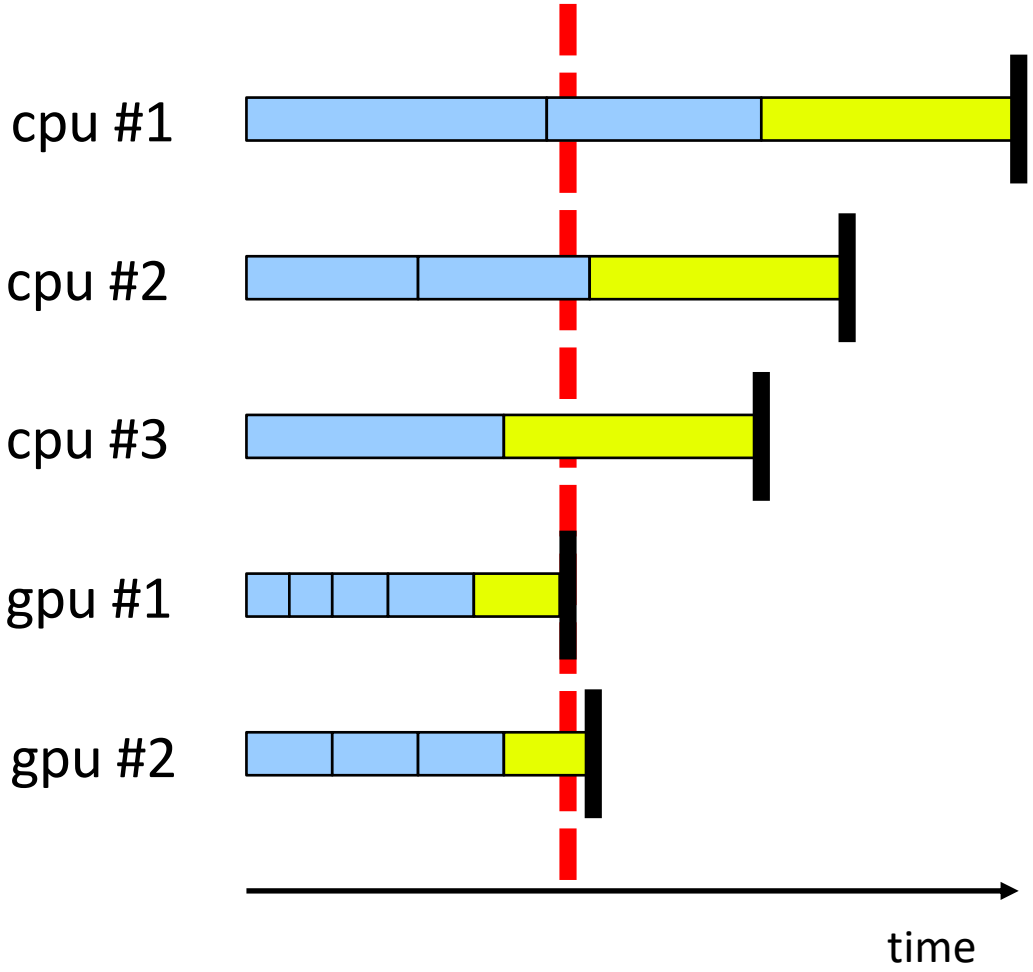
- Tâche =
  - > Références (R, W, RW) sur des données
  - > Une (ou plusieurs) implémentation(s)
- Ordonnanceur programmable
  - > Stratégies = plugins
- Caches logiciels pour minimiser les transferts



StarPU

Exemple de stratégie : MCT (minimum completion time)

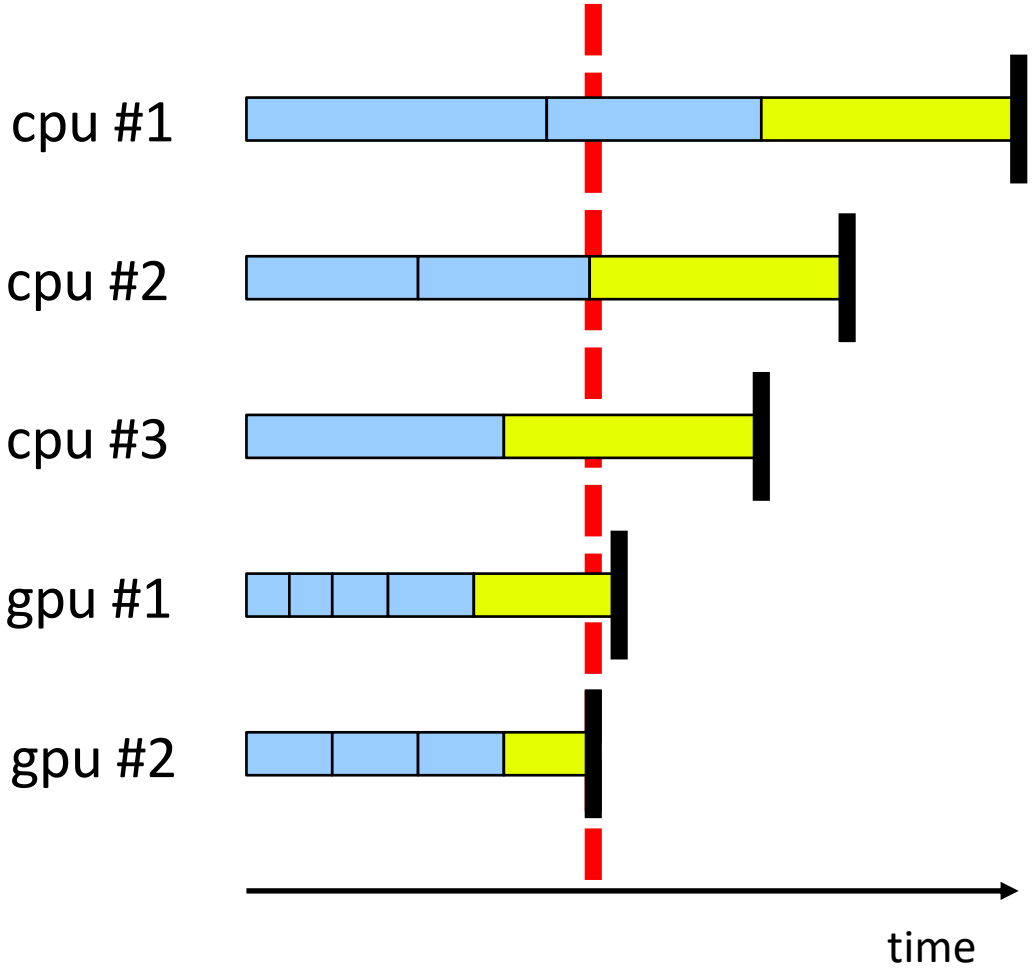
- Prédiction de la durée des tâches par historique
  - > Très fiable avec des noyaux BLAS
- On affecte les tâches dès qu'elles sont prêtes
  - > Préchargement des données



StarPU

Exemple de stratégie : MCT (minimum completion time)

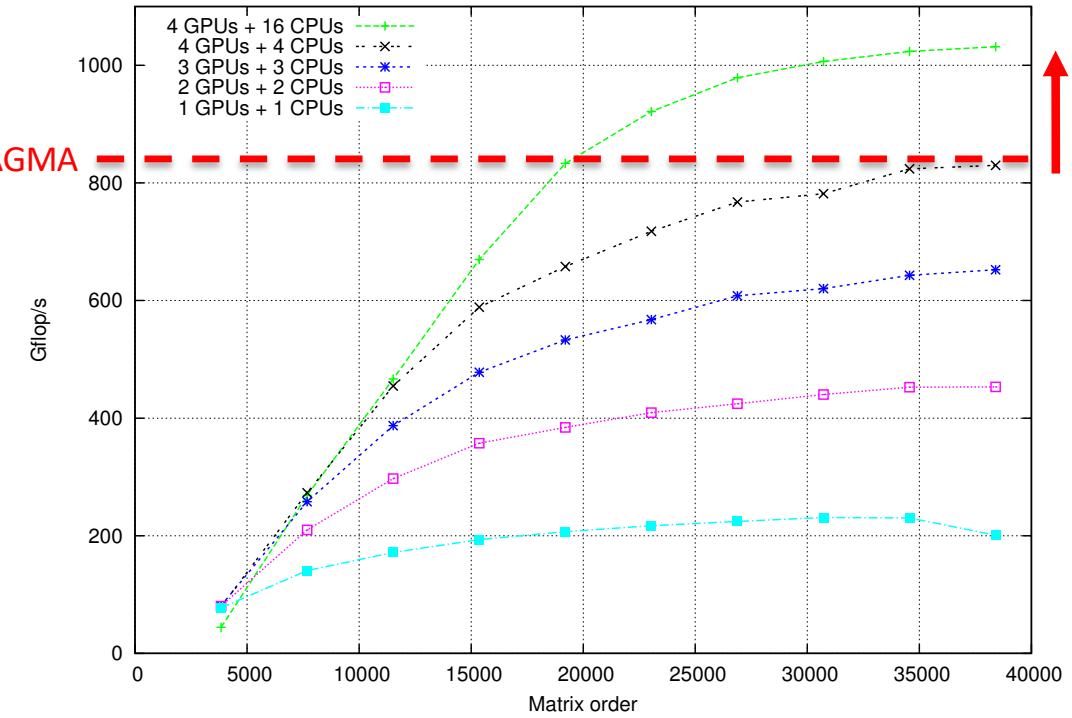
- On intègre le temps de transfert non-recouvert au temps de calcul
  - > Plus difficile à prédire précisément



StarPU

PLASMA/StarPU

- Décomposition QR
  - > 16 CPU AMD + 4 GPU C1060
- Le nombre de flops obtenu est supérieur à la somme des flops sur chaque type d'unité !



+12 CPUs
~200 GFlops

(although
12 CPUs alone
~150 Gflops)

StarPU

Efficacité comparée des noyaux de calcul

- sgeqrt
  - > CPU: 9 Gflops GPU: 30 Gflops Ratio: x3
- stsqrt
  - > CPU: 12 Gflops GPU: 37 Gflops Ratio: x3
- somqr
  - > CPU: 8.5 Gflops GPU: 227 Gflops Ratio: x27
- Sssmqr
  - > CPU: 10 Gflops GPU: 285 Gflops Ratio: x28

Distribution des tâches

- sgeqrt: 20% of tasks on GPUs
- Sssmqr: 92.5% of tasks on GPUs

Inria

Les tâches gagnent du terrain

Dans de nombreux domaines

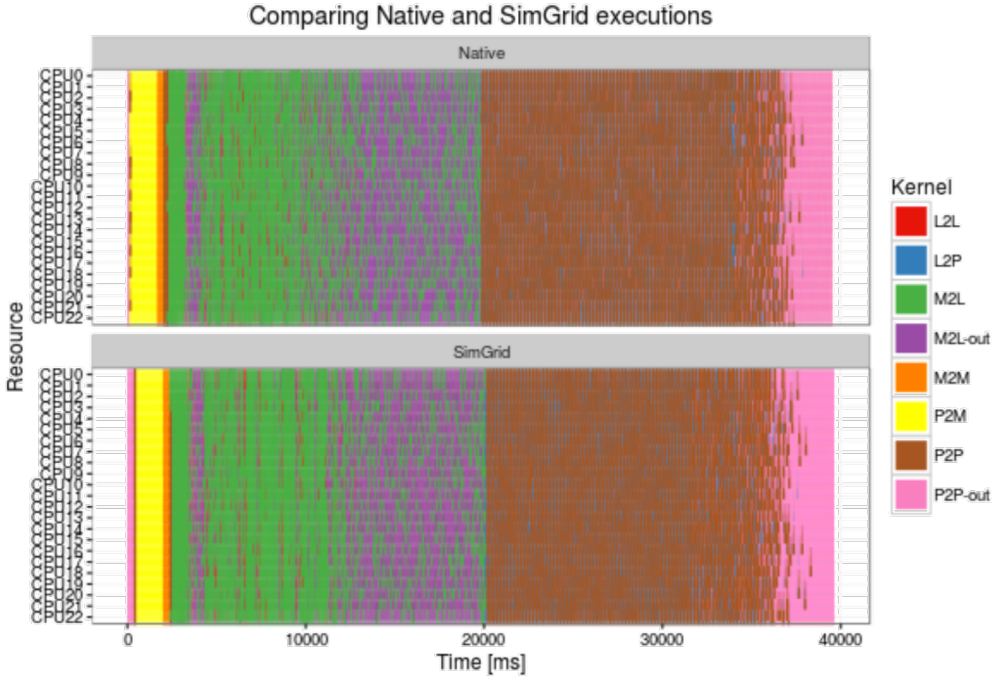
- Algèbre linéaire creuse (e.g. H-matrices)
- CFD, FMM, CG, MD
- Calcul In Situ
- Etc.

Performances proches de l'optimal

- D'après l'étude des bornes théoriques et les outils de simulation

Outils de simulation performants

- StarPU/SimGrid
  - > Permet de simuler le comportement sur des machines artificielles



02

Sur la route de l'Exascale

Quels défis pour les supports d'exécution ?

La tendance architecturale des nœuds de calcul se confirme

- Nombre important de cœurs
- Unités de calcul vectorielles larges
- Processeurs hétérogènes

Parallélisme extrême au niveau des supercalculateurs

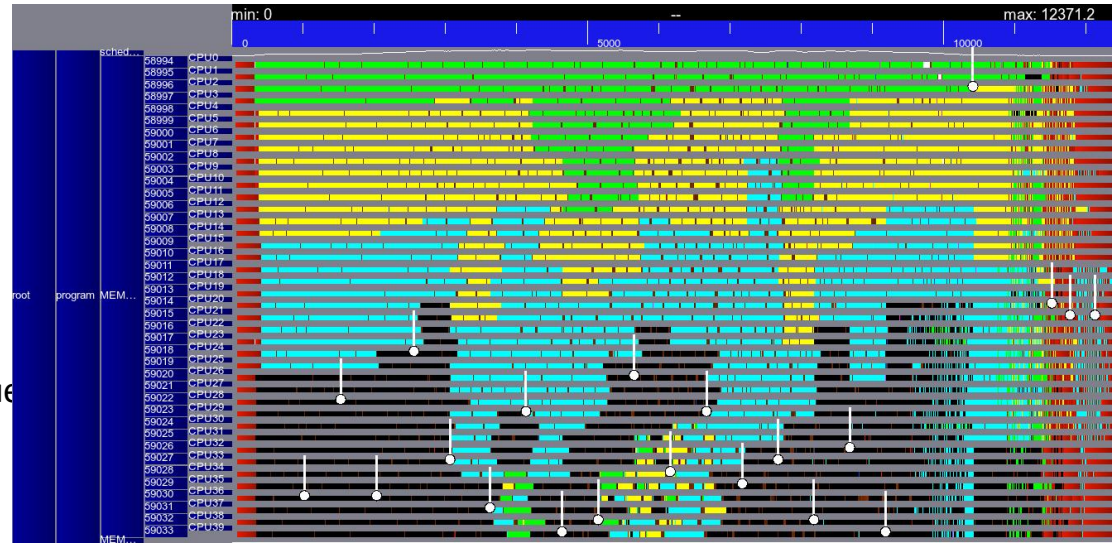
- Combien de tâches pour exploiter efficacement tous les cœurs ?
  - > Amortir les temps de transfert, le préchargement des caches, etc.
- Est-il raisonnable de vouloir ordonnancer finement toutes ces tâches ?
- Est-ce que les stratégies d'ordonnancement usuelles passent à cette échelle ?

Quels défis pour les supports d'exécution ?

Passage à l'échelle des stratégies d'ordonnancement

- Accepter de ne plus avoir une vue globale de l'activité des cœurs
  - > Ordonnanceurs hiérarchiques
  - > Algorithmes locaux

- Partitionner la machine dynamiquement
  - > Co-ordonnancement de plusieurs bibliothèque parallèles
  - > Eviter les interférences
  - > Utiliser des supports d'exécution différents
    - Réutiliser du code !



Quels défis pour les supports d'exécution ?

Déséquilibre des puissances de calcul

- La granularité des tâches est difficile à calibrer
  - > Tâches trop volumineuses
    - OK pour GPU, mais trop long sur CPU
  - > Tâches trop petites

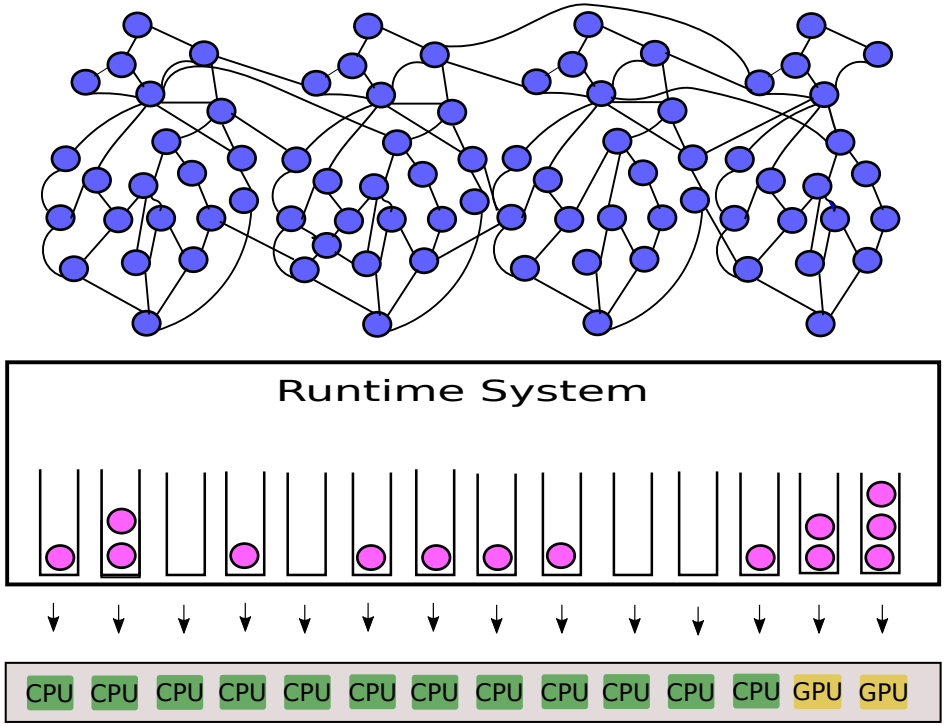
- Vers une granularité variable des tâches
  - > Tâches divisibles
  - > Tâches hiérarchiques
    - Récursivement divisibles
  - > Tâches parallèles

Modèle à base de tâches classiques

Les noyaux s'exécutent de manière parallèle sur les accélérateurs

Mais en séquentiel sur les CPU !

- Les mauvais choix d'ordonnancement peuvent coûter cher…



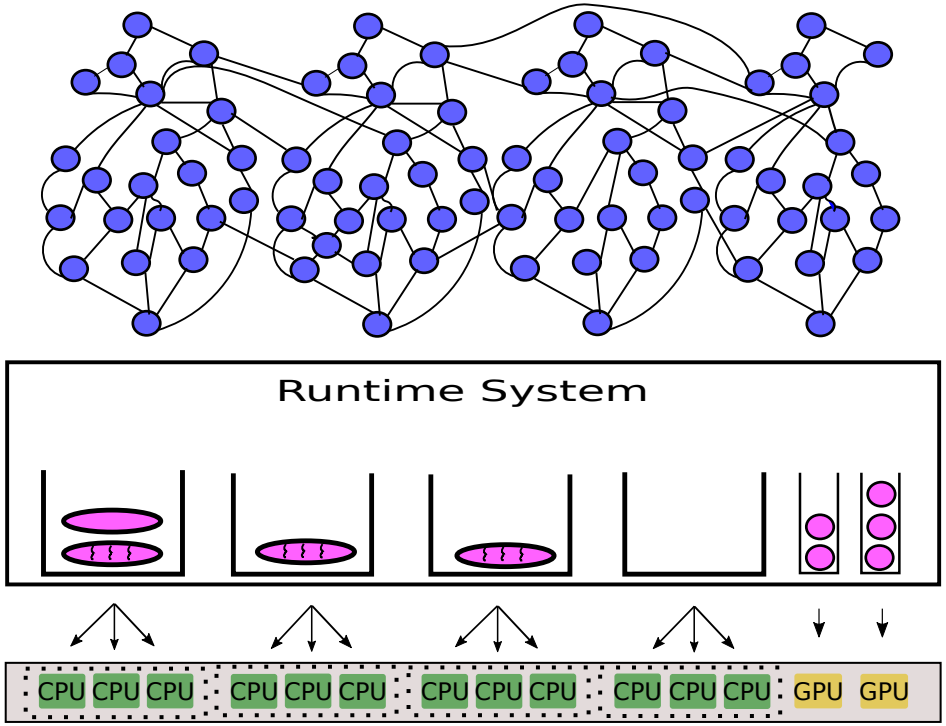
Modèle à base de tâches parallèles

On forme des « clusters » de CPU

- Exécution parallèle (e.g. OpenMP) à l'intérieur
- Les tâches deviennent maléables
- Moins d'écart avec les accélérateurs
- Moins de pression sur le cache

Comment déterminer la taille et le nombre des clusters ?

- Problème difficile
- Modifier dynamiquement les clusters ?
  - > Encore plus difficile ☹



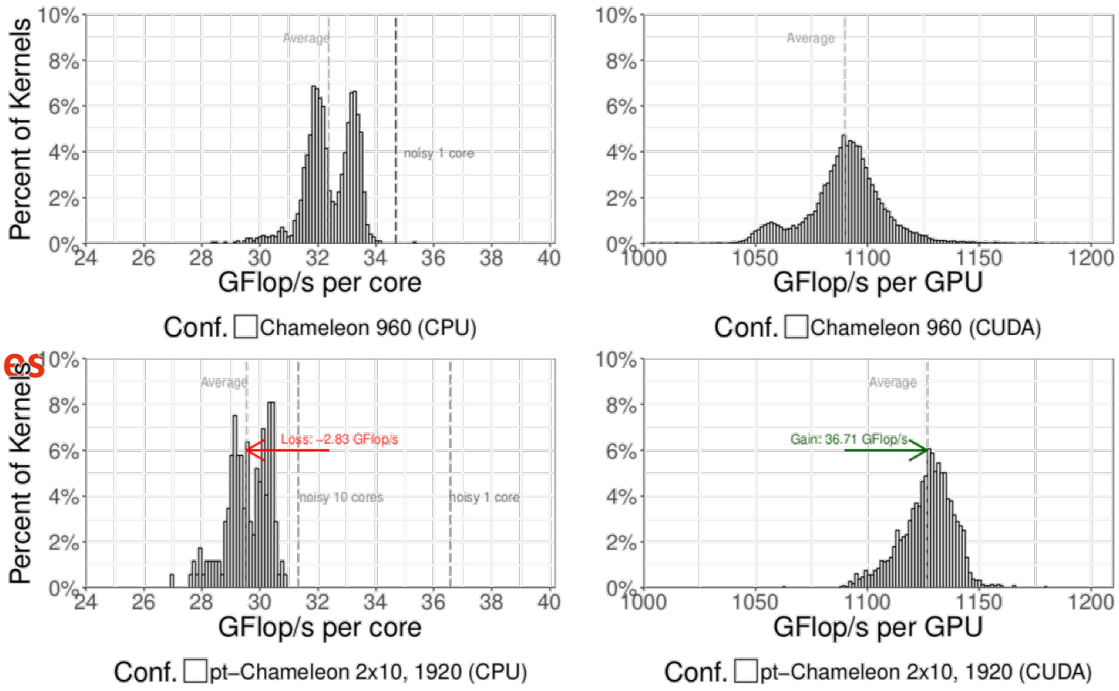
Modèle à base de tâches parallèles

On forme des « clusters » de CPU

- Exécution parallèle (e.g. OpenMP) à l'intérieur
- Les tâches deviennent maléables
- Moins d'écart avec les accélérateurs
- Moins de pression sur le cache

Technique validée dans plusieurs codes

- Bibliothèque Chameleon (HIEPACS)
- Code FLUPESA (Airbus D&S)



Tâches hiérarchiques

Décomposition des tâches déclenchée par l'ordonnanceur

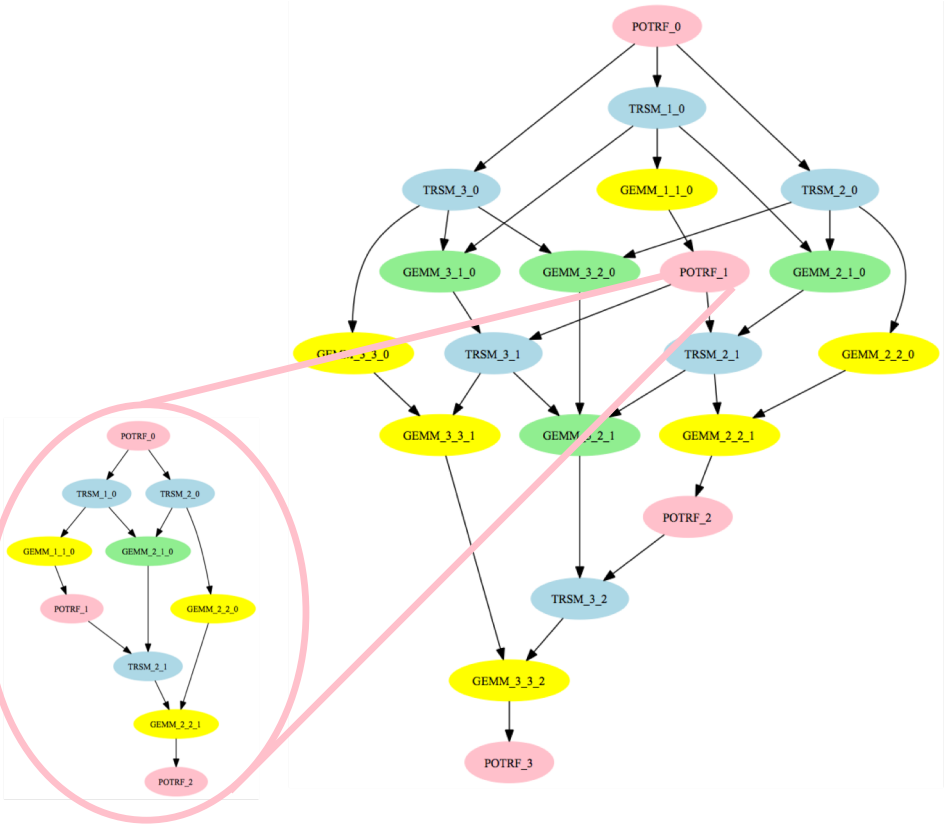
- en un sous-graphe…
- ou en une tâche parallèle (e.g. OpenMP)…
- ou en une implémentation séquentielle

Adaptation dynamique

- Parallélisme à la demande
  - > En fonction de l'occupation de la machine

Défis

- Éviter les sur-synchronisations
- Décomposition/recomposition des données



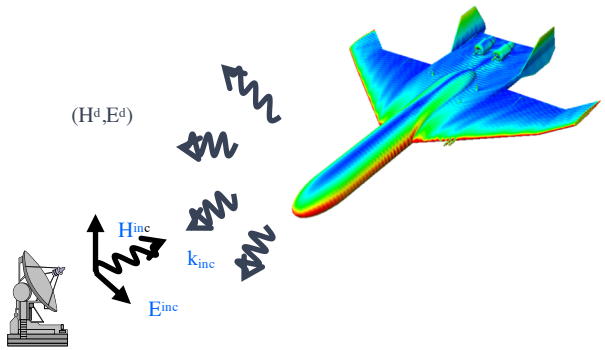
Code Arlène 3D, Cesta

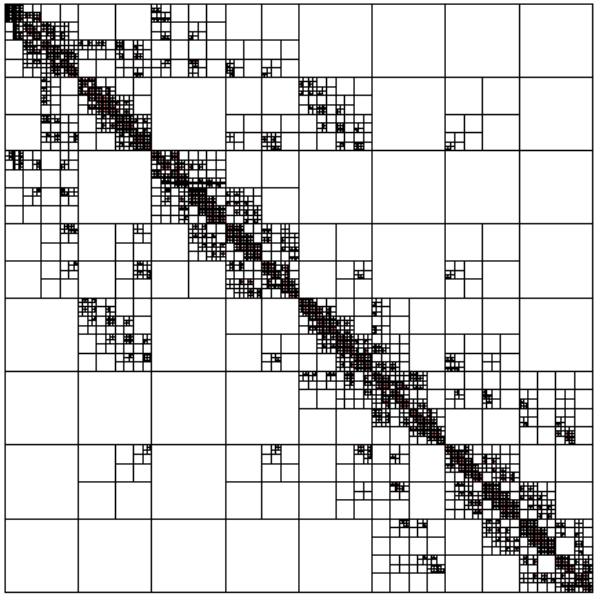
Equations de Maxwell

Solveur direct LU ou LL^T

Compression hiérarchique

- H-Matrices

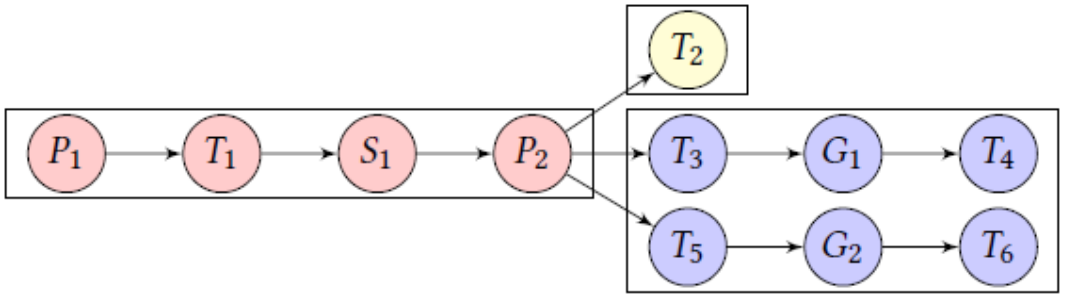




Code Arlène 3D, Cesta

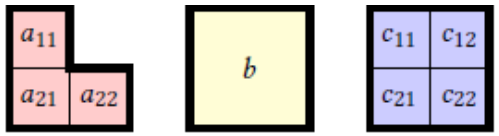
Modèle « Sequential Task Flow »

- Tout est tâche, même les étapes des communications MPI

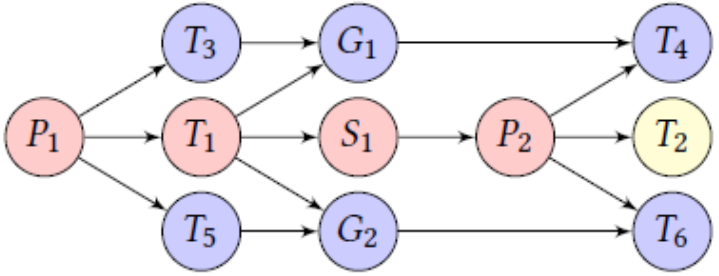


Fork-Join

Les tâches sont récursives



H-POTRF(A^W)
H-TRSM(A^R,B^W)
H-TRSM(A^R,C^W)



Dépendances fines

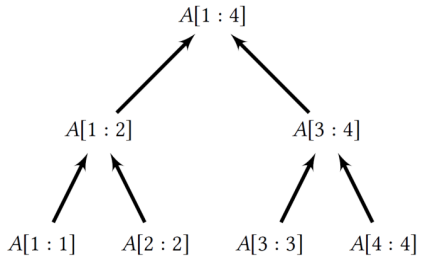
Code Arlène 3D, Cesta

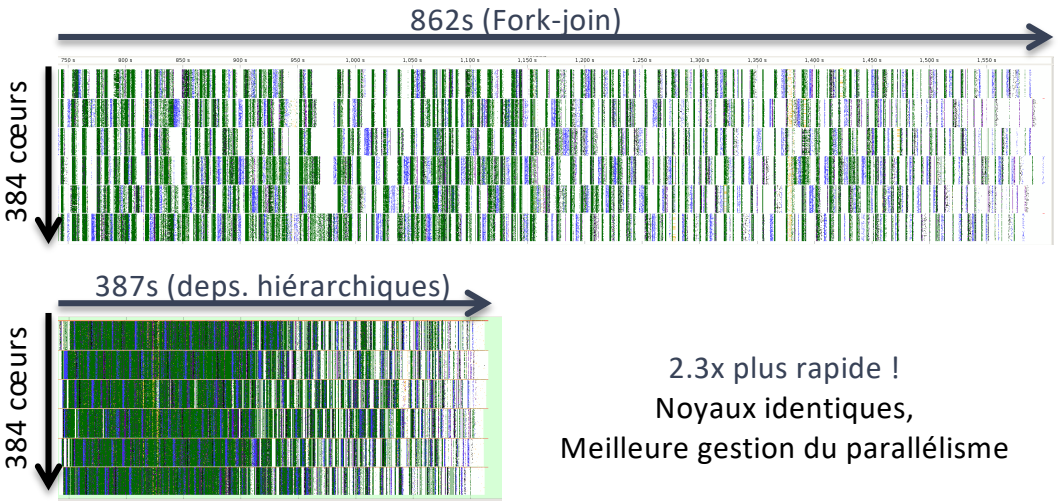
Les données sont hiérarchiques

- Les tâches peuvent manipuler des données à différents niveaux de découpage

- Système de dépendances capable d'inférer les dépendances automatiquement à partir des accès aux données / sous-données
  > Nécessite des mécanismes de description hiérarchique des données
    - Runtime du Cesta, StarPU

- 70% d'efficacité sur un cluster de 380 KNL (24K cœurs)





2.3x plus rapide !
Noyaux identiques,
Meilleure gestion du parallélisme

Conclusion

La programmation à base de tâches est bien adaptée

- Aux architectures hétérogènes
- Aux applications complexes (non régulières)
- Au couplage simulation + analyse/visualisation

Il reste de nombreux défis posé par l'évolution des architectures

- Granularité du parallélisme
- Ordonnancement hiérarchique

…mais aussi par l'évolution des applications

- Composition, couplage, visualisation in situ
- Nombreux travaux sur la convergence/interopérabilité entre supports d'exécution

Inria

Merci !

https://team.inria.fr/storm/