Programmation orientée tâches

GLCS

Introduction à la
programmation orientée tâches

Remerciements :
Olivier Aumage
Jack Dongarra

Le matériel intra-nœud

- Dans un nœud aujourdhui
  - CPUs classiques, ~16-32 threads
  - GPUs, ~10k threads
- Hier
  - MIC (Intel Xeon-Phi), ~248 threads
- Et demain ?
- Choix débit vs. latence
- Accélérateurs (GPU et certains MICs)
  - Machine hétérogène + transferts mémoire

Parallélisme intra nœud

- Plusieurs modèles, p.ex.
  - MPI (partage mémoire, CPU+MIC)
  - Pthreads (bas niveau, CPU+MIC)
  - Cuda/OpenCL (bas niveau, GPU)
  - OpenMP // for (CPU+MIC)
  - OpenACC/OpenMP target + // for (GPU)
  - OpenMP tâches (CPU+MIC+GPU)

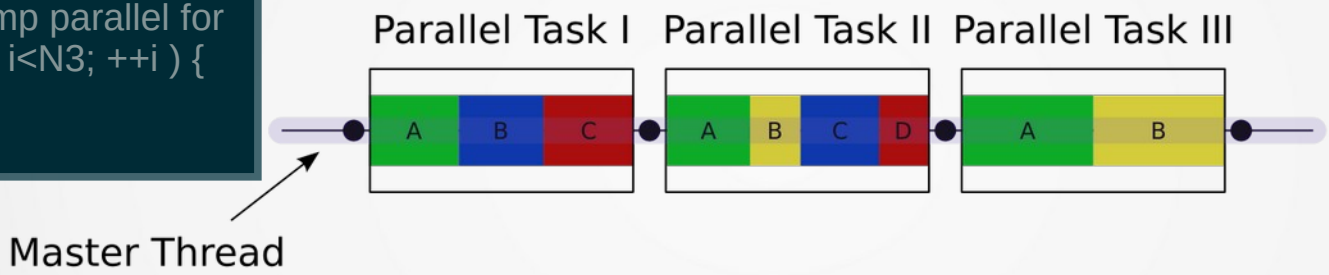
- Objectif
  - Portabilité + **portabilité de performances !**

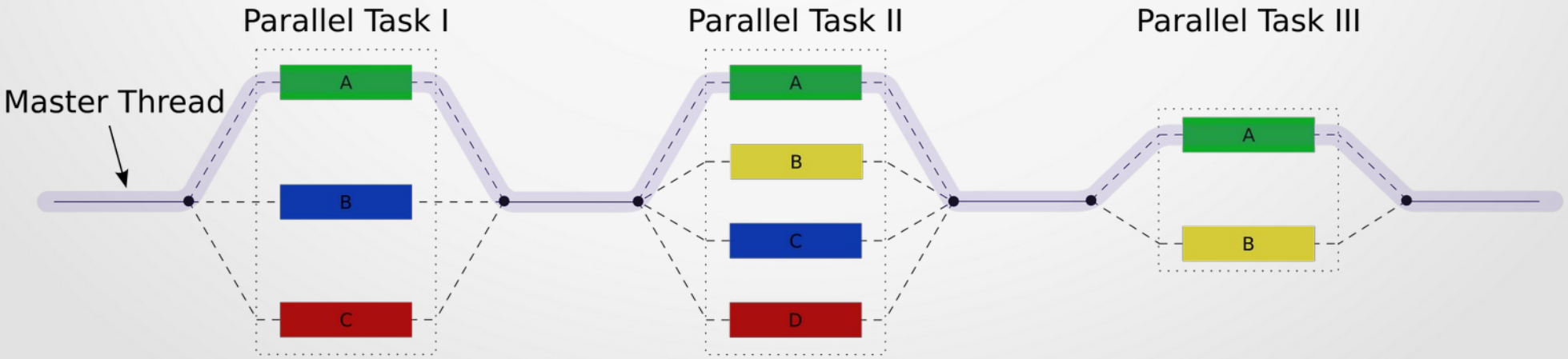
Le modèle parallel for

- Modèle classique dans OpenMP
- Code séquentiel + boucles parallèles
- Aussi connu comme fork-join
  - Fork avant boucle
  - Join en fin de boucle

Fork-join

```
#pragma omp parallel for
for ( int i=0; i<N1; ++i ) {
  //…
}
#pragma omp parallel for
for ( int i=0; i<N2; ++i ) {
  //…
}
#pragma omp parallel for
for ( int i=0; i<N3; ++i ) {
  //…
}
```





Fork-join : discussion

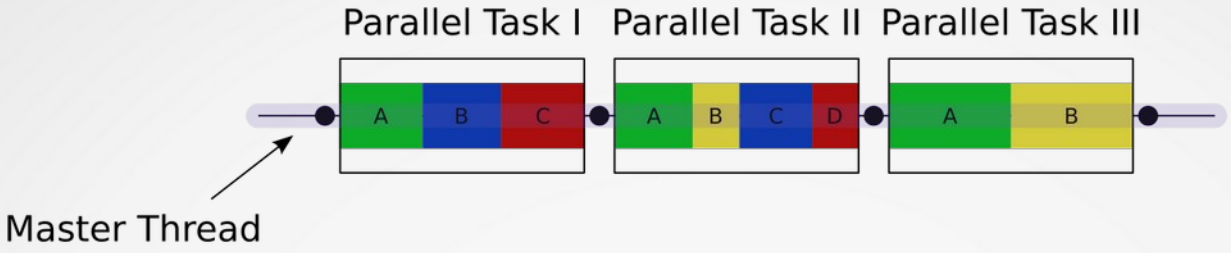
+ Relativement simple
+ Adapté au parallélisme de données

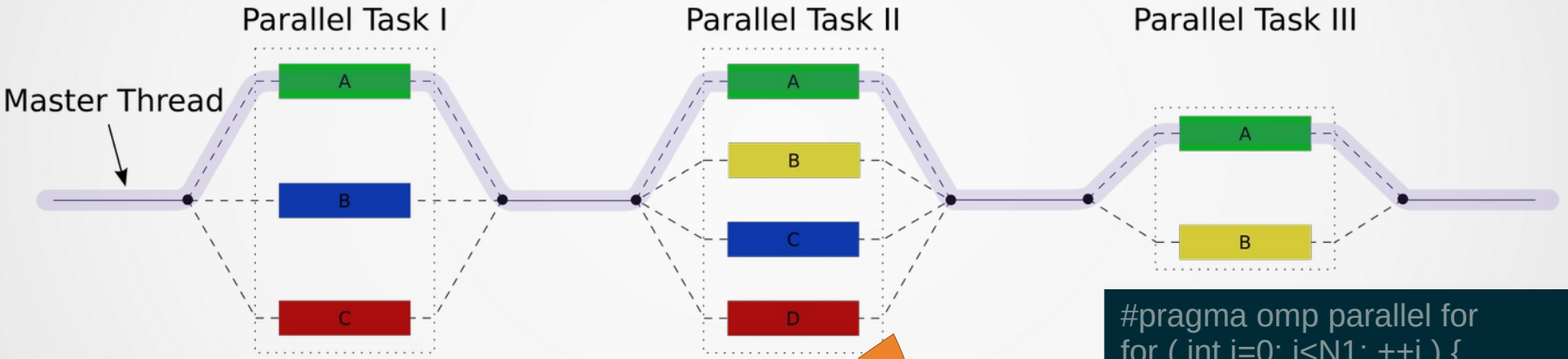
- Limité au parallélisme de données
- Loi d'Amdahl
  - Zones séquentielles dominent à large échelle

Le modèle parallel for

- Modèle classique dans OpenMP
- Code séquentiel + boucles parallèles
- Aussi connu comme fork-join
  - Fork avant boucle
  - Join en fin de boucle
- Extension target
  - Choix du processeur pour chaque boucle (CPU/GPU)
  - Transfert des données nécessaire

Fork-join + target






```
#pragma omp parallel for
for ( int i=0; i<N1; ++i ) {
  //…
}
#pragma omp target device(GPU)
#pragma omp parallel for
for ( int i=0; i<N2; ++i ) {
  //…
}
#pragma omp parallel for
for ( int i=0; i<N3; ++i ) {
  //…
}
```

Fork-join avec target

+ Choix du meilleur processeur possible
+ Peu de modifications / au fork-join classique

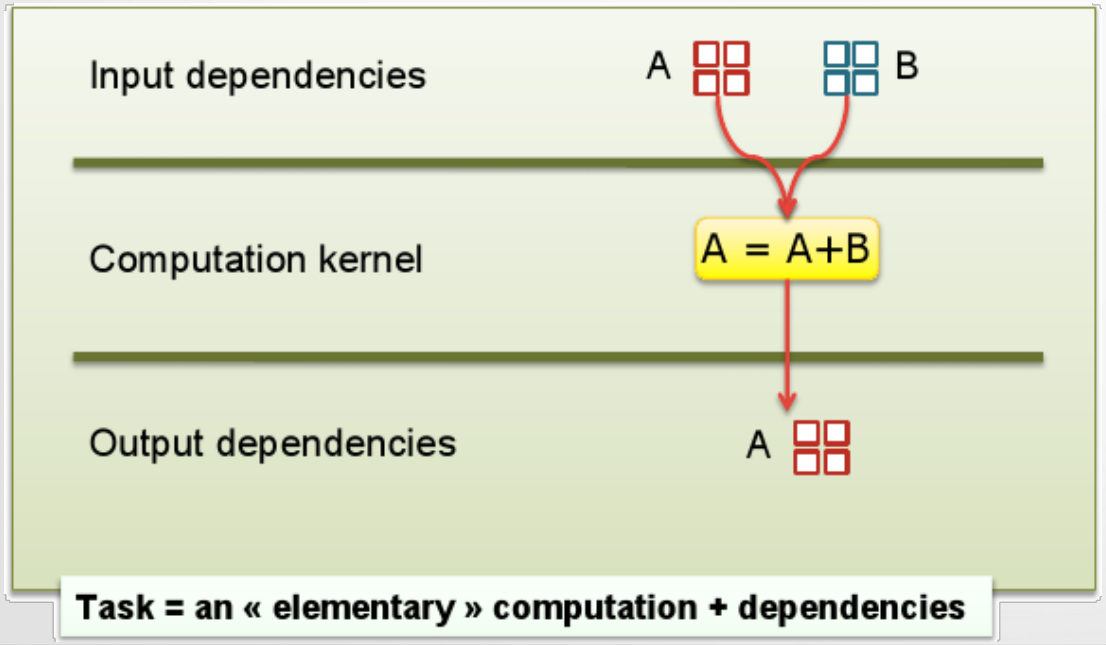
- Coût de transfert de données
- 1 seul processeur utilisé par boucle

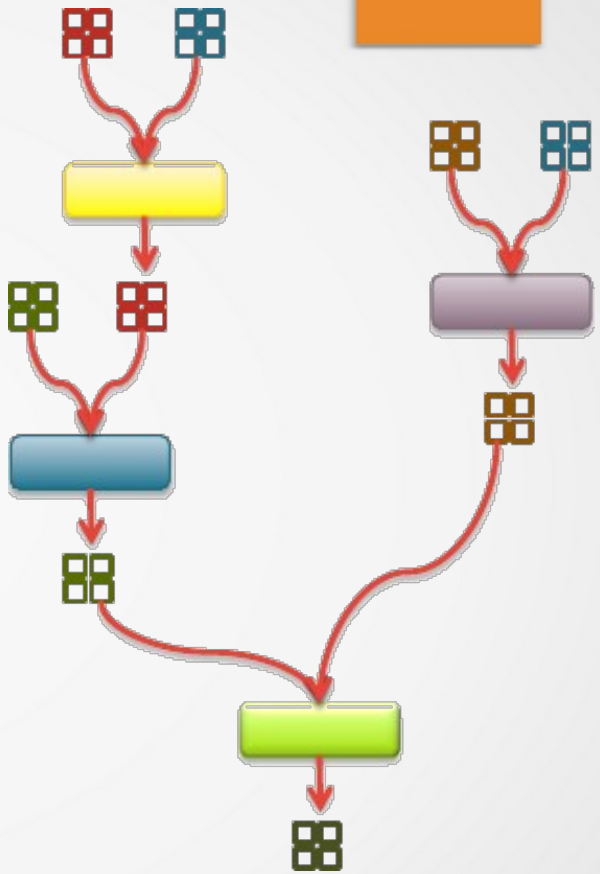
Modèle à taches

- Objectif
  - Portabilité de performances
  - Support multiples *PU (CPU/GPU/…) hétérogènes
- 1 Langage
  - Cilk, OpenMP4+, OmpSs, …
- 1 Exécutif (Runtime)
  - Parsec, StarPU, Xkaapi, …

Tâches : les concepts

- Une tâche
  - Données d'entrée
  - Noyau de calcul
  - Données de sortie

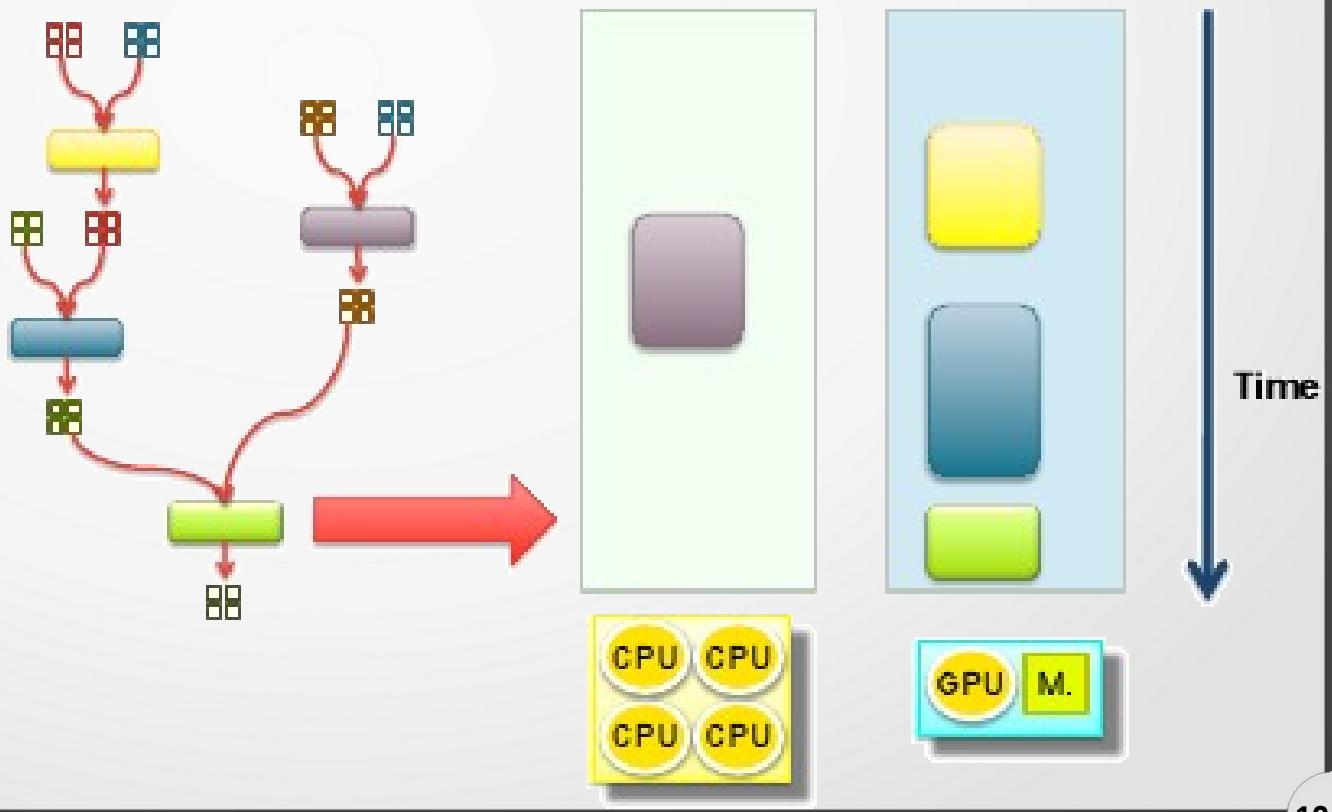




- Application
  - Graphe acyclique dirigé (DAG)

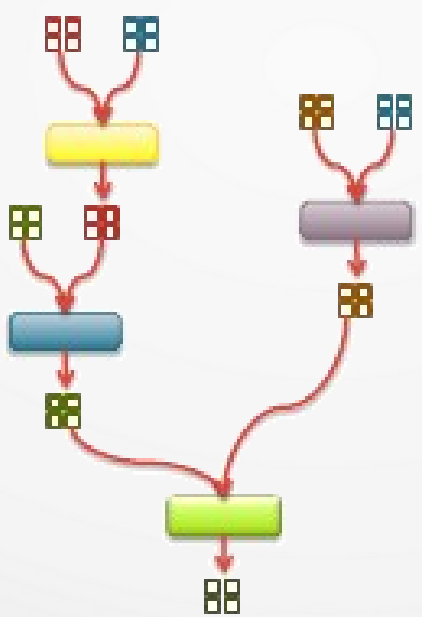
Tâches : exécution

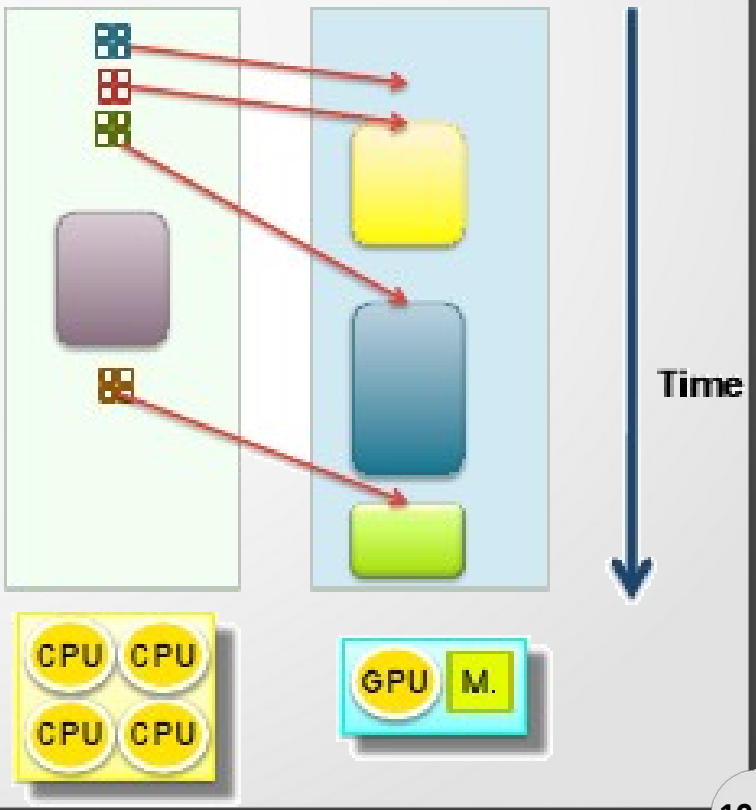
- Exécution du graphe
  - Mapping tâches sur CPUs, GPUs, …
  - Quand toutes dépendances prêtes
  - Transfert données si nécessaire



Tâches : exécution

- Exécution du graphe
  - Mapping tâches sur CPUs, GPUs, …
  - Quand toutes dépendances prêtes
  - Transfert données si nécessaire



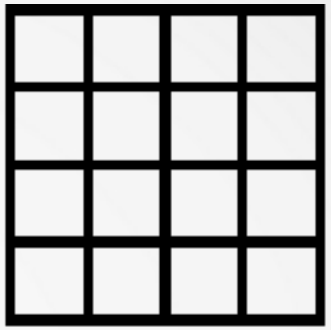


Tâches, ex. Cholesky

```
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]);
  }
}
__wait__();
```

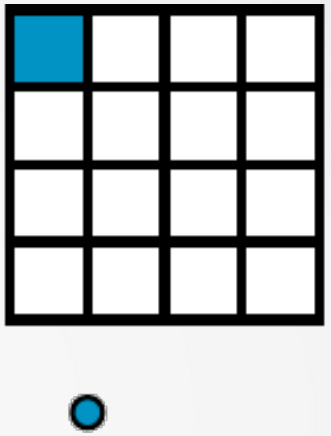
Tâches, ex. Cholesky

```
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]);
  }
}
__wait__();
```



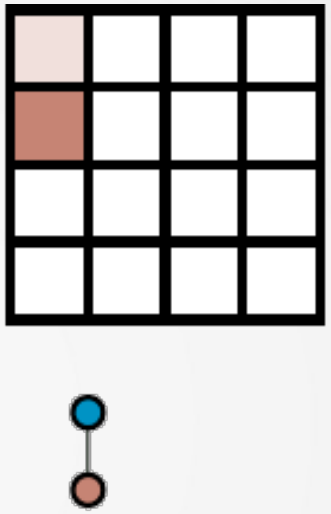
Tâches, ex. Cholesky

```
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]);
  }
}
__wait__();
```



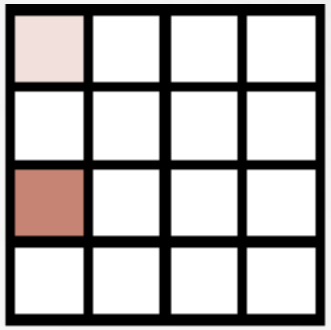
Tâches, ex. Cholesky

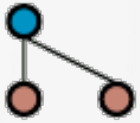
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```



Tâches, ex. Cholesky

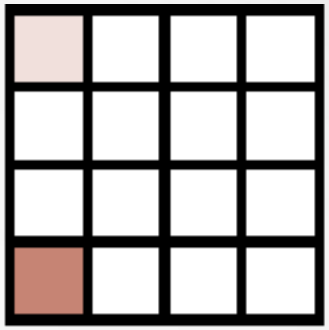
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

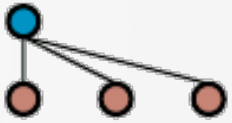




Tâches, ex. Cholesky

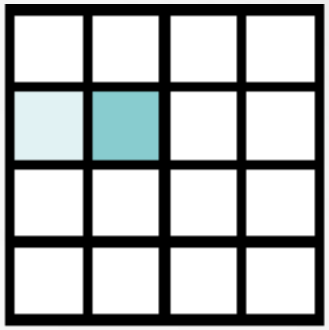
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

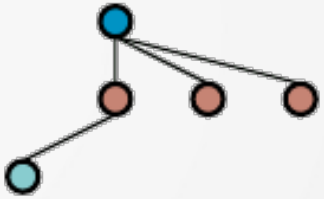




Tâches, ex. Cholesky

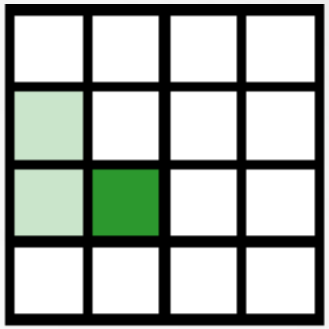
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

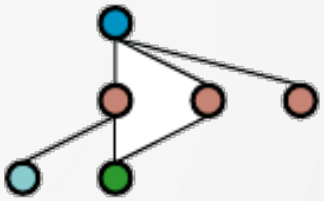




Tâches, ex. Cholesky

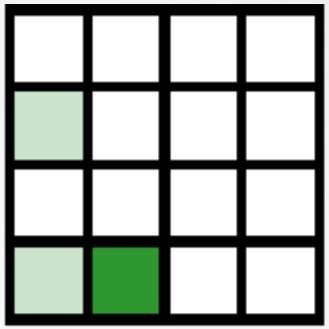
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

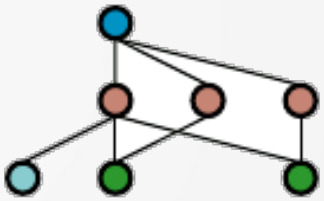




Tâches, ex. Cholesky

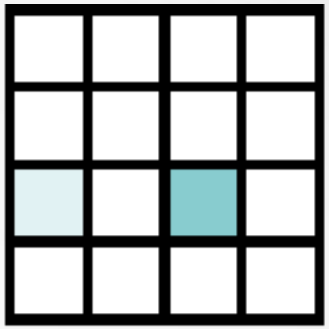
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

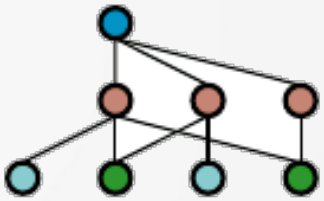




Tâches, ex. Cholesky

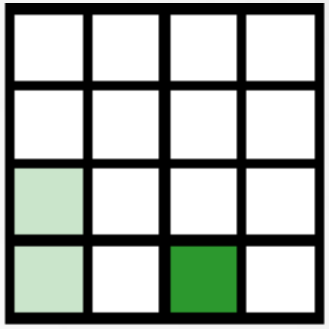
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

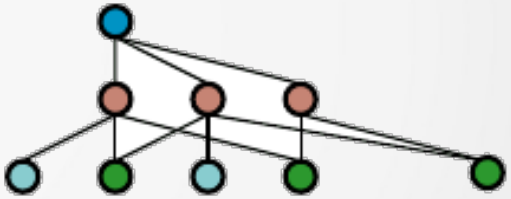




Tâches, ex. Cholesky

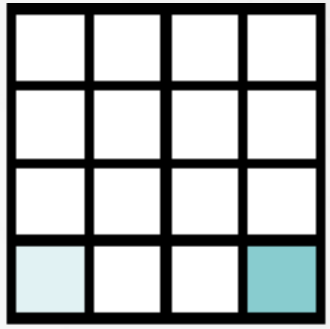
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

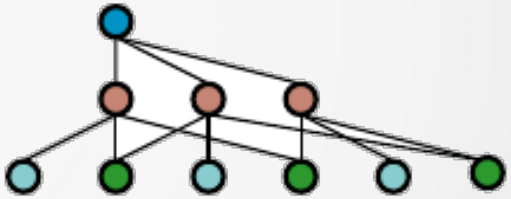




Tâches, ex. Cholesky

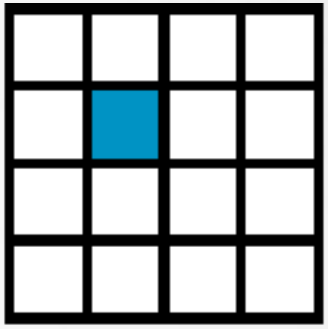
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

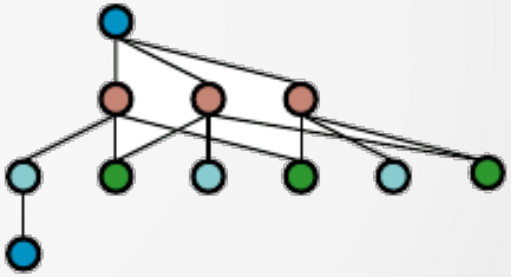




Tâches, ex. Cholesky

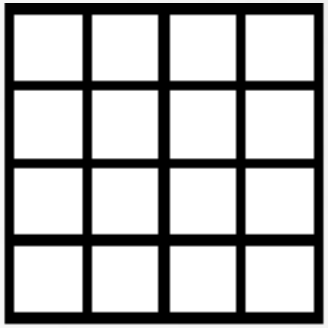
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
__wait__();
```

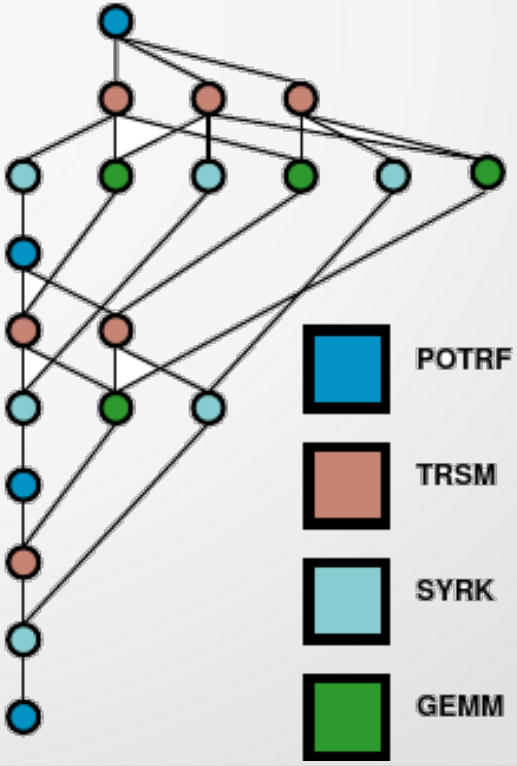




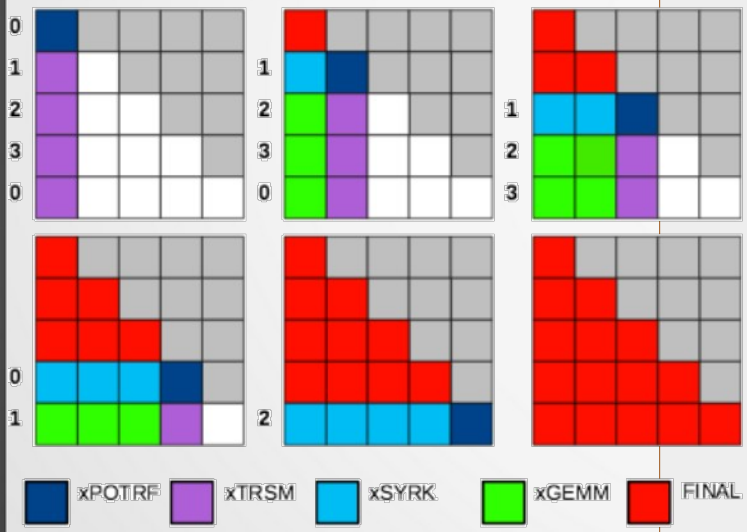
Tâches, ex. Cholesky

```
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]);
  }
}
__wait__();
```



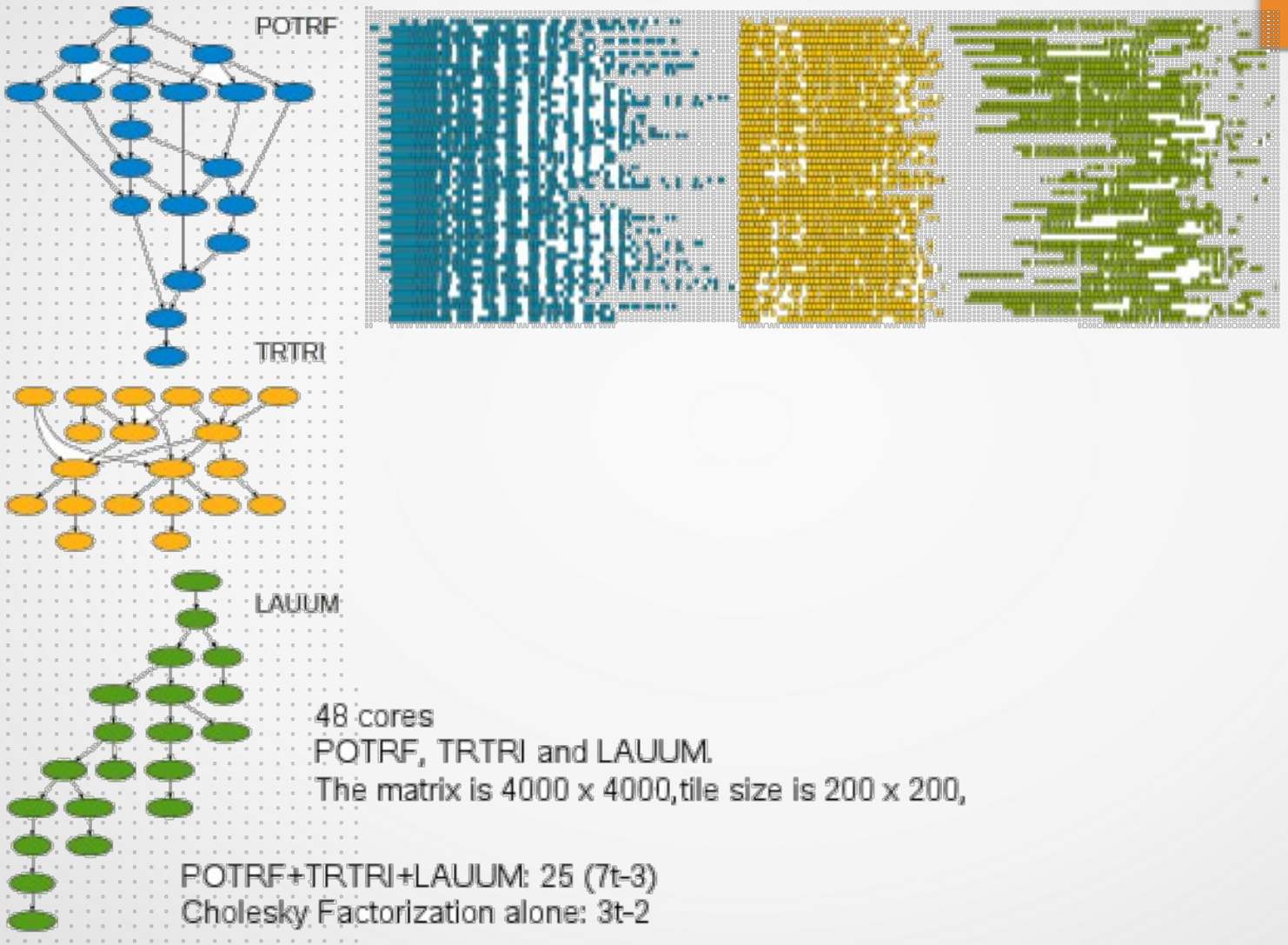


Cholesky : OpenMP




```
#pragma omp parallel
#pragma omp master
{  CHOLESKY( A );  }
CHOLESKY( A ) {
    for (k = 0; k < M; k++) {

        {  POTRF( A(k,k) );  }
        for (m = k+1; m < M; m++) {


            { TRSM( A(k,k), A(m,k) ); }
        }
        for (m = k+1; m < M; m++) {


            { SYRK( A(m,k), A(m,m) ); }
            for (n = k+1; n < m; n++) {


                {  GEMM( A(m,k), A(n,k), A(m,n) ); }
            }
        }
    }
}
```

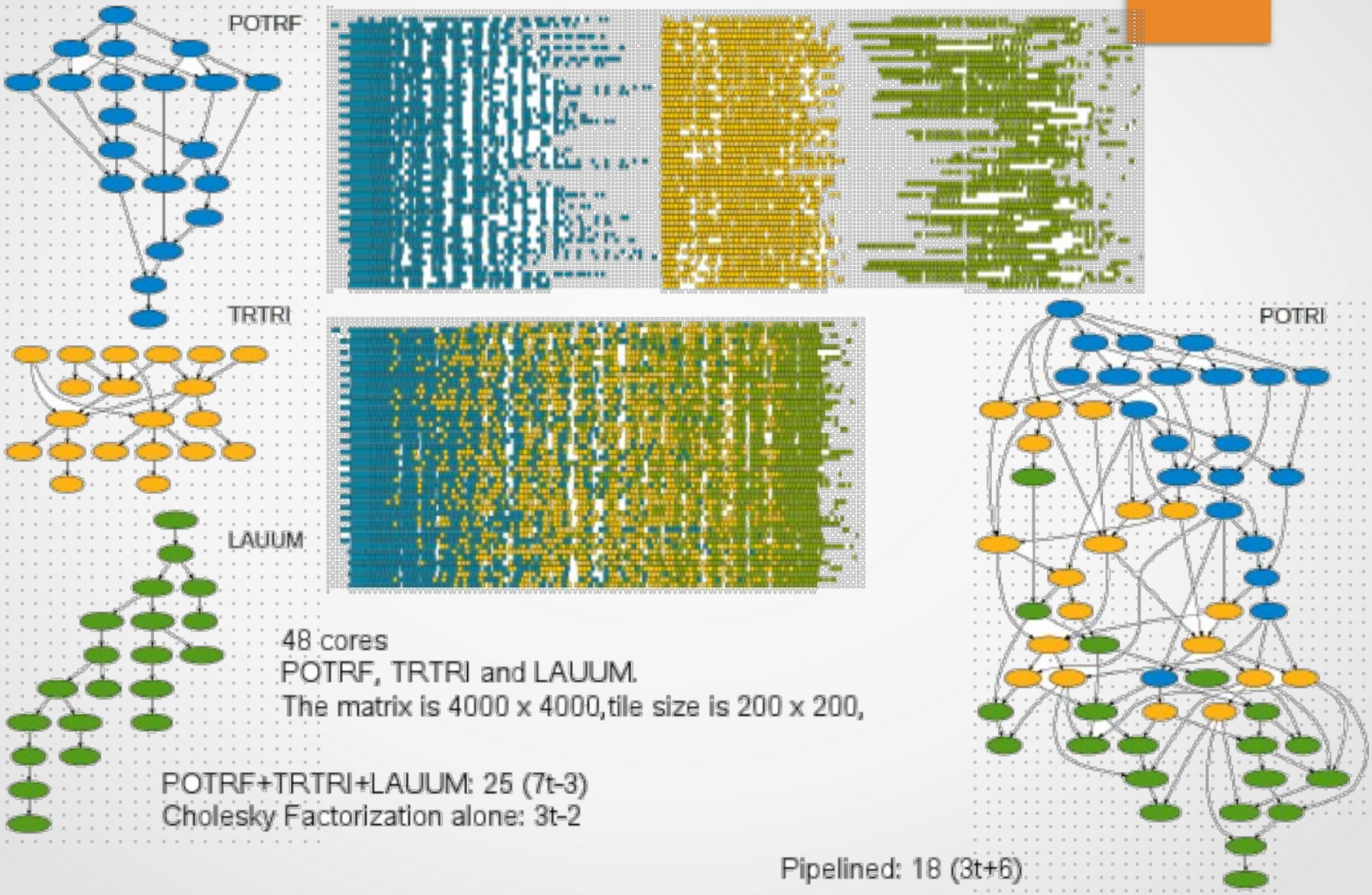
Cholesky : pipelining



48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200,

POTRF+TRTRI+LAUUM: 25 (7t-3)
Cholesky Factorization alone: 3t-2

Cholesky : pipelining



48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200,

POTRF+TRTRI+LAUUM: 25 (7t-3)
Cholesky Factorization alone: 3t-2

Pipelined: 18 (3t+6)

Tâches : discussion

+ parallélisme de données + tâches
+ choix dynamique meilleur CPU
+ recouvrement calcul / transfert
+ moins de sections séquentielles que fork-join
+ runtime => meilleure portabilité de performances
+ **ordre d'exécution ≠ ordre du code**

- plus complexe que fork-join
- surcoût exécution dynamique
  - De l'ordre de 10-100 µs **/!\ Granularité !!!**