Complexité I/O

Auguste Olivry, Guillaume Iooss, Fabrice Rastello

Équipe Inria CORSE, Grenoble

17 Mars 2022

Optimisation de programme

- Comment modifier un programme pour qu'il s'exécute le plus rapidement possible sur une machine?

Optimisation de programme

- Comment modifier un programme pour qu'il s'exécute le plus rapidement possible sur une machine?

- Aspects à prendre en compte:
  - Utiliser au maximum les unités de calcul
  - Acheminer les données demandées par ces unités de calcul

Optimisation de programme

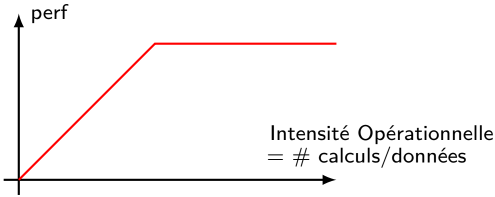
- Comment modifier un programme pour qu'il s'exécute le plus rapidement possible sur une machine?

- Aspects à prendre en compte:
  - Utiliser au maximum les unités de calcul
  - Acheminer les données demandées par ces unités de calcul

- Limites venant de l'architecture:
  - Puissance de calcul (nombre d'unités de calcul, fréquence, ...)
  - Bande-passante entre les mémoires (RAM, caches, registres)

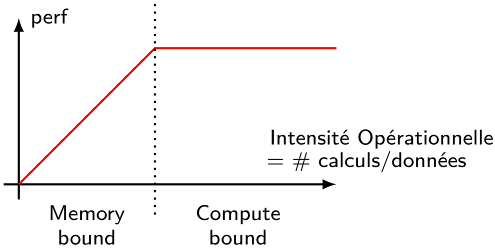
Roofline model

- Roofline model: modélise l'équilibre de la machine



Roofline model

- Roofline model: modélise l'équilibre de la machine

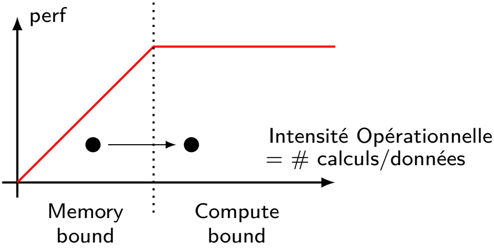


Programme memory/compute-bound

- *Memory-bound*: les transferts mémoires limitent la perf
- *Compute-bound*: les unités de calcul limitent la perf

Roofline model

- Roofline model: modélise l'équilibre de la machine



Programme memory/compute-bound

- *Memory-bound*: les transferts mémoires limitent la perf
- *Compute-bound*: les unités de calcul limitent la perf

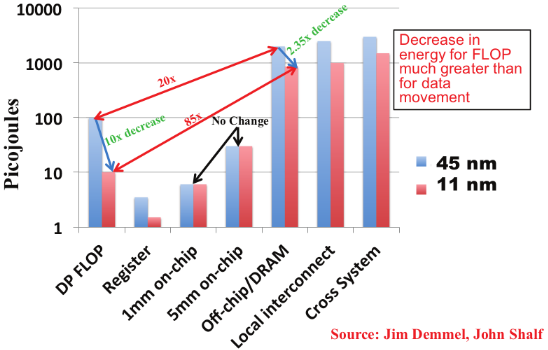
- Transformation: peut améliorer la réutilisation mémoire
  - $\Rightarrow$ Décalage vers la droite

Optimiser l'I/O

- Évolution de l'équilibre machine:
  - Intel 80286: 2 MIPS, 13 MB/s transfert RAM->CPU
  - Intel core i7: 50k MIPS, 16k MB/s transfert RAM->CPU

Optimiser l'I/O

- Évolution de l'équilibre machine:
  - Intel 80286: 2 MIPS, 13 MB/s transfert RAM->CPU
  - Intel core i7: 50k MIPS, 16k MB/s transfert RAM->CPU
- Évolution ratio énergie calcul/transfert données:



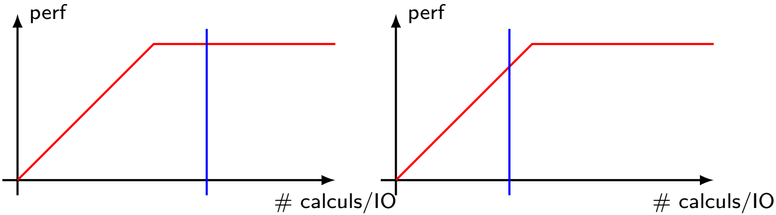
$\Rightarrow$ I/O de plus en plus critique/difficile à optimiser.

Limite algorithmique à l'intensité opérationnelle

- Transfo améliorant l'efficacité d'utilisation des données:
  - $\Rightarrow$ Décale vers la droite... mais jusqu'à quel point est-ce possible?

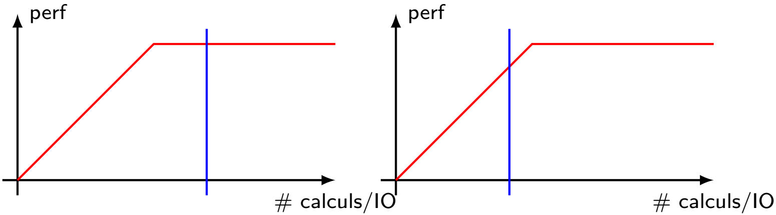
Limite algorithmique à l'intensité opérationnelle

- Transfo améliorant l'efficacité d'utilisation des données:
  $\Rightarrow$ Décale vers la droite... mais jusqu'à quel point est-ce possible?



Limite algorithmique à l'intensité opérationnelle

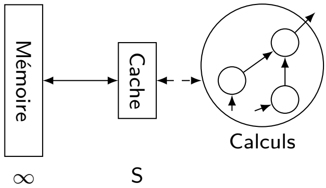
- Transfo améliorant l'efficacité d'utilisation des données:
  $\Rightarrow$ Décale vers la droite... mais jusqu'à quel point est-ce possible?



- Quel est le nombre maximum de calcul par I/O ?
  (OU) Quel est le nombre minimal d'I/O devant être fait?

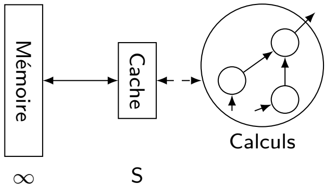
Complexité I/O - Définition

- Modèle mémoire à 2 niveaux:



Complexité I/O - Définition

- Modèle mémoire à 2 niveaux:

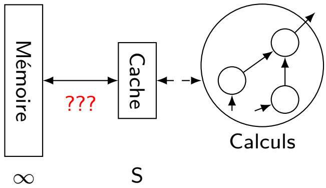


Coût I/O

- Nombre de transferts mémoire, pour un ordonnancement donné

Complexité I/O - Définition

- Modèle mémoire à 2 niveaux:



Coût I/O

- Nombre de transferts mémoire, pour un ordonnancement donné

Complexité I/O

- Nombre de transferts mémoire **min**, **pour tout** ordonnancement

$\Rightarrow$ Comment calculer la complexité I/O d'un programme?

Plan du reste de l'exposé

1. Introduction
   - Motivation générale
   - Complexité I/O
2. Red-white pebble game
3. Borne inférieure
4. Borne supérieure
5. Résultats

Modélisation des mouvements I/O

- Comment modéliser le nombre de transferts vers la mémoire rapide?
- **Red/White Pebble game** (variation de [Hong and Kung 1981])

Modélisation des mouvements I/O

- Comment modéliser le nombre de transferts vers la mémoire rapide?
- **Red/White Pebble game** (variation de [Hong and Kung 1981])

Principes généraux

- Computational Directed Acyclic Graph (CDAG): graphe des calculs d'un programme
  - Nœud: Entrées, sorties et opérations du programme
  - Arête: Dépendance de donnée entre calculs

Modélisation des mouvements I/O

- Comment modéliser le nombre de transferts vers la mémoire rapide?
- **Red/White Pebble game** (variation de [Hong and Kung 1981])

Principes généraux

- Computational Directed Acyclic Graph (CDAG):
  graphe des calculs d'un programme
  - Nœud: Entrées, sorties et opérations du programme
  - Arête: Dépendance de donnée entre calculs
- Jetons se placent sur les nœuds:
  - Jeton Rouge = Donnée actuellement dans la mémoire rapide
    Seulement S jetons rouges!
  - Jeton Blanc = Calcul effectué, donnée dans la mémoire lente

Modélisation des mouvements I/O

- Comment modéliser le nombre de transferts vers la mémoire rapide?
- **Red/White Pebble game** (variation de [Hong and Kung 1981])

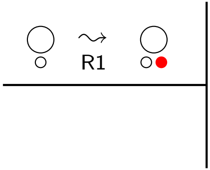
Principes généraux

- Computational Directed Acyclic Graph (CDAG):
  graphe des calculs d'un programme
  - Nœud: Entrées, sorties et opérations du programme
  - Arête: Dépendance de donnée entre calculs
- Jetons se placent sur les nœuds:
  - Jeton Rouge = Donnée actuellement dans la mémoire rapide
    Seulement S jetons rouges!
  - Jeton Blanc = Calcul effectué, donnée dans la mémoire lente
- Situation initiale: jetons blancs sur les nœuds d'entrée
- But du jeu: couvrir tous les nœuds avec des jetons blancs

Règles du jeu

Règle 1 - Load

On peut mettre un jeton rouge sur un nœud avec un jeton blanc.



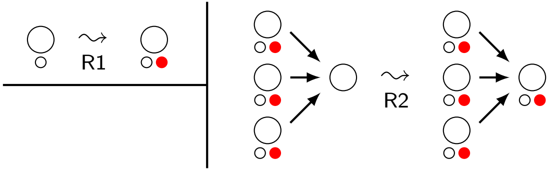
Règles du jeu

Règle 1 - Load

On peut mettre un jeton rouge sur un nœud avec un jeton blanc.

Règle 2 - Calcul

Si un nœud n'a pas de jeton blanc et que tous ses prédécesseurs ont un jeton rouge, on peut y mettre un rouge et un blanc.



Règles du jeu

Règle 1 - Load

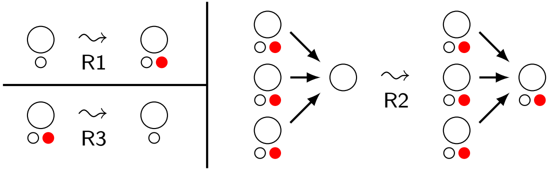
On peut mettre un jeton rouge sur un nœud avec un jeton blanc.

Règle 2 - Calcul

Si un nœud n'a pas de jeton blanc et que tous ses prédécesseurs ont un jeton rouge, on peut y mettre un rouge et un blanc.

Règle 3 - Oubli

On peut retirer un jeton rouge d'un nœud.



Règles du jeu

Règle 1 - Load

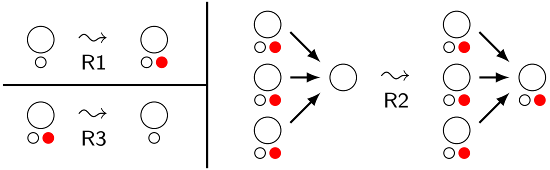
On peut mettre un jeton rouge sur un nœud avec un jeton blanc.

Règle 2 - Calcul

Si un nœud n'a pas de jeton blanc et que tous ses prédécesseurs ont un jeton rouge, on peut y mettre un rouge et un blanc.

Règle 3 - Oubli

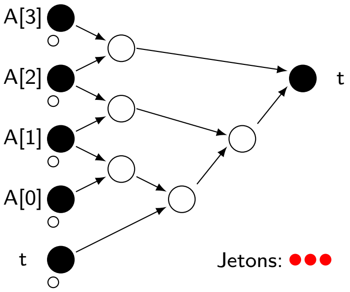
On peut retirer un jeton rouge d'un nœud.



- Pas de recalculs. Nombre Load = nombre de Règle 1.

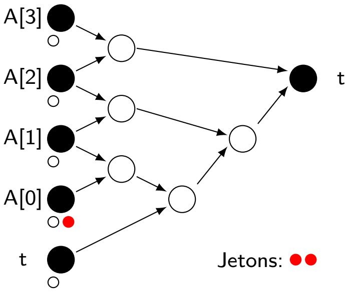
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



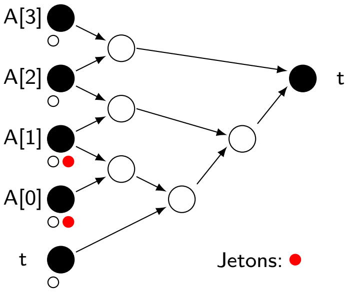
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



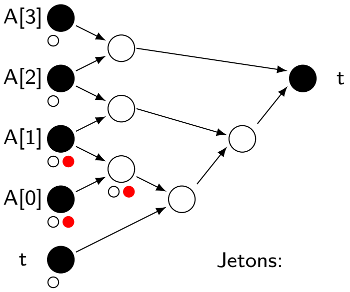
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



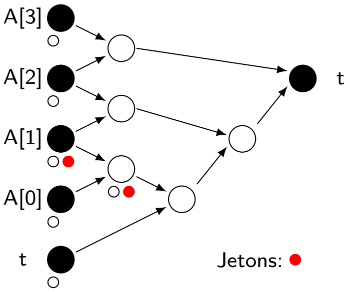
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



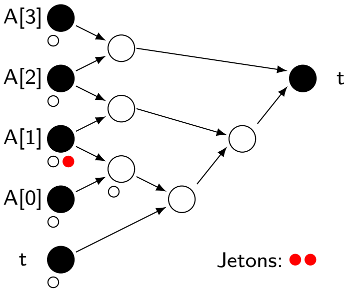
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



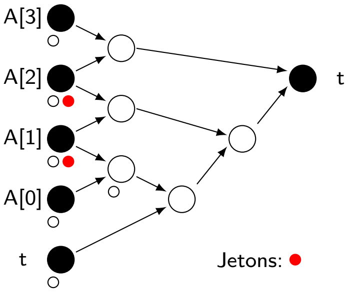
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



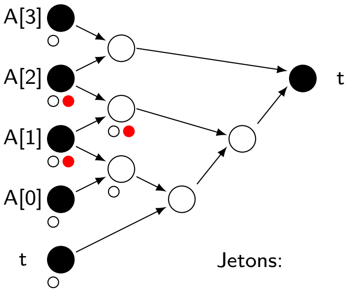
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



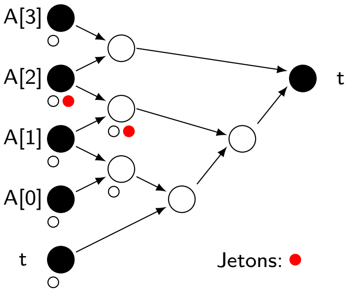
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



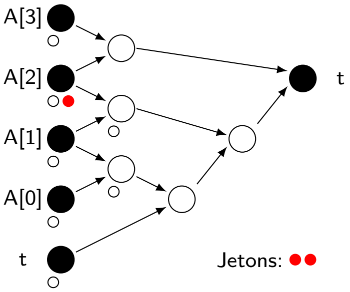
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



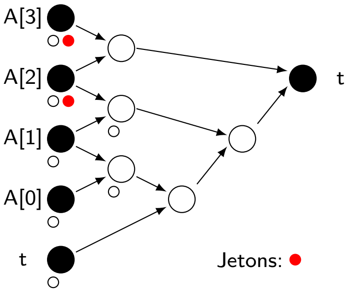
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



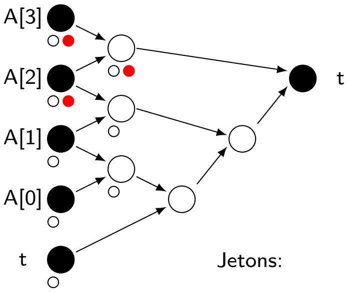
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



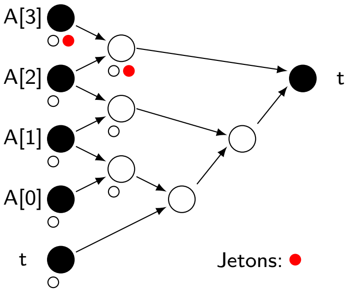
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



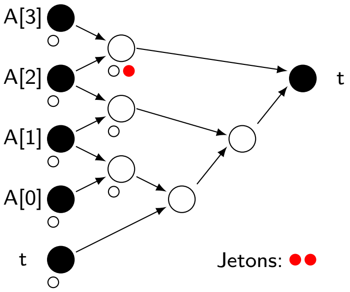
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



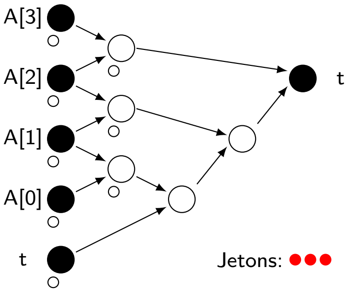
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



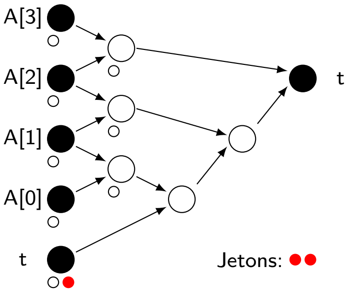
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



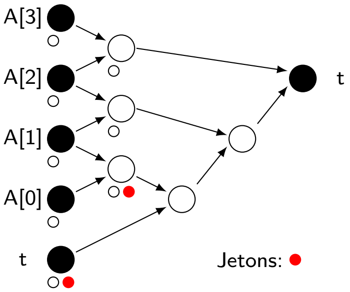
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



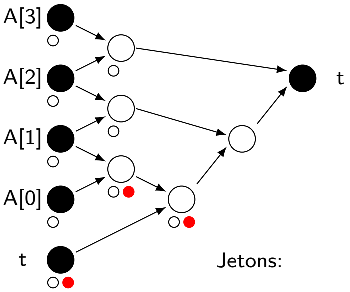
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



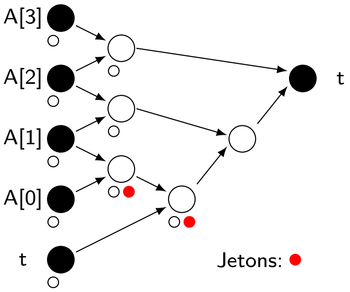
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



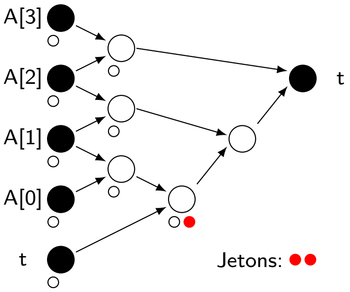
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



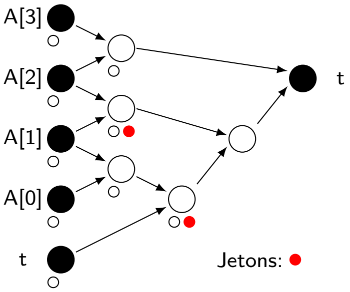
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



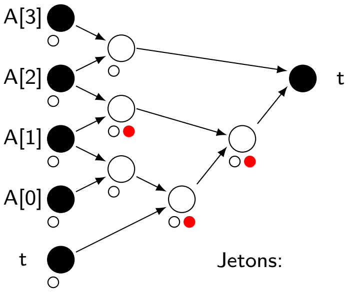
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



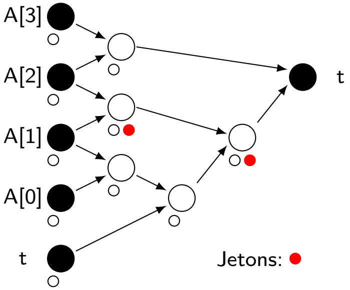
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



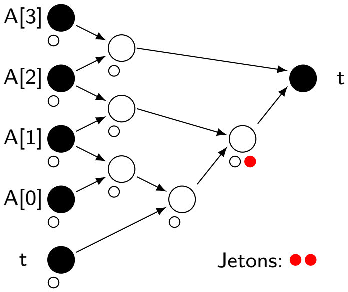
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



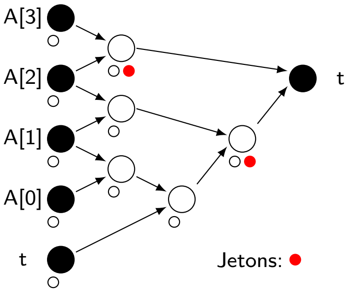
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



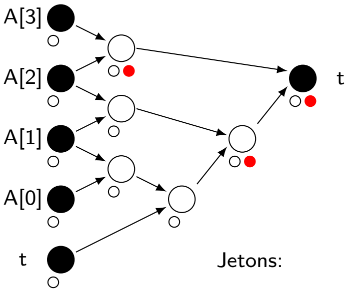
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



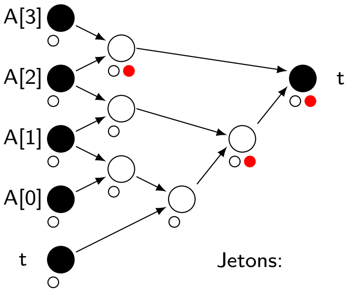
Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



Exemple de partie (avec animation)

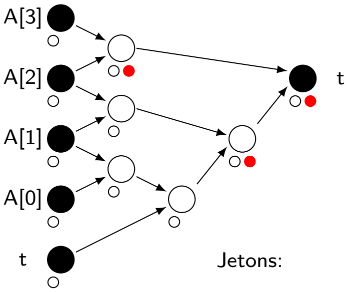
```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



Nombre total de loads: 14.

Exemple de partie (avec animation)

```
for (i=1; i<4; i++)
  t += A[i-1] + A[i];
```



Nombre total de loads: 14. (Il y a une solution à 9)

Problèmes (Challenges)

- **Problèmes:**
  - Besoin d'un minimum sur *toutes les parties*

Problèmes (Challenges)

- **Problèmes:**
  - Besoin d'un minimum sur *toutes les parties*
  - Taille du CDAG fréquemment paramétrique:

```
for (i=1; i<N; i++)
  t += A[i-1] + A[i];
```

    Nombre de Load dépend de ces paramètres.

Problèmes (Challenges)

- **Problèmes:**
  - Besoin d'un minimum sur *toutes les parties*
  - Taille du CDAG fréquemment paramétrique:

```
for (i=1; i<N; i++)
  t += A[i-1] + A[i];
```

    Nombre de Load dépend de ces paramètres.

- Infaisable sur des graphes quelconques

$\Rightarrow$ Se restreint à des graphes de programmes réguliers.

Programme polyédrique

Classe de programme: *programme polyédrique*

Conditions de boucle + fonctions d'accès affines

Example (Multiplication de matrice)

```
for (i=0; i<I; i++)
   for (j=0; j<J; i++) {
     for (k=0; k<K; i++)
S:       C[i,j] += A[i,k] * B[k,j];
   }
```

Domaine d'itération: $D_S = \{i, j, k \mid 0 \leq i < I,\ 0 \leq j < J,\ 0 \leq k < K\}$
Fonction d'accès: $f_C = (i, j, k- > i, j)$

$\Rightarrow$ Représentation mathématique précise et compacte du CDAG.

(Parenthèse opportuniste sur la compilation polyédrique)

Plan de l'exposé (en cours)

1. Introduction
   - Motivation générale
   - Complexité I/O
2. Red-white pebble game
3. Borne inférieure
4. Borne supérieure
5. Résultats

Borne inférieure d'une complexité I/O

- Considérons un programme polyédrique:
  Borne symbolique inférieure de sa complexité I/O ?
- Plusieurs méthodes de preuve: wavefront, 2S-partitions, ...
  - Efficacité dépend de la structure du graphe
- Ici: présentation de la méthode des 2S-partitions.

K-partition - Définition

En posant:

- CDAG $= (V, E)$.
- $I \subseteq V$: Nœuds d'entrée de V.
- V_i : sous-ensemble de V.

$InSet(V_i)$

Ensemble des nœuds hors de V_i et prédécesseurs de nœuds de V_i.

K-ensemble

Ensemble V de nœuds tel que $|InSet(V)| \leq K$

K-partition

K-partition

Partition $(V_i)_i$ de $(V - I)$, telle que:

- tous les V_i sont des K-ensembles
- Pas de cycles entre V_i

K-partition

K-partition

Partition $(V_i)_i$ de $(V - I)$, telle que:

- tous les V_i sont des K-ensembles
- Pas de cycles entre V_i

Utilité?

- Si $K = S + T$ (avec $S =$ taille de la mémoire rapide), au moins T loads dans le meilleur cas par ensemble.
- Possible de ramener tout jeu de jetons Rouge-Blanc à une décomposition en $2S$-partition.

Idée générale de la méthode des 2S-partitions

Lemma (Lien entre une 2S-partition et la complexité I/O)

Soit P le plus grand ensemble d'une 2S-partition,
et Q la complexité I/O d'un programme.
Alors:

$$Q \geq \left(\left\lceil \frac{|V|}{|P|} \right\rceil - 1 \right) \times S - |I|$$

Idée générale de la méthode des 2S-partitions

Lemma (Lien entre une 2S-partition et la complexité I/O)

Soit P le plus grand ensemble d'une 2S-partition,
et Q la complexité I/O d'un programme.
Alors:

$$Q \geq \left(\left\lceil \frac{|V|}{|P|} \right\rceil - 1 \right) \times S - |I|$$

Morceau restant: taille maximale d'un 2S-ensemble (càd de P) ?

Théorème de Loomis-Whitney (1)

Taille maximale d'un K-ensemble?

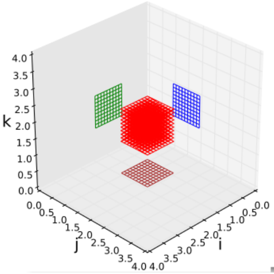
Théorème de Loomis-Whitney

Soit $E \subset R^d$ et $\phi_1, \ldots, \phi_d$ les projections canoniques.
Alors:

$$|E| \leq \prod_{j=1}^{d} |\phi_i(E)|^{1/(d-1)}$$

Théorème de Loomis-Whitney (2)

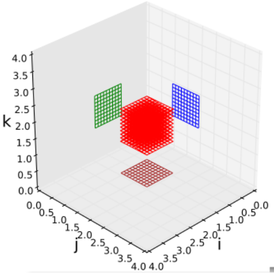
Example ($d = 3$):



$$|E| \leq |\phi_1(E)|^{1/2} \times |\phi_2(E)|^{1/2} \times |\phi_3(E)|^{1/2}$$

Théorème de Loomis-Whitney (2)

Example ($d = 3$):



$$|E| \leq |\phi_1(E)|^{1/2} \times |\phi_2(E)|^{1/2} \times |\phi_3(E)|^{1/2}$$

- **Intuition:** S'arranger pour que les $\phi_i(E)$ s'injectent dans une partie de $InSet(E)$ (qui est de taille bornée)

Exemple: multiplication de matrice (1)

```
for (i=0; i<I; i++)
   for (j=0; j<J; i++) {
     C[i,j] = 0;
     for (k=0; k<K; i++)
S:       C[i,j] += A[i,k] * B[k,j];
   }
```

- **CDAG:** cube de $2IJK$ nœuds, avec projections selon les axes i et j et chaîne de dépendance selon k (axe de réduction)

Exemple: multiplication de matrice (1)

```
for (i=0; i<I; i++)
   for (j=0; j<J; i++) {
     C[i,j] = 0;
     for (k=0; k<K; i++)
S:        C[i,j] += A[i,k] * B[k,j];
   }
```

- **CDAG:** cube de $2IJK$ nœuds, avec projections selon les axes i et j et chaîne de dépendance selon k (axe de réduction)
- Soit P un K-ensemble de cet espace:

$$|P| \leq |\phi_1(P)|^{1/2} \times |\phi_2(P)|^{1/2} \times |\phi_3(P)|^{1/2}$$

avec les ϕ_i projections canoniques.

Exemple: multiplication de matrice (1)

```
for (i=0; i<I; i++)
   for (j=0; j<J; i++) {
     C[i,j] = 0;
     for (k=0; k<K; i++)
S:       C[i,j] += A[i,k] * B[k,j];
   }
```

- **CDAG:** cube de $2IJK$ nœuds, avec projections selon les axes i et j et chaîne de dépendance selon k (axe de réduction)
- Soit P un K-ensemble de cet espace:

$$|P| \leq |\phi_1(P)|^{1/2} \times |\phi_2(P)|^{1/2} \times |\phi_3(P)|^{1/2}$$

  avec les ϕ_i projections canoniques.
- Or, $|\phi_i(P)| \leq |InSet(P)| \leq K$.

Exemple: multiplication de matrice (1)

```
for (i=0; i<I; i++)
   for (j=0; j<J; i++) {
     C[i,j] = 0;
     for (k=0; k<K; i++)
S:       C[i,j] += A[i,k] * B[k,j];
   }
```

- **CDAG:** cube de $2IJK$ nœuds, avec projections selon les axes i et j et chaîne de dépendance selon k (axe de réduction)
- Soit P un K-ensemble de cet espace:

$$|P| \leq |\phi_1(P)|^{1/2} \times |\phi_2(P)|^{1/2} \times |\phi_3(P)|^{1/2}$$

  avec les ϕ_i projections canoniques.
- Or, $|\phi_i(P)| \leq |InSet(P)| \leq K$.
- Donc, $|P| \leq K^{3/2}$.

Exemple: multiplication de matrice (2)

- Considérons une (2S)-partition:

$$Q \geq \left(\left\lceil \frac{|V|}{|P|} \right\rceil - 1 \right) \times S - |I|$$

Exemple: multiplication de matrice (2)

- Considérons une (2S)-partition:

$$Q \geq \left(\left\lceil \frac{|V|}{|P|} \right\rceil - 1 \right) \times S - |I|$$

- On a trouvé que $|P| \leq (2S)^{3/2}$ et $|V| = 2IJK$.

$$Q \geq \left(\left\lceil \frac{2IJK}{(2S)^{3/2}} \right\rceil - 1 \right) \times S - IK - JK$$

Exemple: multiplication de matrice (2)

- Considérons une (2S)-partition:

$$Q \geq \left(\left\lceil \frac{|V|}{|P|} \right\rceil - 1 \right) \times S - |I|$$

- On a trouvé que $|P| \leq (2S)^{3/2}$ et $|V| = 2IJK$.

$$Q \geq \left(\left\lceil \frac{2IJK}{(2S)^{3/2}} \right\rceil - 1 \right) \times S - IK - JK$$

- Borne asymptotique:

$$Q \geq \frac{IJK}{\sqrt{2S}}$$

Exemple: multiplication de matrice (2)

- Considérons une (2S)-partition:

$$Q \geq \left(\left\lceil \frac{|V|}{|P|} \right\rceil - 1 \right) \times S - |I|$$

- On a trouvé que $|P| \leq (2S)^{3/2}$ et $|V| = 2IJK$.

$$Q \geq \left(\left\lceil \frac{2IJK}{(2S)^{3/2}} \right\rceil - 1 \right) \times S - IK - JK$$

- Borne asymptotique:

$$Q \geq \frac{IJK}{\sqrt{2S}}$$

- **Raffinement:** Peut améliorer borne sur $|P|$ (facteur $2^{3/2}$):

$$Q \geq \frac{2IJK}{\sqrt{S}}$$

… ce dont je n'ai pas parlé.

- Raffinement de la borne sup sur la taille d'un K-ensemble
    - Projection dans des zones de l'InSet disjointes

... ce dont je n'ai pas parlé.

- Raffinement de la borne sup sur la taille d'un K-ensemble
  - Projection dans des zones de l'InSet disjointes
  - "Petites dimensions" (ex: convolution)

… ce dont je n'ai pas parlé.

- Raffinement de la borne sup sur la taille d'un K-ensemble
  - Projection dans des zones de l'InSet disjointes
  - "Petites dimensions" (ex: convolution)

- **Théorème de Brascamp-Lieb:** généralisation de Loomis-Whitney pour tout homomorphisme de groupe ϕ_i

  $\Rightarrow$ Permet des projections selon des directions arbitraires

  $\Rightarrow$ Analyse les dépendances du programme pour les trouver.

... ce dont je n'ai pas parlé.

- Raffinement de la borne sup sur la taille d'un K-ensemble
  - Projection dans des zones de l'InSet disjointes
  - "Petites dimensions" (ex: convolution)

- **Théorème de Brascamp-Lieb:** généralisation de Loomis-Whitney pour tout homomorphisme de groupe ϕ_i

  $\Rightarrow$ Permet des projections selon des directions arbitraires

  $\Rightarrow$ Analyse les dépendances du programme pour les trouver.

- Programme avec des nids de boucles non parfaitement imbriquées

  $\Rightarrow$ Combinaison de bornes venant de plusieurs sections du graphe

Plan de l'exposé (toujours en cours)

1. Introduction
   - Motivation générale
   - Complexité I/O
2. Red-white pebble game
3. Borne inférieure
4. Borne supérieure
5. Résultats

Borne supérieure d'une complexité I/O

- **Borne inférieure:** à quel point proche de la vraie valeur?

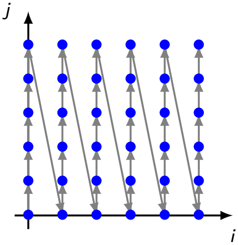
  $\Rightarrow$ Veut un encadrement serré de la complexité I/O.

Borne supérieure d'une complexité I/O

- **Borne inférieure:** à quel point proche de la vraie valeur?
  $\Rightarrow$ Veut un encadrement serré de la complexité I/O.

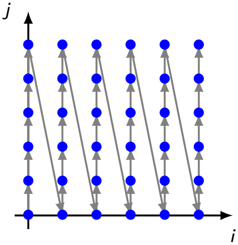
- **Borne supérieure:** essaye de trouver une partie (un ordonnancement) qui minimise le nombre de transfert.
  $\Rightarrow$ Indication sur comment optimiser un programme pour minimiser l'I/O.

Background - tuilage de programme

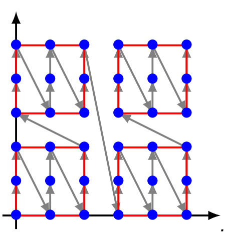



```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
   C[i,j] = A[i] + B[j];
```

Background - tuilage de programme



Tuilage ⇝



i

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    C[i,j] = A[i] + B[j];
```

```
for (i_t=0; i_t<=N; i_t+=3)
for (j_t=0; j_t<=N; j_t+=3)
  for (i=3*i_t; i<min(3*(i_t+1),N); i++)
  for (j=3*j_t; j<min(3*(j_t+1),N); j++)
   C[i,j] = A[i] + B[j];
```

- **Tuilage:** groupe des itérations en tuiles atomiques
- Localité des données, introduit niveau de granularité

Background - autres notions

Emprunte mémoire (Footprint)

Ensemble des cases mémoires accédées par une portion de programme.

Background - autres notions

Emprunte mémoire (Footprint)

Ensemble des cases mémoires accédées par une portion de programme.

Réutilisation mémoire

Utilisation multiple d'une donnée qui reste dans la mémoire rapide

Accès de tableau: dimension de réutilisation (ex: `A[i,k]` $\rightsquigarrow j$)

Background - autres notions

Emprunte mémoire (Footprint)

Ensemble des cases mémoires accédées par une portion de programme.

Réutilisation mémoire

Utilisation multiple d'une donnée qui reste dans la mémoire rapide

Accès de tableau: dimension de réutilisation (ex: `A[i,k]` $\rightsquigarrow j$)

Peut calculer ces informations avec le modèle polyédrique.

Espace des implémentations

- Classe de programme encore plus simple (tensor-like):
  - Boucles parfaitement imbriquées
  - Boucles pouvant être permutées

  $\Rightarrow$ Tuilable avec des tuiles rectangulaires

Espace des implémentations

- Classe de programme encore plus simple (tensor-like):
  - Boucles parfaitement imbriquées
  - Boucles pouvant être permutées
  - $\Rightarrow$ Tuilable avec des tuiles rectangulaires

- Forme des implémentations recherchées:
  - Programme tuilé
  - Tailles des tuiles sont des paramètres

Example

```
for (i1=0; i1<I; i1+=Ti)
  for (j1=0; j1<J; j1+=Tj)
-----------------------------------
    for (i=i1; i<min(i1+Ti,I); i++)
      for (j=j1; i<min(j1+Tj,J); j++)
        S(i,j);
```

Espace des implémentations

- Classe de programme encore plus simple (tensor-like):
  - Boucles parfaitement imbriquées
  - Boucles pouvant être permutées
  - ⇒ Tuilable avec des tuiles rectangulaires

- Forme des implémentations recherchées:
  - Programme tuilé
  - Tailles des tuiles sont des paramètres

Example

```
for (i1=0; i1<I; i1+=Ti)
  for (j1=0; j1<J; j1+=Tj)
-----------------------------------------
    for (i=i1; i<min(i1+Ti,I); i++)
      for (j=j1; i<min(j1+Tj,J); j++)
        S(i,j);
```

⇒ Quelle permutation de boucle? Tailles de tuile?

Choix de la permutation de boucle

- **Contrainte:** emprunte mémoire de la tuile interne rentre dans la mémoire rapide.

  $\Rightarrow$ Permutation des boucles internes n'importe pas.

Choix de la permutation de boucle

- **Contrainte:** emprunte mémoire de la tuile interne rentre dans la mémoire rapide.

  $\Rightarrow$ Permutation des boucles internes n'importe pas.

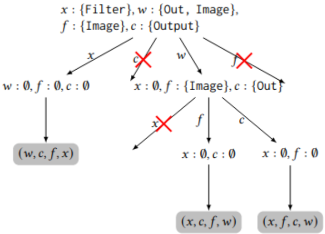
- Permutation de boucle au dessus? (beaucoup de choix)
  - Maximiser la réutilisation des données
  - Certaines dimensions sont meilleures que d'autres

Choix de la permutation de boucle

- **Contrainte:** emprunte mémoire de la tuile interne rentre dans la mémoire rapide.
  - $\Rightarrow$ Permutation des boucles internes n'importe pas.

- Permutation de boucle au dessus? (beaucoup de choix)
  - Maximiser la réutilisation des données
  - Certaines dimensions sont meilleures que d'autres

- **Algorithme de sélection de permutation:** (sur un exemple)



Choix des taille de tuile - Fonction de coût

- Pour une permutation donnée, minimiser l'I/O:

Example (Matmult, permutation (i, j, k))

```
for (i1=0; i1<N_i; i1+=Ti)
   for (j1=0; j1<N_j; j1+=Tj)
     for (k=0; k<N_k; k++)
-------------------------------------------------
       for (i=i1; i<min(i1+Ti,I); i++)
         for (j=j1; i<min(j1+Tj,J); j++)
           C[i,j] += A[i,k] × B[k,j];
```

- Suppose que $T_i T_j + T_i + T_j < S$

Choix des taille de tuile - Fonction de coût

- Pour une permutation donnée, minimiser l'I/O:

Example (Matmult, permutation (i, j, k))

```
for (i1=0; i1<N_i; i1+=Ti)
  for (j1=0; j1<N_j; j1+=Tj)
    for (k=0; k<N_k; k++)
-----------------------------------------
      for (i=i1; i<min(i1+Ti,I); i++)
        for (j=j1; i<min(j1+Tj,J); j++)
          C[i,j] += A[i,k] × B[k,j];
```

- Suppose que $T_i T_j + T_i + T_j < S$
- Boucle sur k:
  - (k=0) Tout miss: $T_i T_j + T_i + T_j$
  - (k>0) Réutilisation de C[i,j]: $T_i + T_j$

$\Rightarrow$ $(T_i T_j + T_i + T_j) + (N_k - 1)(T_i + T_j) = (T_i T_j + N_k T_i + N_k T_j)$

Choix des taille de tuile - Fonction de coût

- Pour une permutation donnée, minimiser l'I/O:

Example (Matmult, permutation (i, j, k))

```
for (i1=0; i1<N_i; i1+=Ti)
  for (j1=0; j1<N_j; j1+=Tj)
    for (k=0; k<N_k; k++)
-----------------------------------------
      for (i=i1; i<min(i1+Ti,I); i++)
        for (j=j1; i<min(j1+Tj,J); j++)
          C[i,j] += A[i,k] × B[k,j];
```

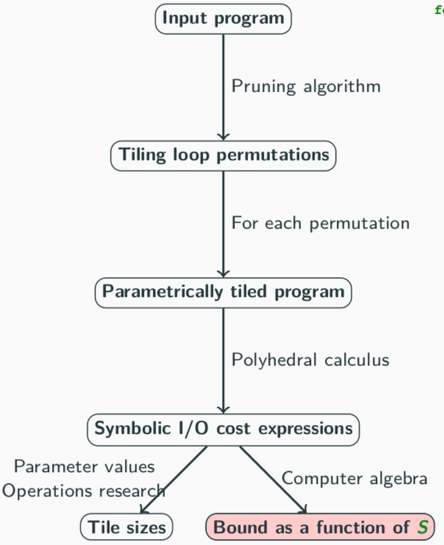
- Suppose que $T_i T_j + T_i + T_j < S$
- Boucle sur k:
  - (k=0) Tout miss: $T_i T_j + T_i + T_j$
  - (k>0) Réutilisation de C[i,j]: $T_i + T_j$

$\Rightarrow (T_i T_j + T_i + T_j) + (N_k - 1)(T_i + T_j) = (T_i T_j + N_k T_i + N_k T_j)$

- Avec les boucles sur i1 et j1 (tout le programme):

$$\frac{N_i}{T_i}\frac{N_j}{T_j}(T_i T_j + N_k T_i + N_k T_j) = N_i N_j N_k \left(\frac{1}{T_i} + \frac{1}{T_j} + \frac{1}{N_k}\right)$$

Flot complet de IOUB - Multiplication de matrice




```
for(i = 0; i < Ni; i++)
  for(j = 0; j < Nj; j++)
    for(k = 0; k < Nk; k++)
      C[i][j] += A[i][k] * B[k][j];
```

$$\{\boxed{(i,j,k)}, (i,k,j), (k,j,i)\}$$

```
for(i1 = 0; i1 < Ni; i1+=Ti)
  for(j1 = 0; j1 < Nj; j1+=Tj)
    for(k = 0; k < Nk; k++)
      for(i = i1; i < i1+Ti; i++)
        for(j = j1; j < j1+Tj; j++)
          C[i][j] += A[i][k] * B[k][j];
```

$$IO = N_i N_j N_k \left(\frac{1}{T_i} + \frac{1}{T_j} + \frac{1}{N_k} \right)$$

$$T_i T_j + T_i + T_j \leq S$$

$$UB = N_i N_j \left(\frac{2N_k}{\sqrt{S+1}-1} + 1 \right)$$

Automatisation

- Borne inférieure: **IOLB**
  Sur tout programme polyédrique

- Borne supérieure: **IOUB**
  Sur une sous-classe de programme "tensor-like"
  (limitation du solver)

- Deux protos disponible en ligne:
  `https://iocomplexity.corse.inria.fr/`

Résultats IOLB sur Polybench (UB: calcul manuel)

kernel	# input data	# ops	Q_{low}^{∞}	$OI_{\text{manual}} \leq OI \leq OI_{\text{up}}$	ratio
2mm	$N_iN_k + N_kN_j + N_jN_l + N_iN_l$	$2(N_iN_jN_k + N_iN_jN_l)$	$2(N_iN_jN_k + N_iN_jN_l)/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
3mm	$N_iN_k + N_kN_j + N_jN_m + N_mN_l$	$2(N_iN_jN_k + N_jN_lN_m + N_iN_jN_l)$	$2(N_iN_jN_k + N_iN_jN_l + N_jN_lN_m)/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
cholesky	$\frac{1}{2}N^2$	$\frac{1}{3}N^3$	$\frac{1}{6}N^3/\sqrt{S}$	$\sqrt{S} \leq OI \leq 2\sqrt{S}$	2
correlation	MN	M^2N	$\frac{1}{2}M^2N/\sqrt{S}$	$\sqrt{S} \leq OI \leq 2\sqrt{S}$	2
covariance	MN	M^2N	$\frac{1}{2}M^2N/\sqrt{S}$	$\sqrt{S} \leq OI \leq 2\sqrt{S}$	2
doitgen	$N_pN_qN_r + N_p^2$	$2N_p^2N_qN_r$	$2N_p^2N_qN_r/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
fdtd-2d	$3N_xN_y + T$	$11N_xN_yT$	$\frac{2}{3\sqrt{3}}N_xN_yT/\sqrt{S}$	$\frac{11}{24}\sqrt{3}\sqrt{S} \leq OI \leq \frac{33}{2}\sqrt{3}\sqrt{S}$	36
floyd-warshall	N^2	$2N^3$	$2N^3/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
gemm	$N_iN_j + N_jN_k + N_iN_k$	$2N_iN_jN_k$	$2N_iN_jN_k/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
heat-3d	N^3	$30N^3T$	$\frac{3}{8}\sqrt[3]{2}N^3T/\sqrt[3]{S}$	$\frac{5}{2}\sqrt[3]{S} \leq OI \leq 40 \cdot 2^{2/3}\sqrt[3]{S}$	$16 \cdot 2^{2/3}$
jacobi-1d	N	$6NT$	$\frac{1}{4}NT/S$	$\frac{3}{2}S \leq OI \leq 24S$	16
jacobi-2d	N^2	$10N^2T$	$\frac{2}{3\sqrt{3}}N^2T/\sqrt{S}$	$\frac{5}{4}\sqrt{S} \leq OI \leq 15\sqrt{3}\sqrt{S}$	$12\sqrt{3}$
lu	N^2	$\frac{2}{3}N^3$	$\frac{2}{3}N^3/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
ludcmp	N^2	$\frac{2}{3}N^3$	$\frac{2}{3}N^3/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
seidel-2d	N^2	$9N^2T$	$\frac{2}{3\sqrt{3}}N^2T/\sqrt{S}$	$\frac{9}{4}\sqrt{S} \leq OI \leq \frac{27\sqrt{3}}{2}\sqrt{S}$	$6\sqrt{3}$
symm	$\frac{1}{2}M^2 + 2MN$	$2M^2N$	$2M^2N/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
syr2k	$\frac{1}{2}N^2 + 2MN$	$2MN^2$	$MN^2/\sqrt{S}$	$\sqrt{S} \leq OI \leq 2\sqrt{S}$	2
syrk	$\frac{1}{2}N^2 + MN$	MN^2	$\frac{1}{2}MN^2/\sqrt{S}$	$\sqrt{S} \leq OI \leq 2\sqrt{S}$	2
trmm	$\frac{1}{2}M^2 + MN$	M^2N	$M^2N/\sqrt{S}$	$\sqrt{S} \leq OI \leq \sqrt{S}$	1
atax	MN	$4MN$	MN	$4 \leq OI \leq 4$	1
bicg	MN	$4MN$	MN	$4 \leq OI \leq 4$	1
deriche	HW	$32HW$	HW	$\frac{16}{3} \leq OI \leq 32$	6
gemver	N^2	$10N^2$	N^2	$5 \leq OI \leq 10$	2
gesummv	$2N^2$	$4N^2$	$2N^2$	$2 \leq OI \leq 2$	1
mvt	N^2	$4N^2$	N^2	$4 \leq OI \leq 4$	1
trisolv	$\frac{1}{2}N^2$	N^2	$\frac{1}{2}N^2$	$2 \leq OI \leq 2$	1
adi	N^2	$30N^2T$	N^2T	$5 \leq OI \leq 30$	6
durbin	N	$2N^2$	$\frac{1}{2}N^2$	$\frac{2}{3} \leq OI \leq 4$	6
gramschmidt	MN	$2MN^2$	$MN^2/\sqrt{S}$	$1 \leq OI \leq 2\sqrt{S}$	$2\sqrt{S}$
nussinov	$\frac{1}{2}N^2$	$\frac{1}{3}N^3$	$\frac{1}{6}N^3/\sqrt{S}$	$1 \leq OI \leq 2\sqrt{S}$	$2\sqrt{S}$

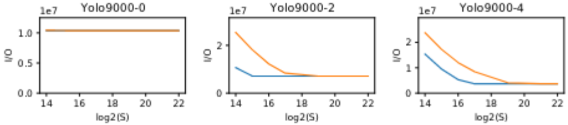
Tight bound for 14 kernels out of 30

Résultats IOUB

TC ab-ac-cb	$UB = \frac{2ABC}{\sqrt{S+1}-1} + BC$
	$LB = \max\left(AB + AC + CB, -2 + C - S + 2A + 2B + \frac{2AB(C-1)}{\sqrt{S}}\right)$
TC abcd-aebf-fdec	$UB = \frac{2ABCDEF}{\sqrt{S+1}-1} + CDEF$
	$LB = \max\left(ABCD + AEBF + FDEC, -3 + EF - S + 3AB + 3CD - ABCD + \frac{2ABCD(EF-1)}{\sqrt{S}}\right)$
2D Convolution	$UB = CFHWXY\left(\frac{1}{XY} + \frac{1}{H\Delta W} + \frac{(H+\Delta-1)(W+X-1)}{H\Delta^2 WX}\right)$ where $\Delta = \frac{-HW+W+\sqrt{H^2W^2+4HSW-2HW^2+4SW+4S+W^2}}{2(HW+W+1)}$
	$LB = \max\Big(BC(Y+H-1)(X+W-1) + BFXY + FCHW,$ $-2 - S + C + 4F + BY + BX + 2BXY - 2BXYF + \frac{BFXY(WHC-1)}{S},$ $-2 - S + C + 4F + BY + BX + 2BXY - 2BXFY + \frac{2BXYCF\sqrt{HW}}{\sqrt{S}} - \frac{2BXYF}{\sqrt{HWS}},$ $-2 - S + C + 4F + BY + BX + 2BXY - 2BXFY + \frac{2XYCF\sqrt{BHW}}{\sqrt{S}} - \frac{2XYF\sqrt{B}}{\sqrt{HWS}}\Big)$

Résultats IOUB

TC ab-ac-cb	$\mathrm{UB} = \frac{2ABC}{\sqrt{S+1}-1} + BC$
	$\mathrm{LB} = \max\left(AB + AC + CB, -2 + C - S + 2A + 2B + \frac{2AB(C-1)}{\sqrt{S}}\right)$
TC abcd-aebf-fdec	$\mathrm{UB} = \frac{2ABCDEF}{\sqrt{S+1}-1} + CDEF$
	$\mathrm{LB} = \max\left(ABCD + AEBF + FDEC, -3 + EF - S + 3AB + 3CD - ABCD + \frac{2ABCD(EF-1)}{\sqrt{S}}\right)$
2D Convolution	$\mathrm{UB} = CFHWXY\left(\frac{1}{XY} + \frac{1}{H\Delta W} + \frac{(H+\Delta-1)(W+X-1)}{H\Delta^2 WX}\right)$ where $\Delta = \frac{-HW+W+\sqrt{H^2W^2+4HSW-2HW^2+4SW+4S+W^2}}{2(HW+W+1)}$
	$\mathrm{LB} = \max\Big(BC(Y+H-1)(X+W-1) + BFXY + FCHW,$ $-2 - S + C + 4F + BY + BX + 2BXY - 2BXYF + \frac{BFXY(WHC-1)}{S},$ $-2 - S + C + 4F + BY + BX + 2BXY - 2BXFY + \frac{2BXYCF\sqrt{HW}}{\sqrt{S}} - \frac{2BXYF}{\sqrt{HWS}},$ $-2 - S + C + 4F + BY + BX + 2BXY - 2BXFY + \frac{2XYCF\sqrt{BHW}}{\sqrt{S}} - \frac{2XYF\sqrt{B}}{\sqrt{HWS}}\Big)$



Directions en cours/futures (aka TODO-list)

- Amélioration de borne inférieure:
  - Calculs de stencils: facteur multiplicatif
  - gramschmidt + nussinov : ratio à améliorer

- Utilisation de l'ordonnancement trouvé par IOUB
  - Possible d'appliquer l'algo sur plusieurs niveaux mémoire
  - Modèle idéal (scratchpad) $\neq$ Vrai cache
  - Utilisable en première approximation?

Merci de votre attention...

... Avez-vous des questions?