Introduction aux outils de synthèse de haut niveau

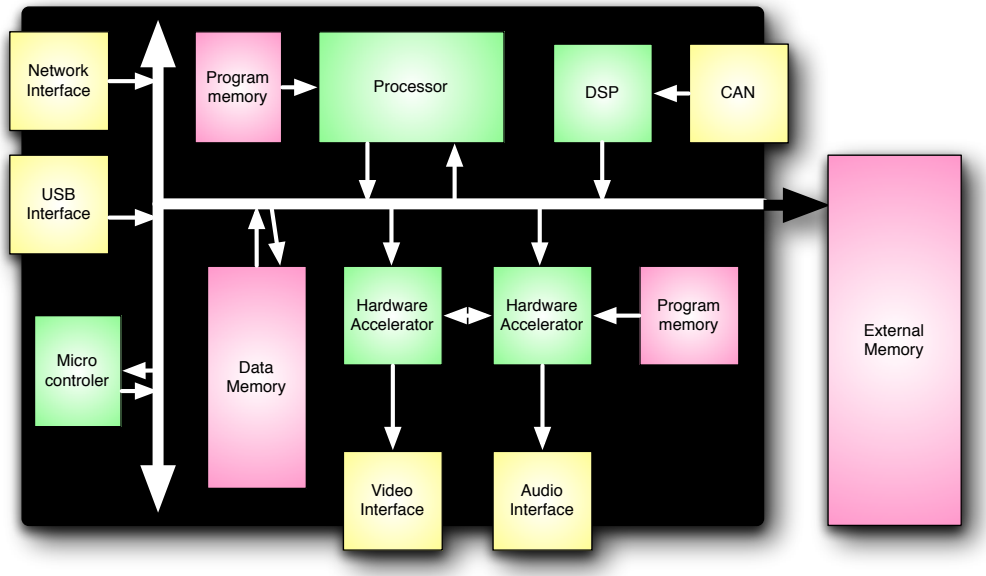
Bertrand LE GAL
[bertrand.legal@ims-bordeaux.fr]

Filière Electronique - 3ème année
ENSEIRB-MATMECA - Bordeaux INP
Talence, France

Exemples introductifs

Problématiques de conception

- Complexité grandissante
- Hétérogénéité :
  - HW/SW
  - Contrôle/Traitement
  - Numérique / Analogique
  - Technologie
- Contraintes :
  - Coût $
  - Temps de conception
  - Surface
  - Consommation



Cas d'étude pour une application de compression vidéo

Compression vidéo h264

Cas d'étude pour une application de compression vidéo

Compression vidéo h264

Développement, Temps réel

Qualité, temps réel

Coût, Qualité

Coût, Energie

Cas d'étude pour une application de compression vidéo

Compression vidéo h264

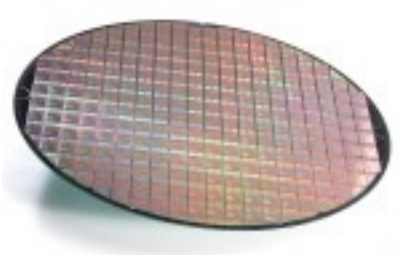
Développement, Temps réel

Qualité, temps réel

Coût, Qualité

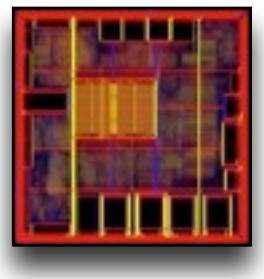
Coût, Energie

Processeur généraliste + programme



ASIC dédié hautes performances

ASIC dédié et partagé performances moyennes



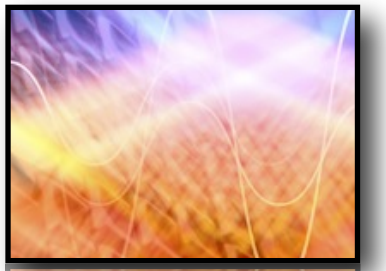
Sous système (HW/SW) faibles performances

Exemple d'implantations dépendantes de la technologie cible

Chaîne de filtrage numérique pour la décimation du signal

Exemple d'implantations dépendantes de la technologie cible

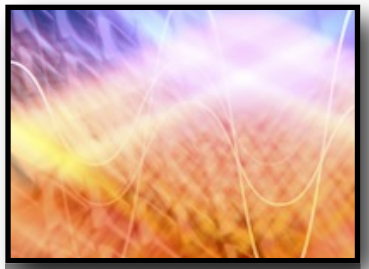
Chaîne de filtrage numérique pour la décimation du signal



Calcul de l'ordre et des coefficients du filtre numérique

Exemple d'implantations dépendantes de la technologie cible

Chaîne de filtrage numérique pour la décimation du signal

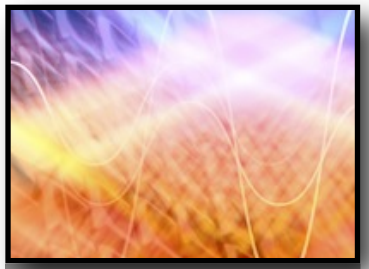


Calcul de l'ordre et des coefficients du filtre numérique

Contraintes :
- 200MBits/sec
- Clock = 100MHz

Exemple d'implantations dépendantes de la technologie cible

Chaîne de filtrage numérique pour la décimation du signal



Calcul de l'ordre et des coefficients du filtre numérique

Contraintes :
- 200MBits/sec
- Clock = 100MHz

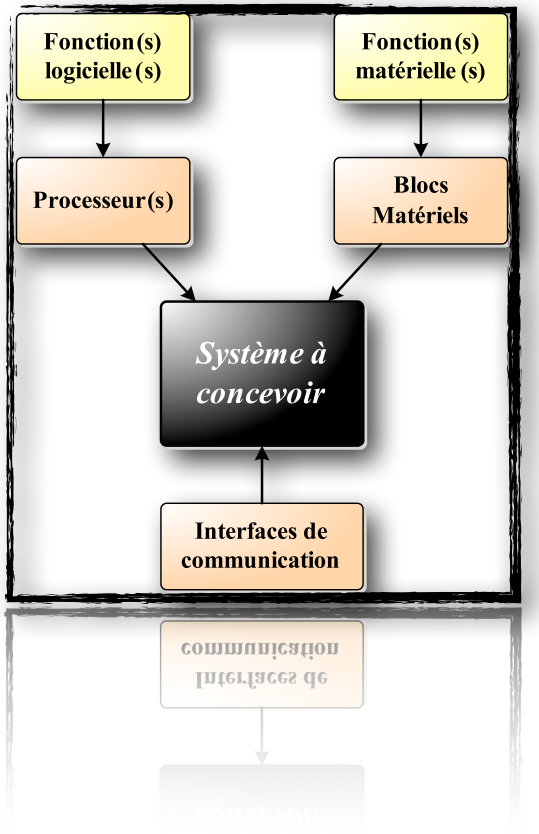
Ressources nécessaires :
- 9 x Multiplieurs
- 5 x Additionneur

Ressources nécessaires :
- 16 x Multiplieurs
- 12 x Additionneur

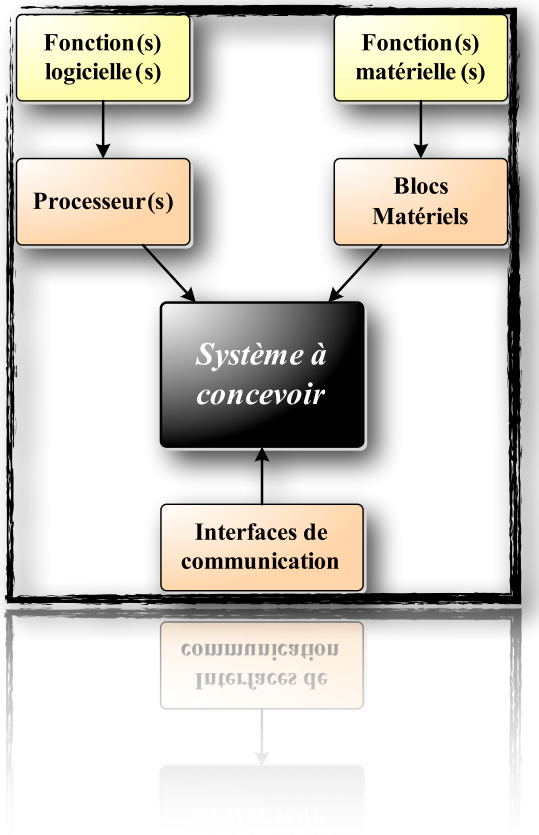
Ressources nécessaires :
- 32 x Multiplieurs
- 18 x Additionneur

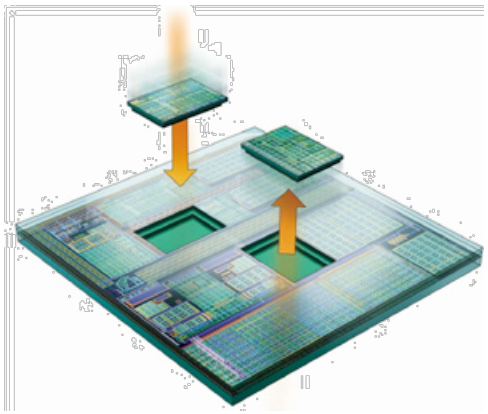
Ressources nécessaires :
- 10 x Multiplieurs
- 6 x Additionneur

Les différentes pistes de progrès envisagées



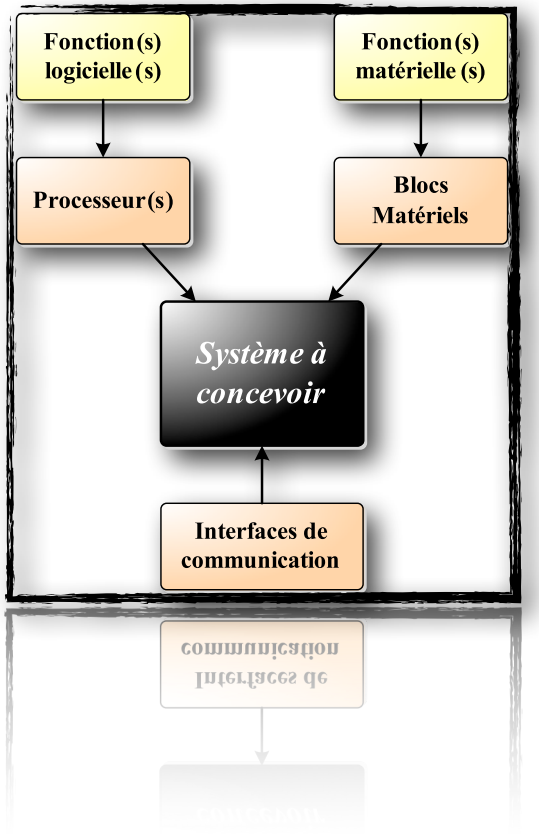
Les différentes pistes de progrès envisagées

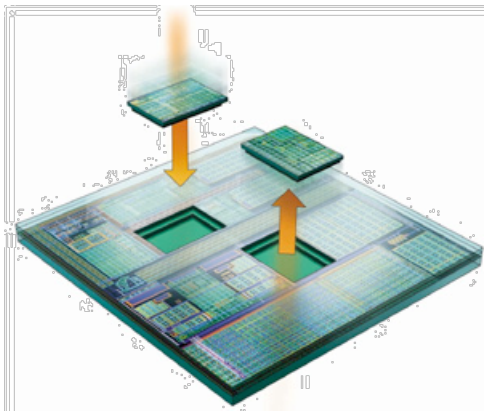




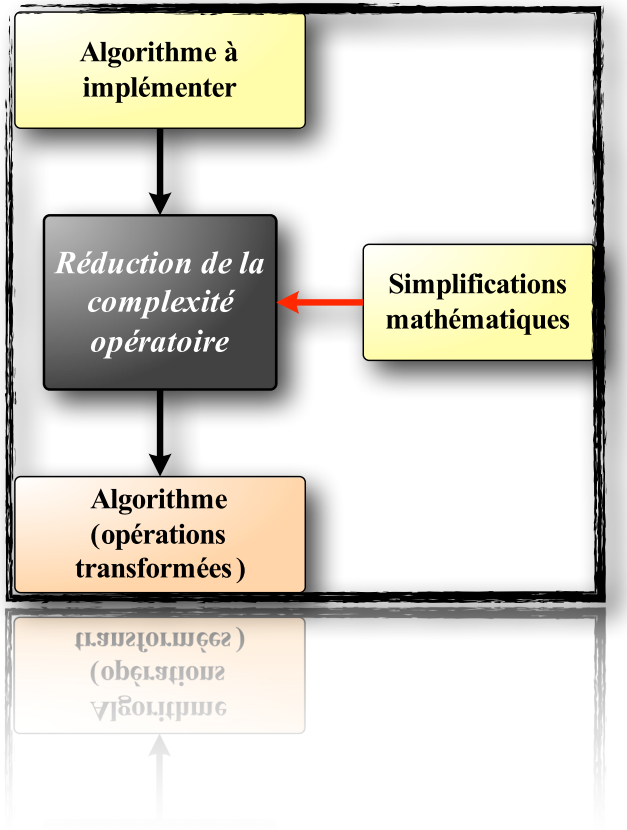
Méthodologie valable dans le cadre du développement de systèmes logiciels et matériels

Les différentes pistes de progrès envisagées

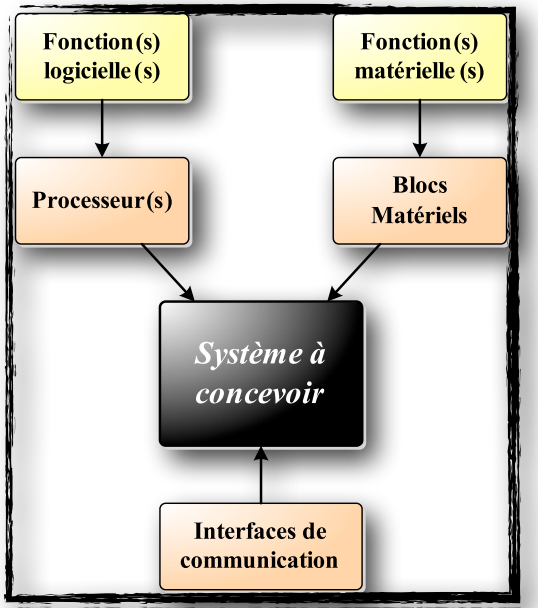


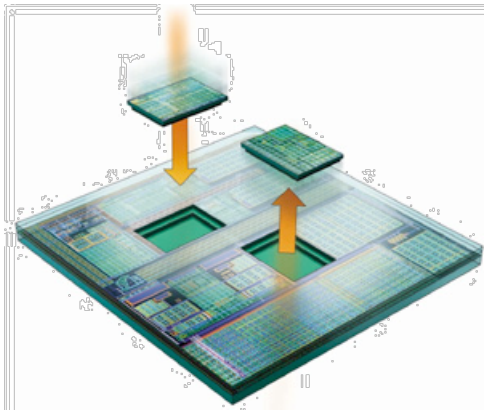


Méthodologie valable dans le cadre du développement de systèmes logiciels et matériels

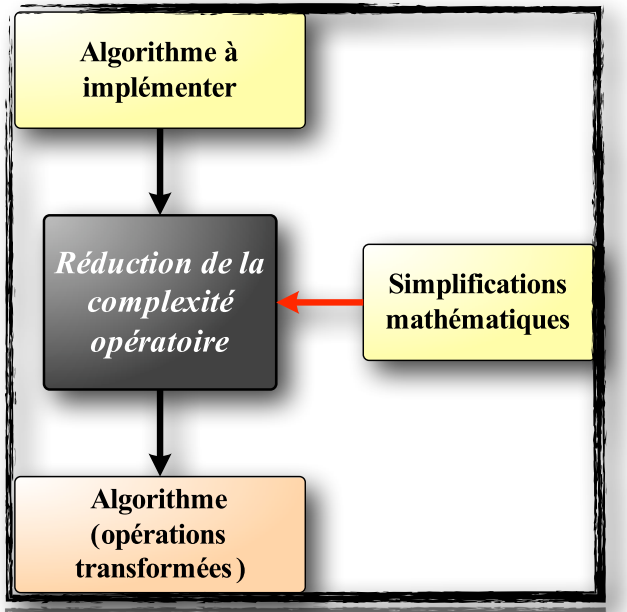


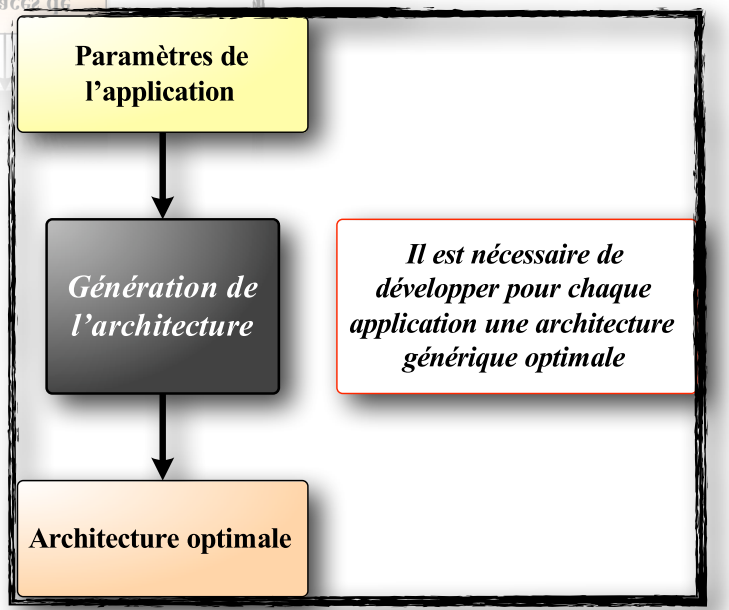
Les différentes pistes de progrès envisagées



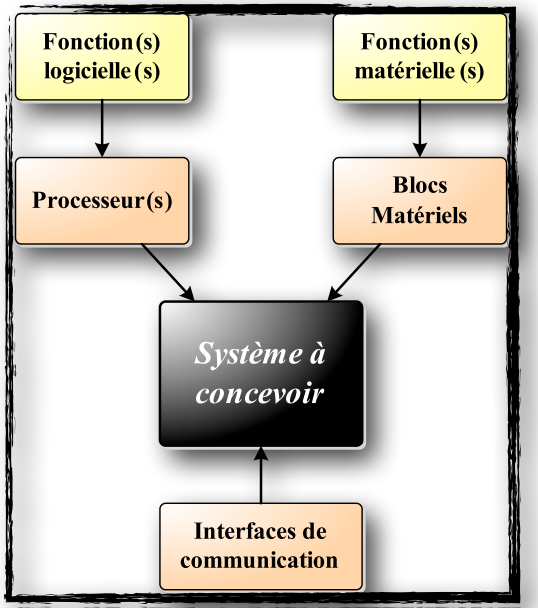


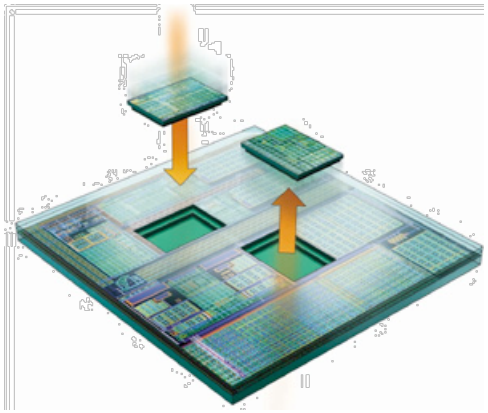
Méthodologie valable dans le cadre du développement de systèmes logiciels et matériels



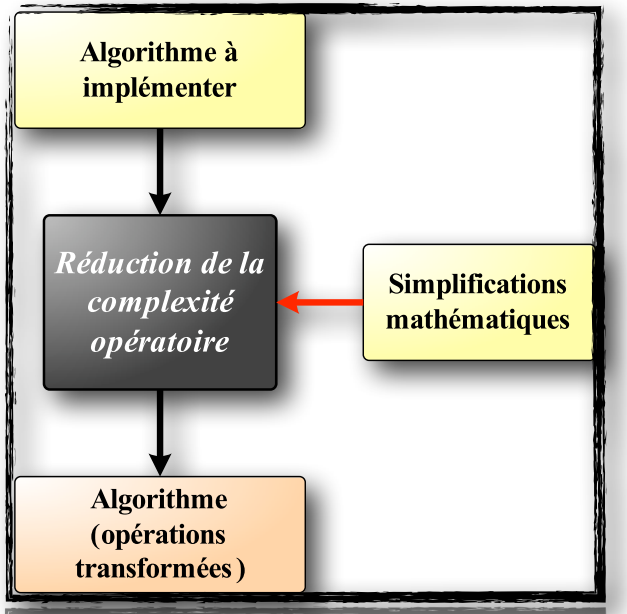


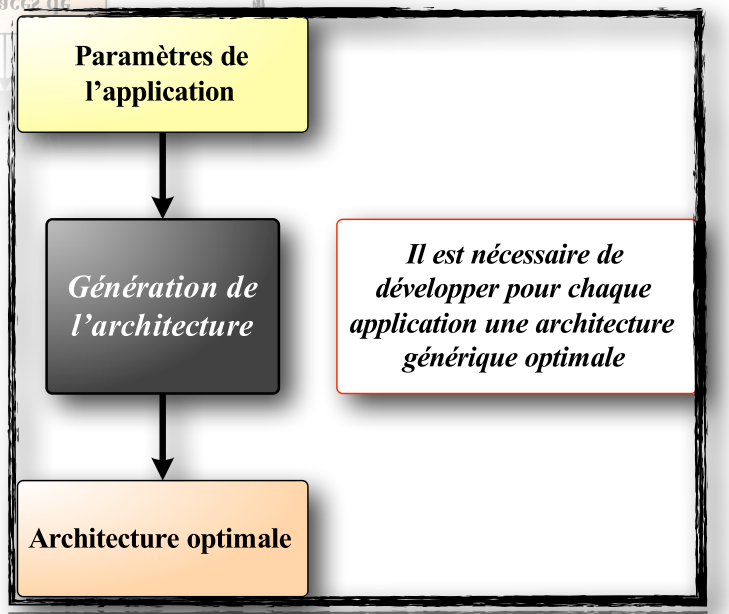
Les différentes pistes de progrès envisagées

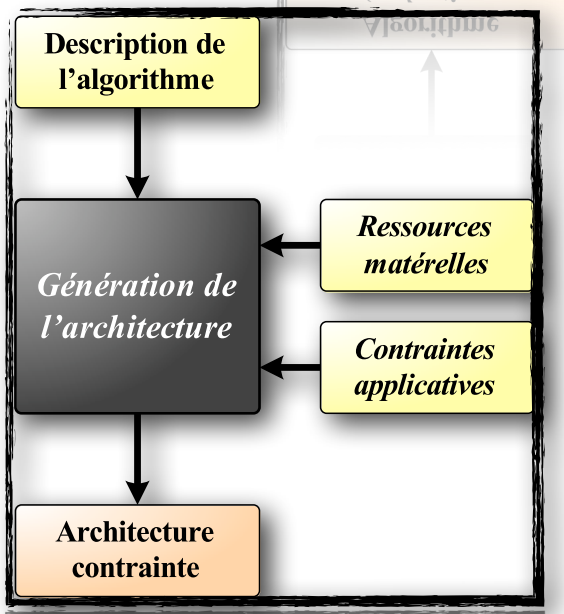




Méthodologie valable dans le cadre du développement de systèmes logiciels et matériels







Génération automatique d'architectures (HLS)

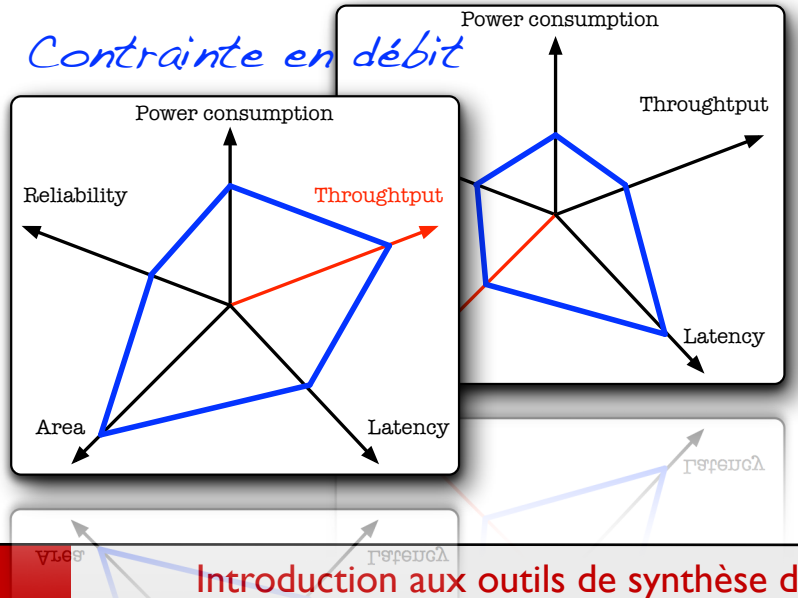
Spécification algorithmique

Génération automatique d'architectures (HLS)

Spécification algorithmique

Contrainte en surface

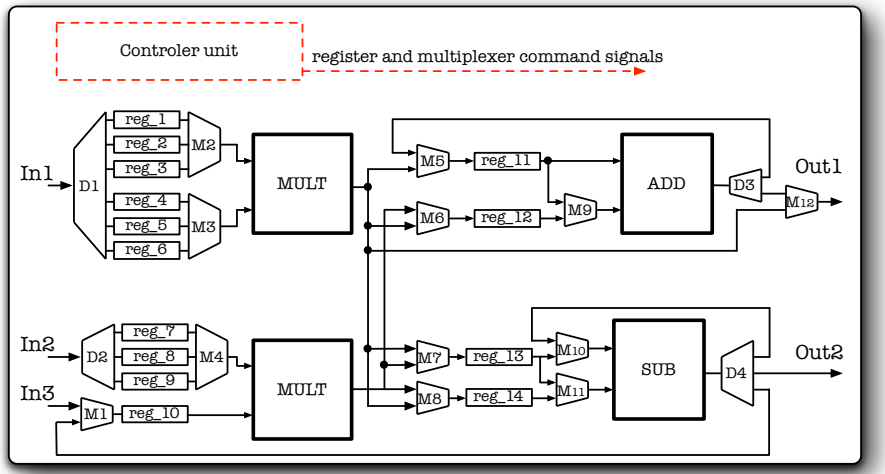
Contrainte en débit



Génération automatique d'architectures (HLS)

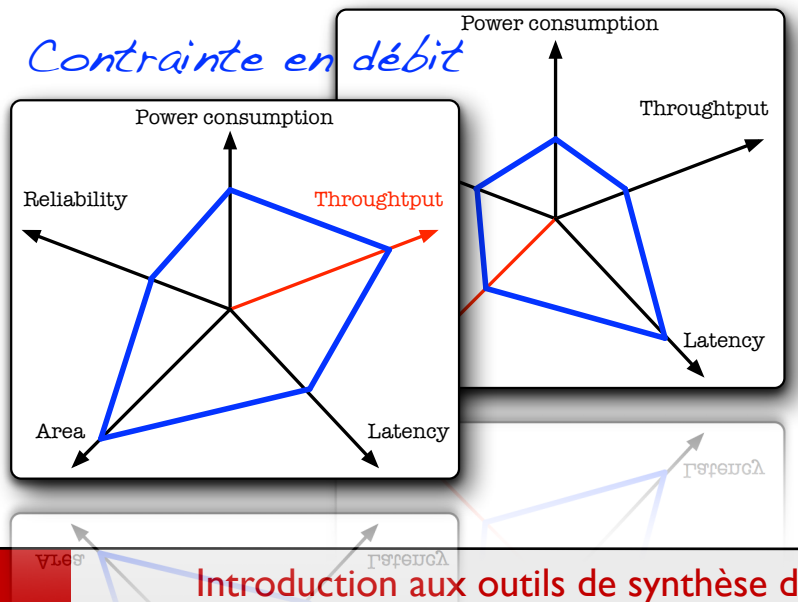
Spécification algorithmique

Architecture matérielle (VHDL)



Contrainte en surface

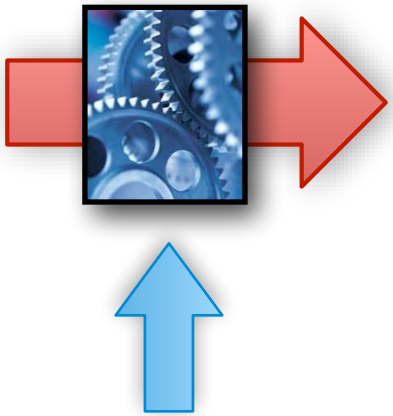
Contrainte en débit



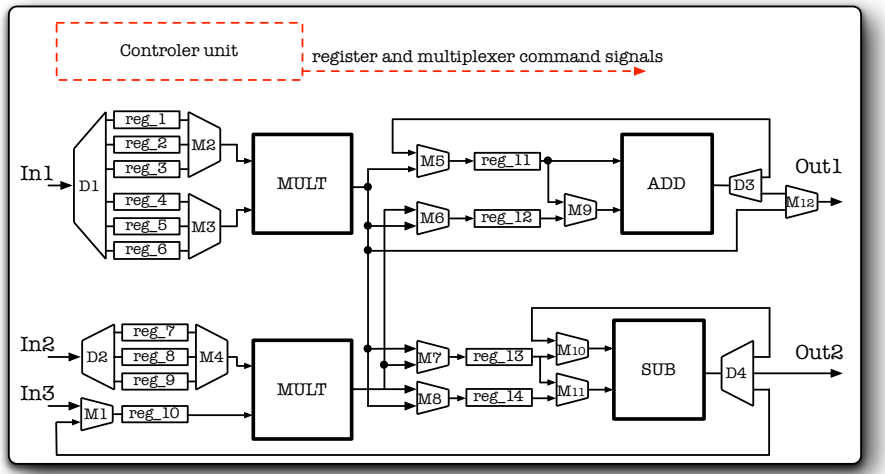
Génération automatique d'architectures (HLS)

Spécification algorithmique

Synthèse de Haut-Niveau

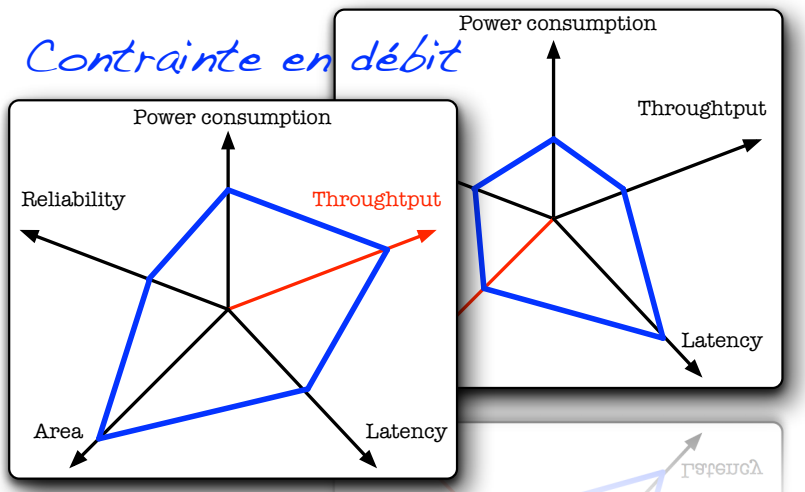


Architecture matérielle (VHDL)



Contrainte en surface

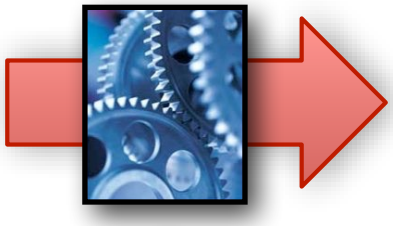
Contrainte en débit



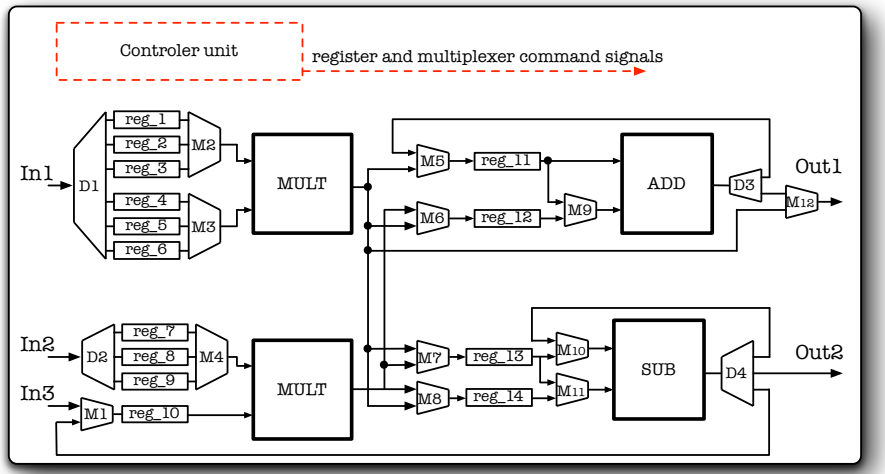
Génération automatique d'architectures (HLS)

Spécification algorithmique

Synthèse de Haut-Niveau



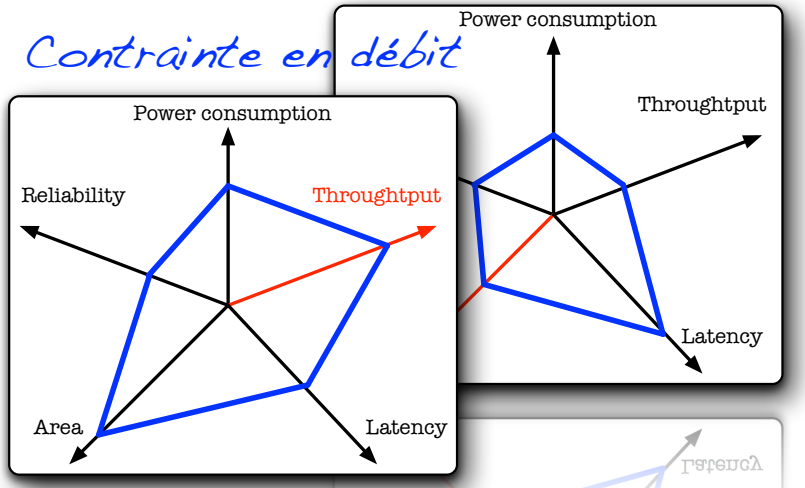
Architecture matérielle (VHDL)

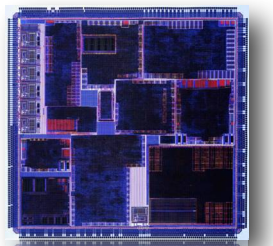


Synthèse logique

Contrainte en surface

Contrainte en débit



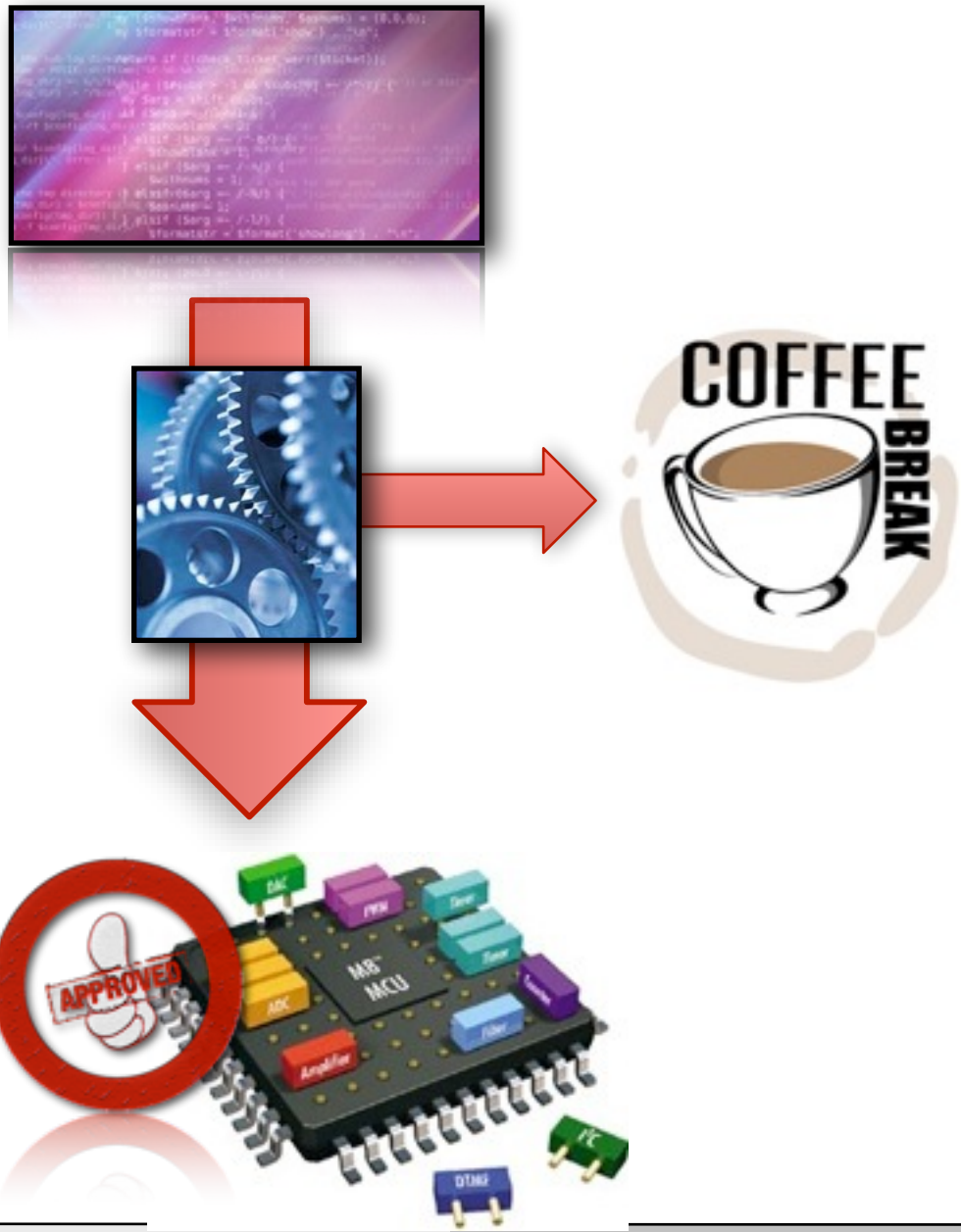


ASIC

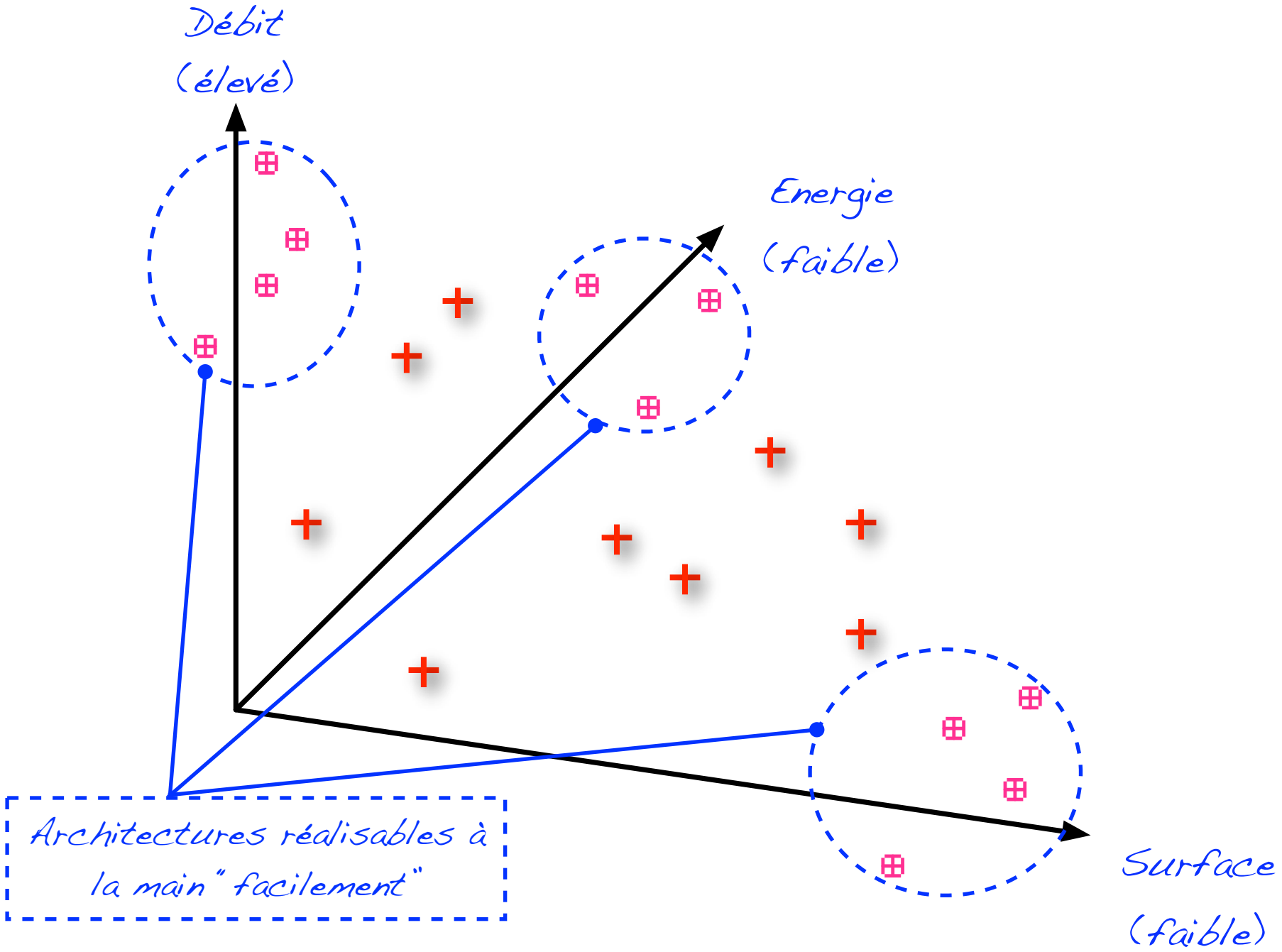
FPGA

L'automatisation des processus

- Avantages de l'automatisation,
  - Gain de temps (développement),
  - Gain de temps (vérification),
  - Exploration d'espaces de solutions difficilement réalisable manuellement,
  - Remonter le niveau d'abstraction => confier de nouvelles taches aux hommes,
- Inconvénients de l'automatisation,
  - Perte de la maîtrise technique des processus (confiance aveugle à l'outil),
  - «Réduction des performances» des composants ainsi générés (pas une généralité),



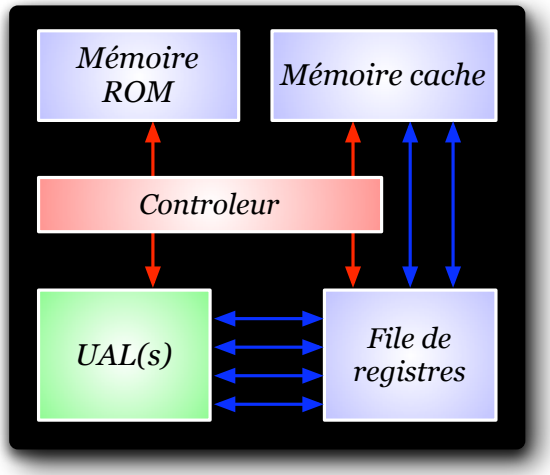
Exploration de l'espace des solutions

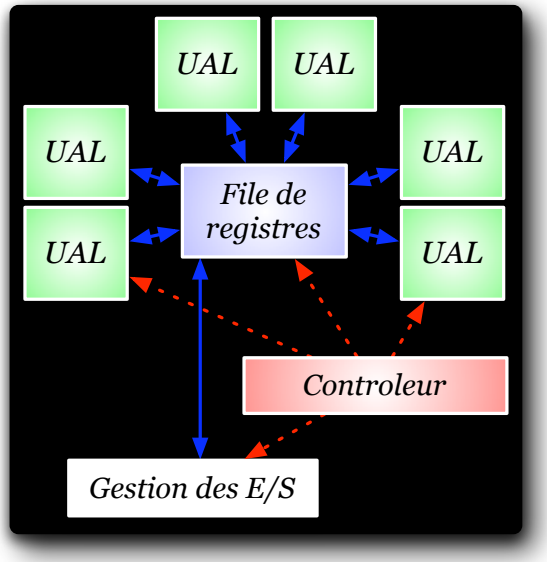


Pourquoi générer automatiquement des architectures ?

- Réduction du temps de conception du circuit,
  - FFT développée à la main = plusieurs jours / semaines,
  - A l'aide d'un outil, cela prend quelques secondes...
- Exploration de l'espace des solutions (meilleur compromis)
- Réduction de la taille des spécifications
  - Taille du code / 10 vis-a-vis du RTL => moins d'erreurs,
  - Codage du circuit plus naturel (lien plus fort avec l'algorithme),
  - Simulation plus rapide qu'à bas niveau d'abstraction,
- Technique indépendante de la technologie
  - Circuit portable => ASIC / FPGA,
- Prise en considération des contraintes à plus haut-niveau
  - Placement routage,
  - Consommation d'énergie,
  - Performances temporelles, etc.

Différentes architectures de circuits



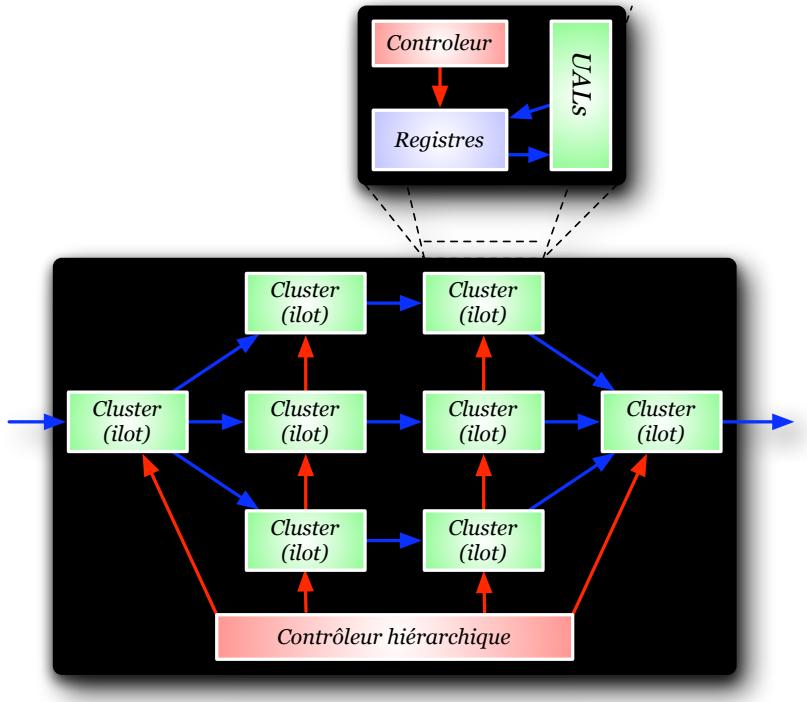


Différentes "classes" d'architectures d'implantation peuvent être mises en oeuvre :

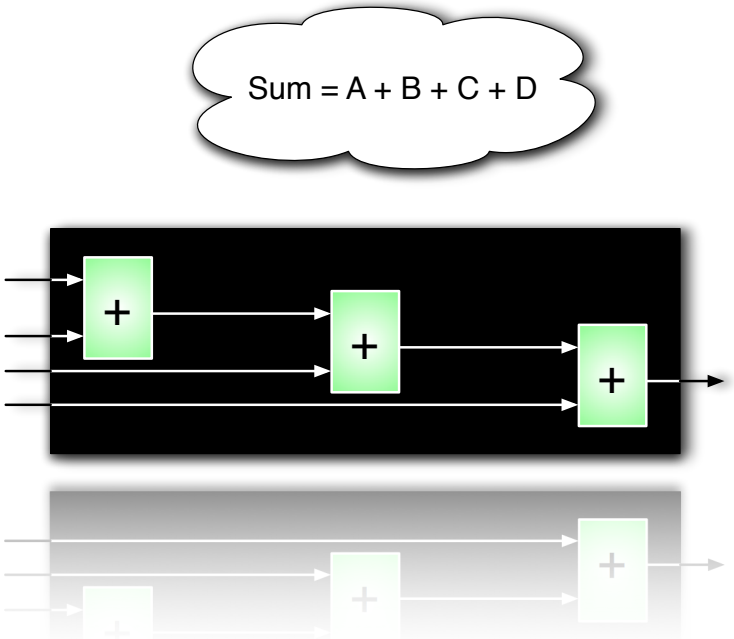
=> Processeurs généraliste,

=> Custom DSP architecture,

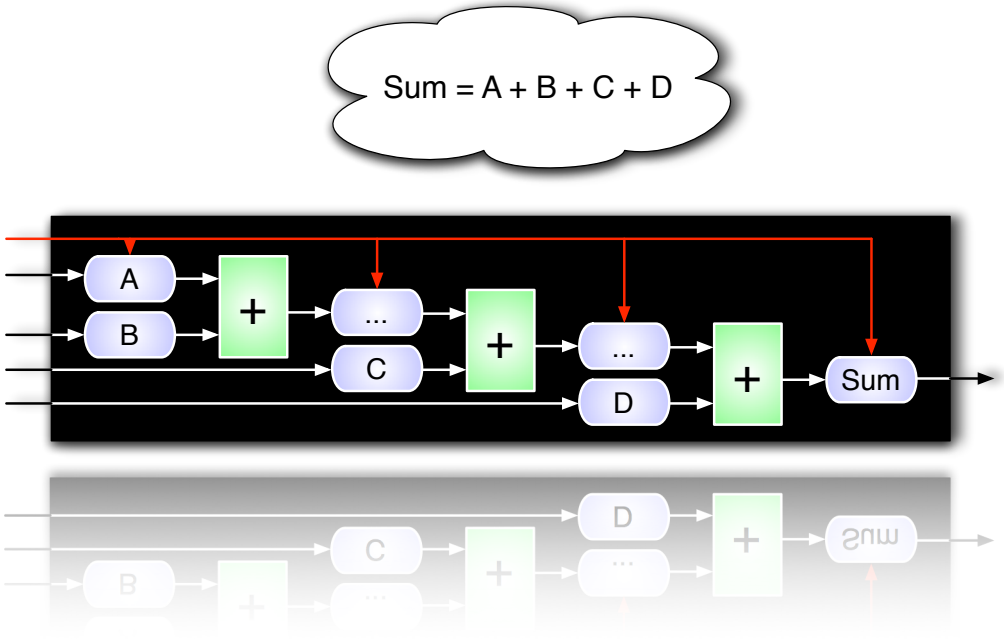
=> Systollic architectures,



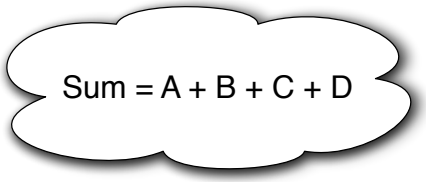
Différentes architectures de circuits



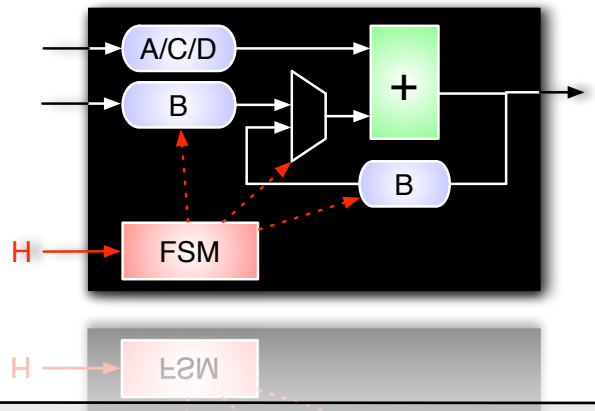
Architecture combinatoire
+ coût moyen,
+ latence moyenne,
- débit faible,



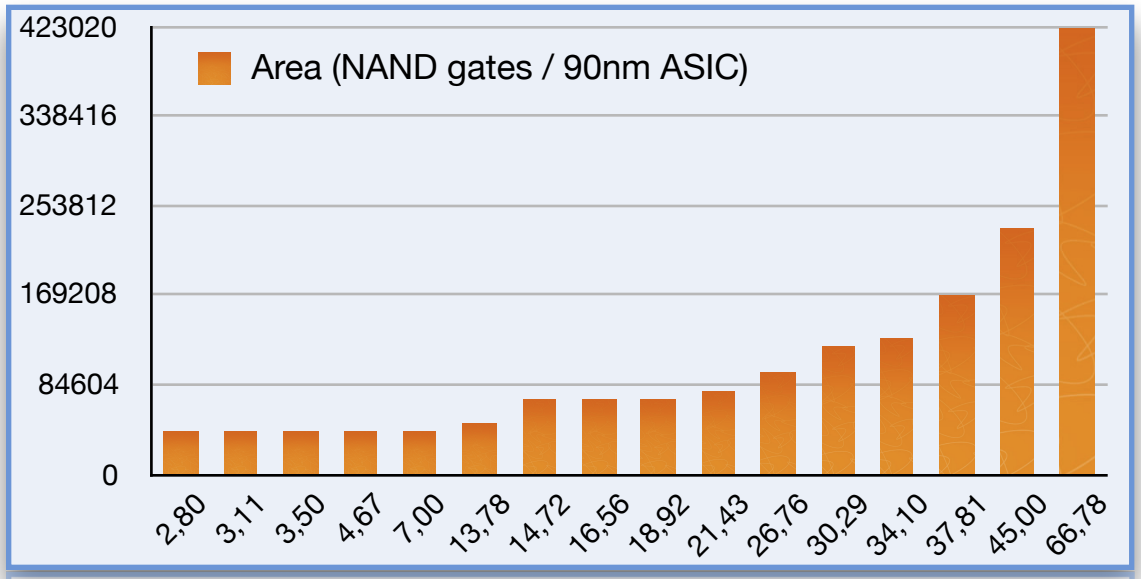
Architecture pipeline
- coût élevé,
+ latence faible,
+ débit élevé,



Architecture séquentielle
+ coût faible,
- latence élevée,
- débit faible/moyen,



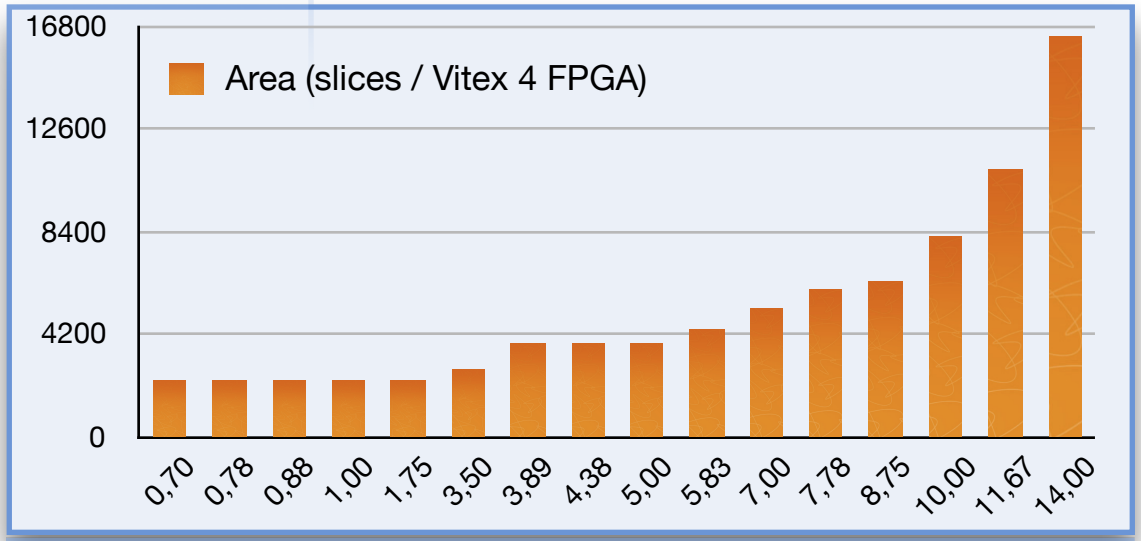
Résultat d'implantation sous contrainte temporelle



Troughtput constraint for the 16 taps floatting point FIR filter (MSamples / second)

Implantation d'un filtre FIR 64 points sous contrainte de latence sur cible ASIC

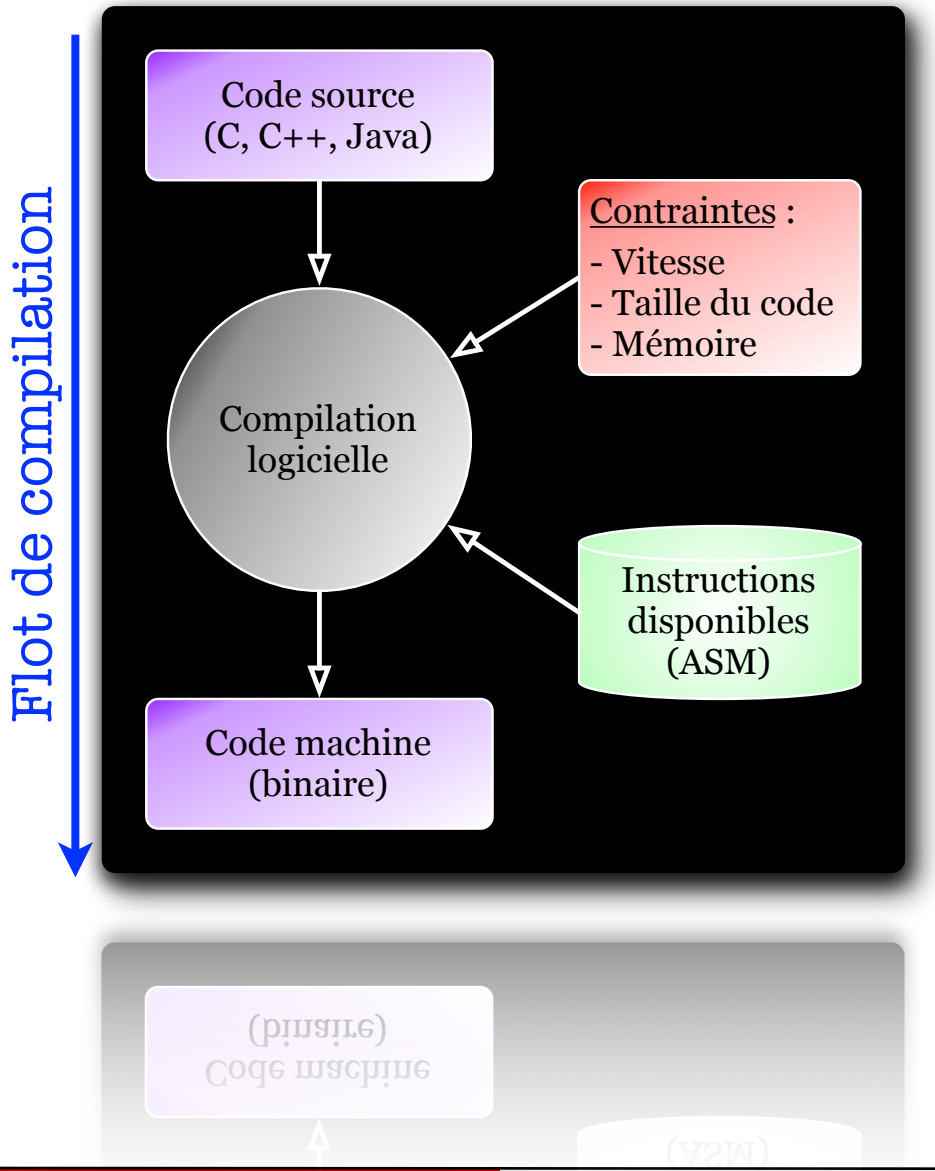
Implantation d'un filtre FIR 64 points sous contrainte de latence sur cible FPGA



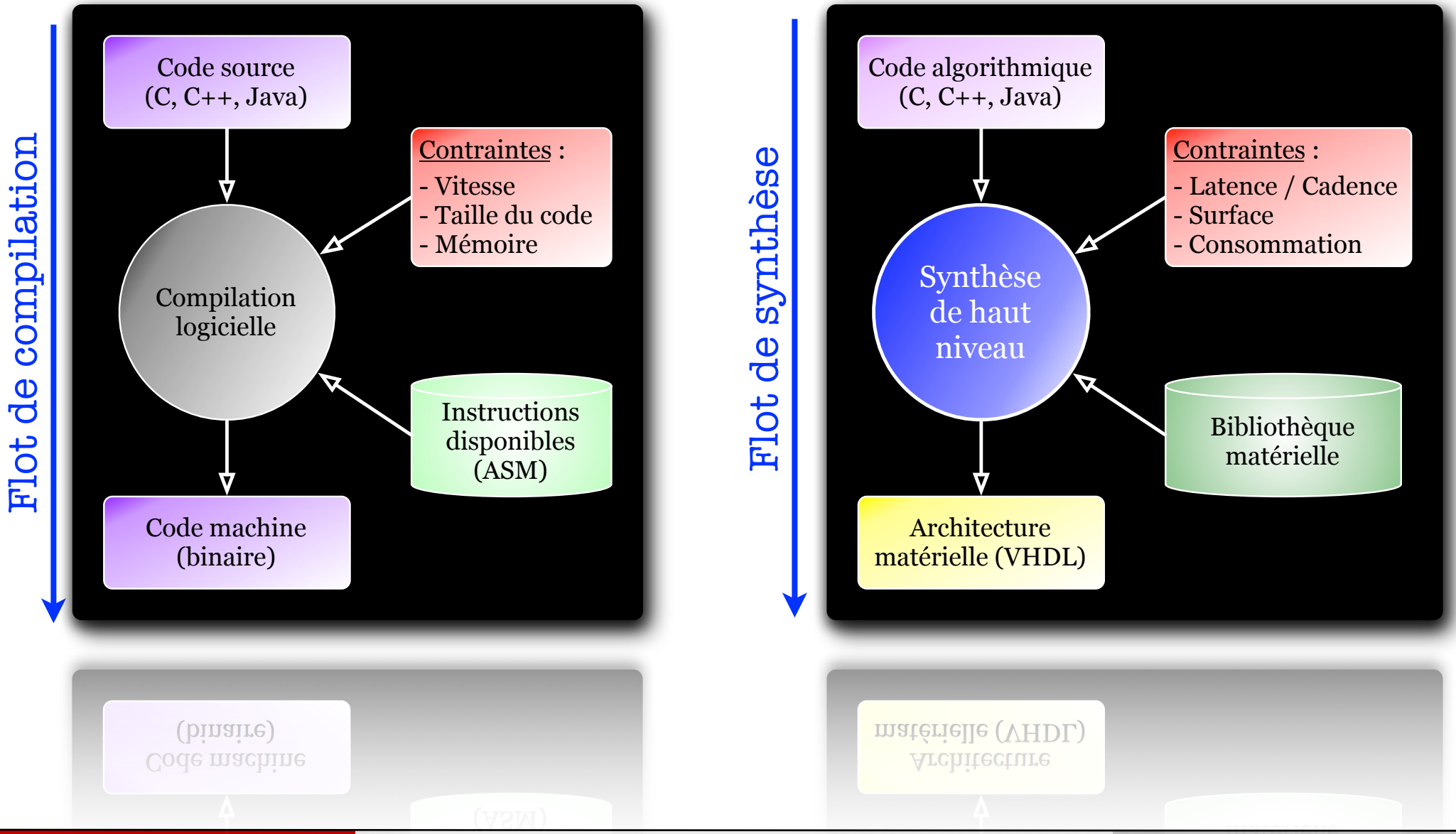
Troughtput constraint for the 16 taps floatting point FIR filter (MSamples / second)

Partie 2:
Présentation de la HLS

Analogie entre les flots de conception matériels et logiciels

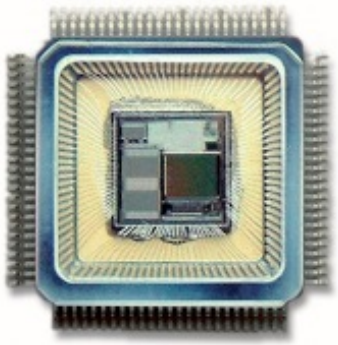


Analogie entre les flots de conception matériels et logiciels

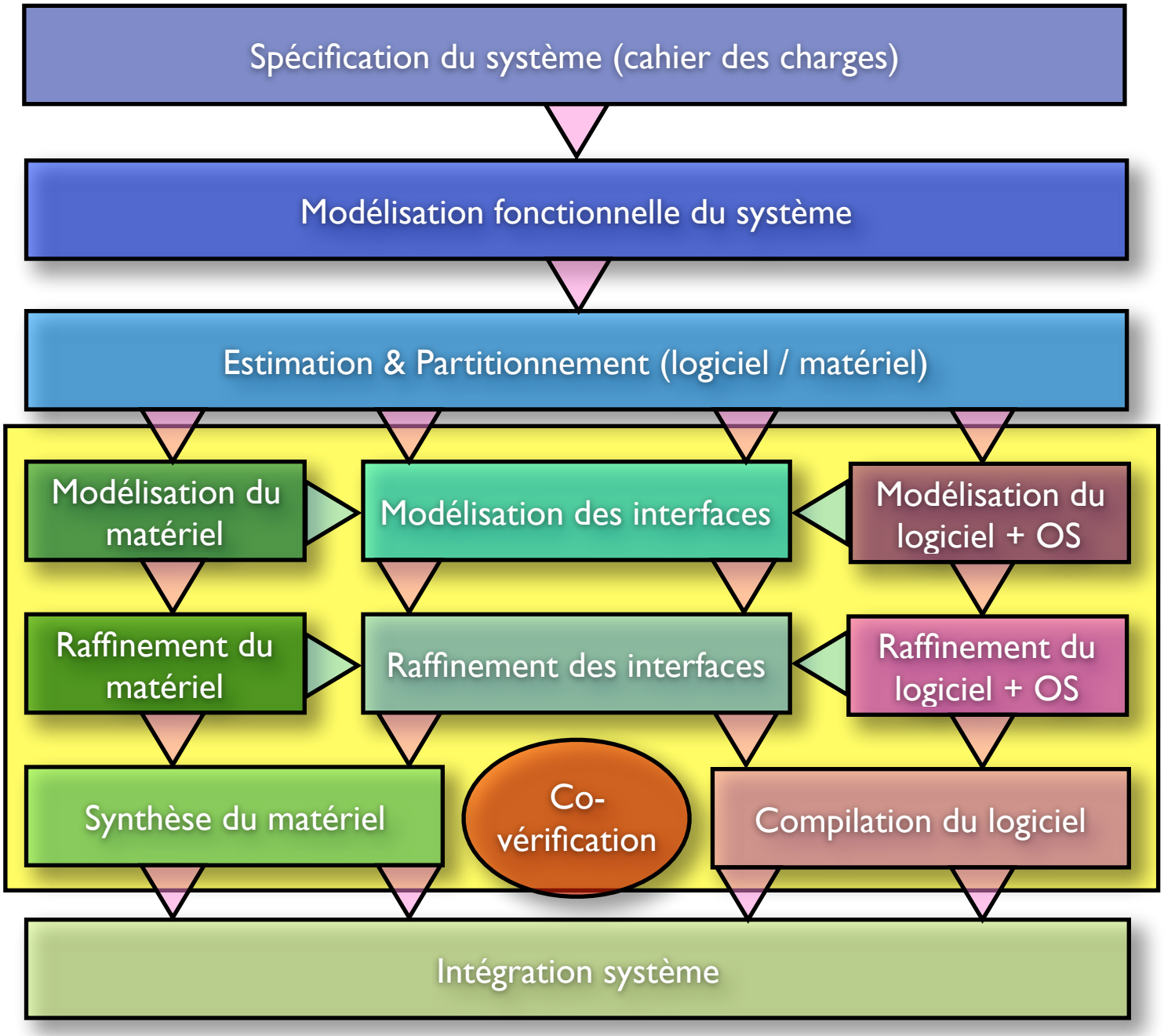


Flot de développement de niveau système

Cahier des charges

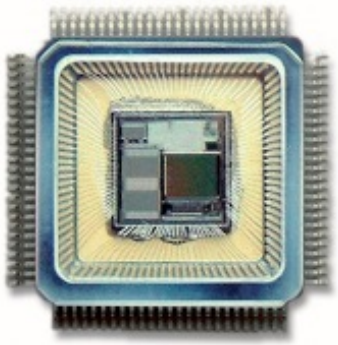


Circuit hétérogène

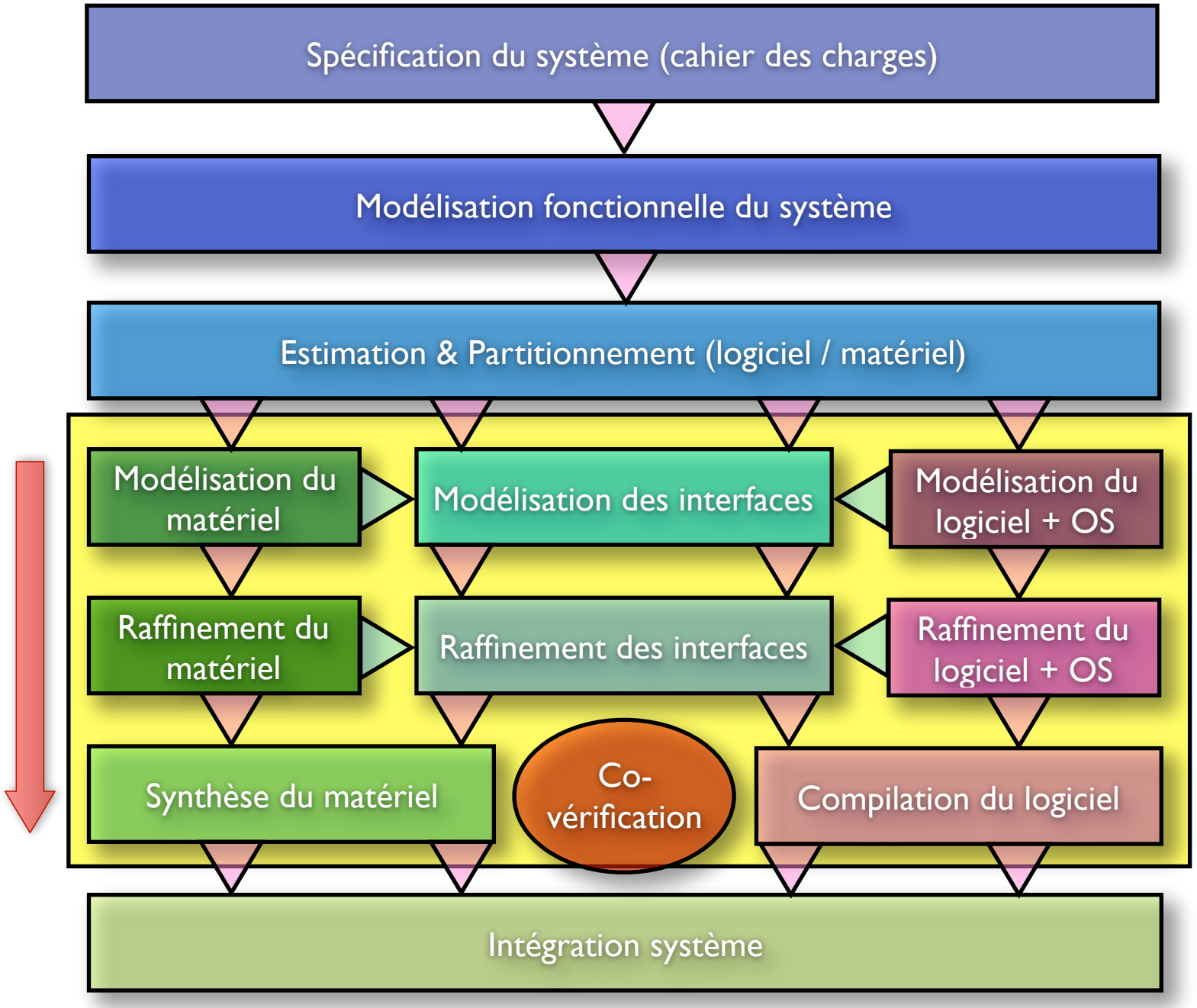


Flot de développement de niveau système

Cahier des charges

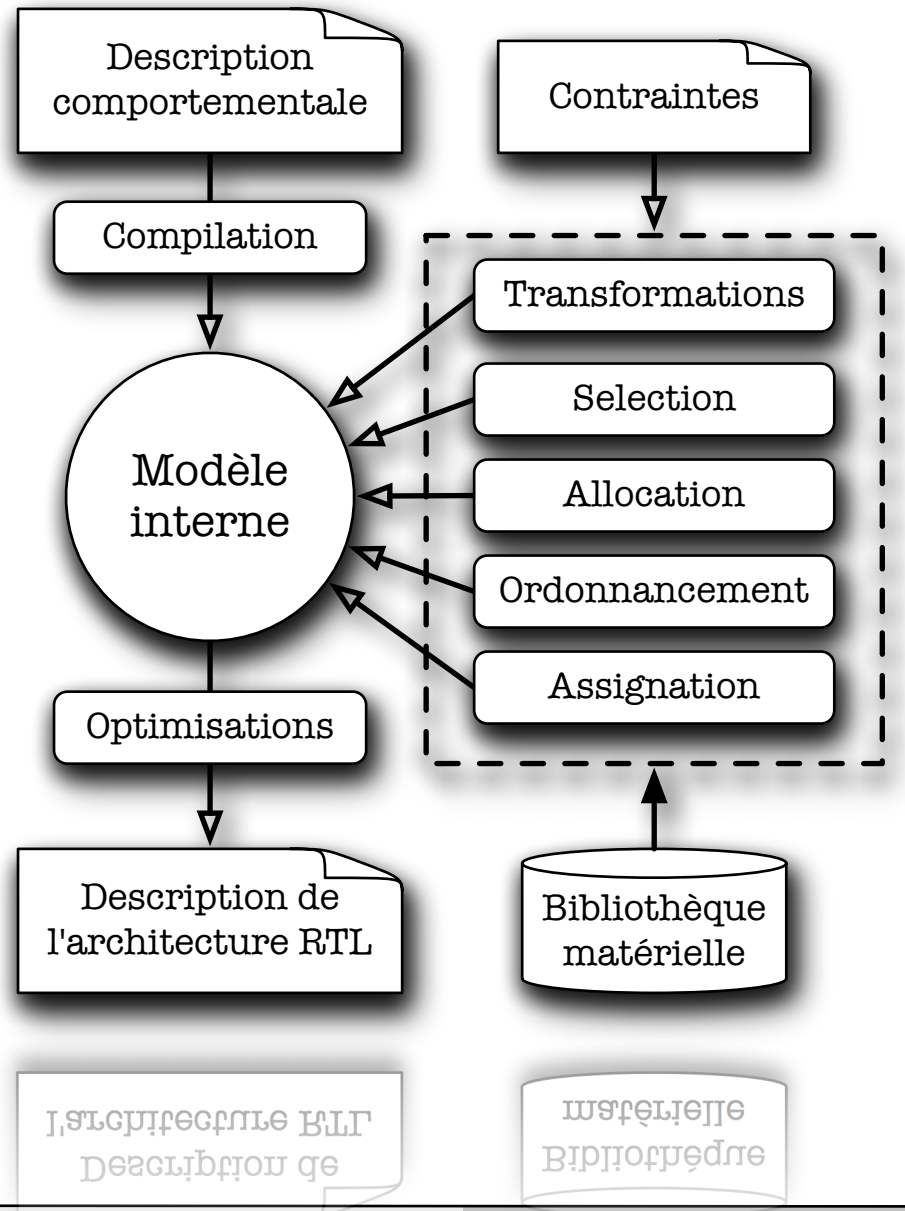


Circuit hétérogène

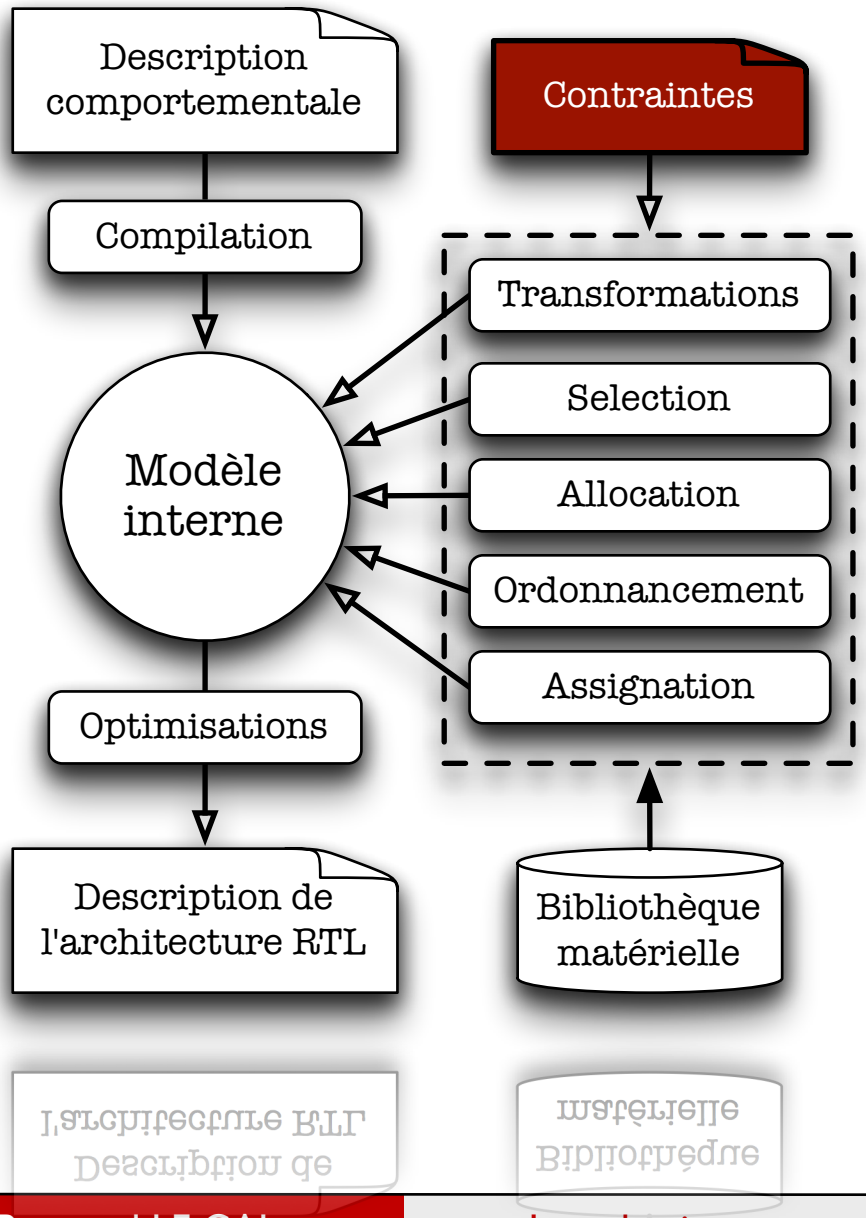


Le flot de synthèse de haut niveau

- Différentes étapes composant un flot de synthèse de haut niveau :
  - Compilation et optimisation de la description algorithmique,
  - Sélection des opérateurs à mettre en oeuvre,
  - Allocation des opérateurs,
  - Ordonnancement des opérations dans le temps,
  - Assignation des opérations sur les ressources matérielles allouées,
  - Génération de l'architecture du circuit (UT + contrôleur + ...).



Les points d'entrée algorithmiques

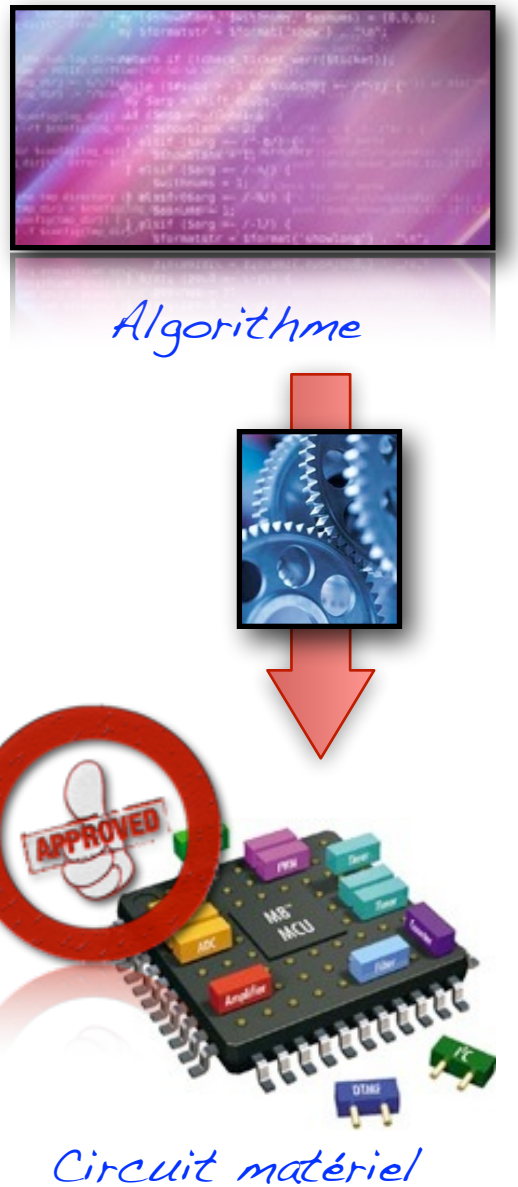


Dans le domaine de la compilation on contraint l'outil afin de produire du code «rapide» ou «peu consommateur en mémoire»

=> Quels sont les degrés de liberté que l'on possède ?

=> Quels sont les contraintes liées aux applications actuelles ?

Les différentes contraintes d'implémentation



Contraintes temporelles

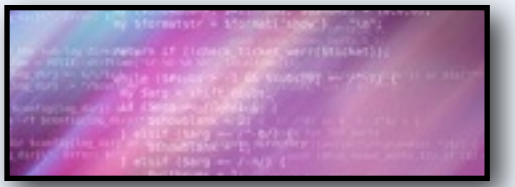
- Les contraintes temporelles peuvent s'exprimer de 2 manières différentes en fonction de l'application que l'on doit intégrer,
  - Une contrainte de latence permet d'indiquer le temps maximum de traitement des données (ex : 100ms).
  - Un contrainte de cadence permet de spécifier quel débit de données doit supporter l'architecture sans durée maximum du temps de traitement (ex : 24 fps dans les applications vidéo),

Contrainte spatiale

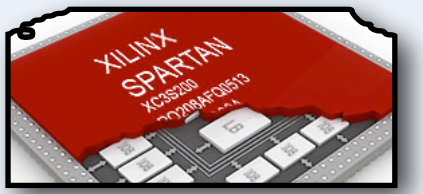
- La contrainte spatiale nommée aussi contrainte de surface permet de spécifier la surface maximum disponible pour implémenter l'architecture du circuit (cm2, LUTs, etc...),
- Cela permet de limiter le coût ($) du circuit à concevoir,

Différents objectifs pour la synthèse d'architectures

Concepteur



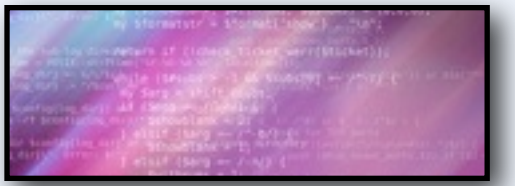
Application



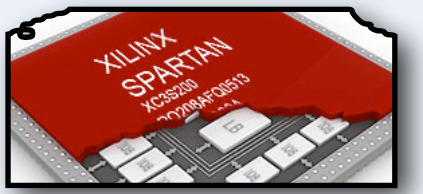
Technologie cible

Différents objectifs pour la synthèse d'architectures

Concepteur

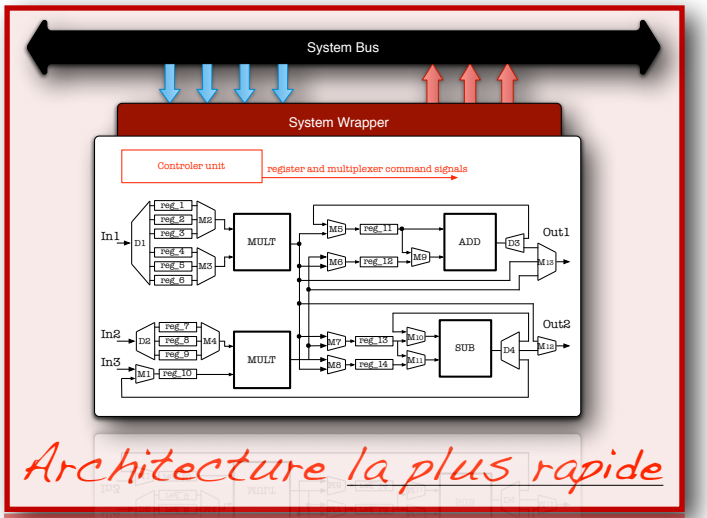


Application



Technologie cible

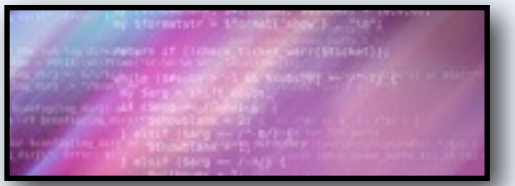
La contrainte matérielle (nombre d'opérateurs, surface...)



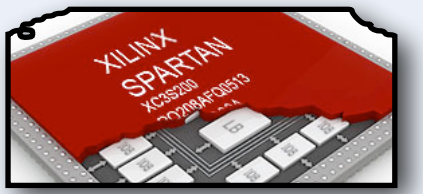
Architecture la plus rapide

Différents objectifs pour la synthèse d'architectures

Concepteur

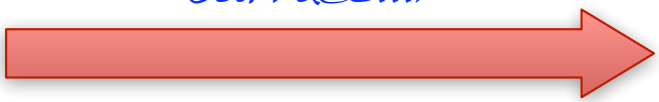


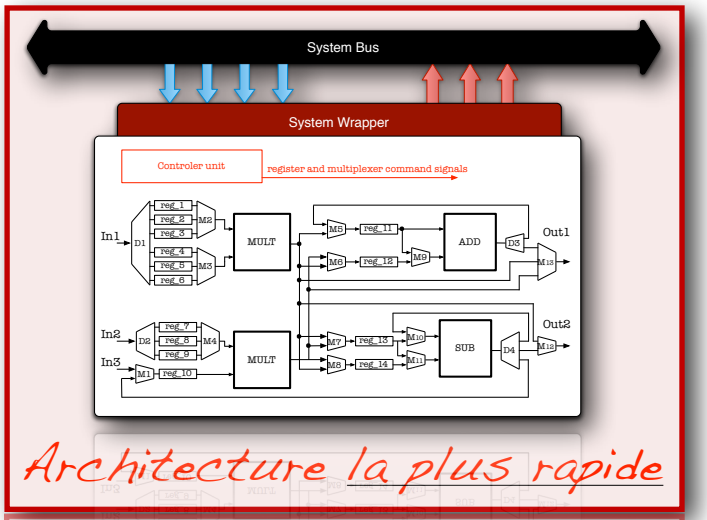
Application



Technologie cible

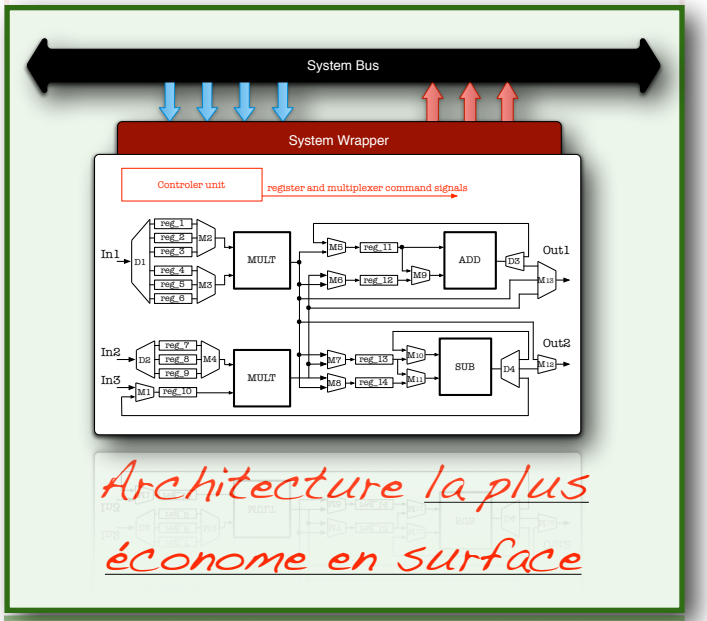
La contrainte matérielle (nombre d'opérateurs, surface...)





Architecture la plus rapide

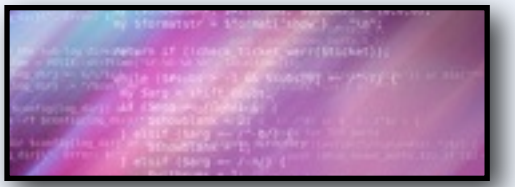
La contrainte temporelle (nombre maximum de cycles d'horloge)



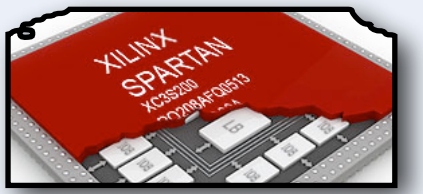
Architecture la plus économe en surface

Différents objectifs pour la synthèse d'architectures

Concepteur

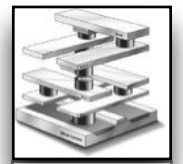


Application

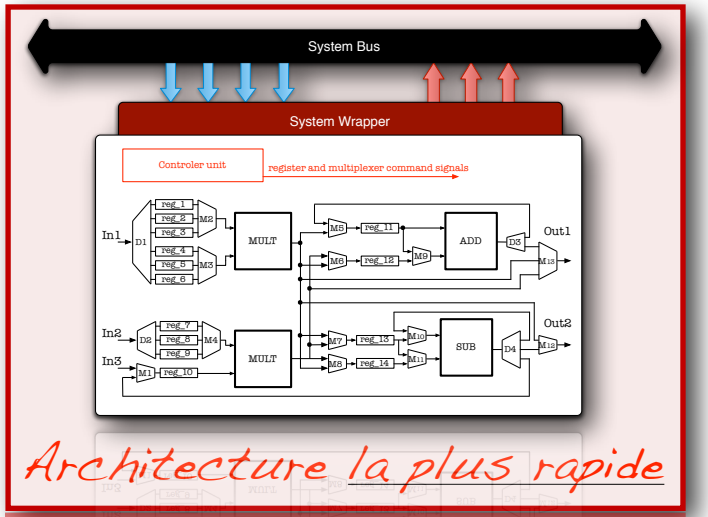


Technologie cible

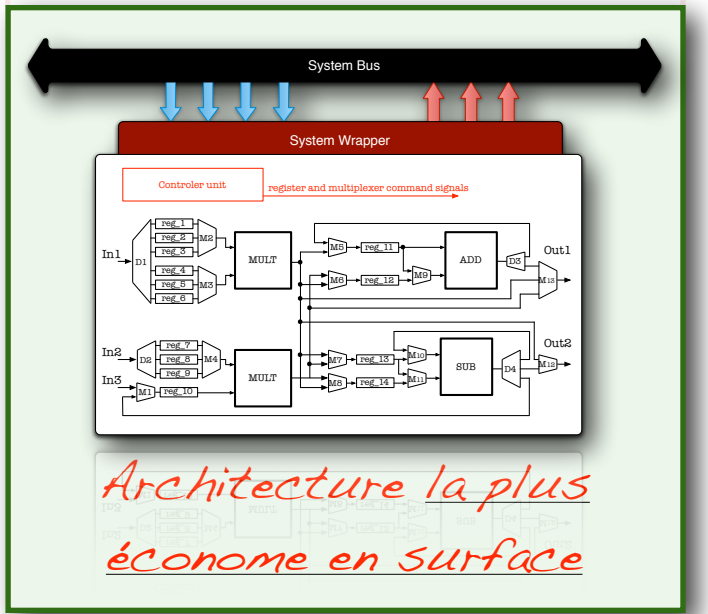
La contrainte matérielle (nombre d'opérateurs, surface...)



La contrainte temporelle (nombre maximum de cycles d'horloge)

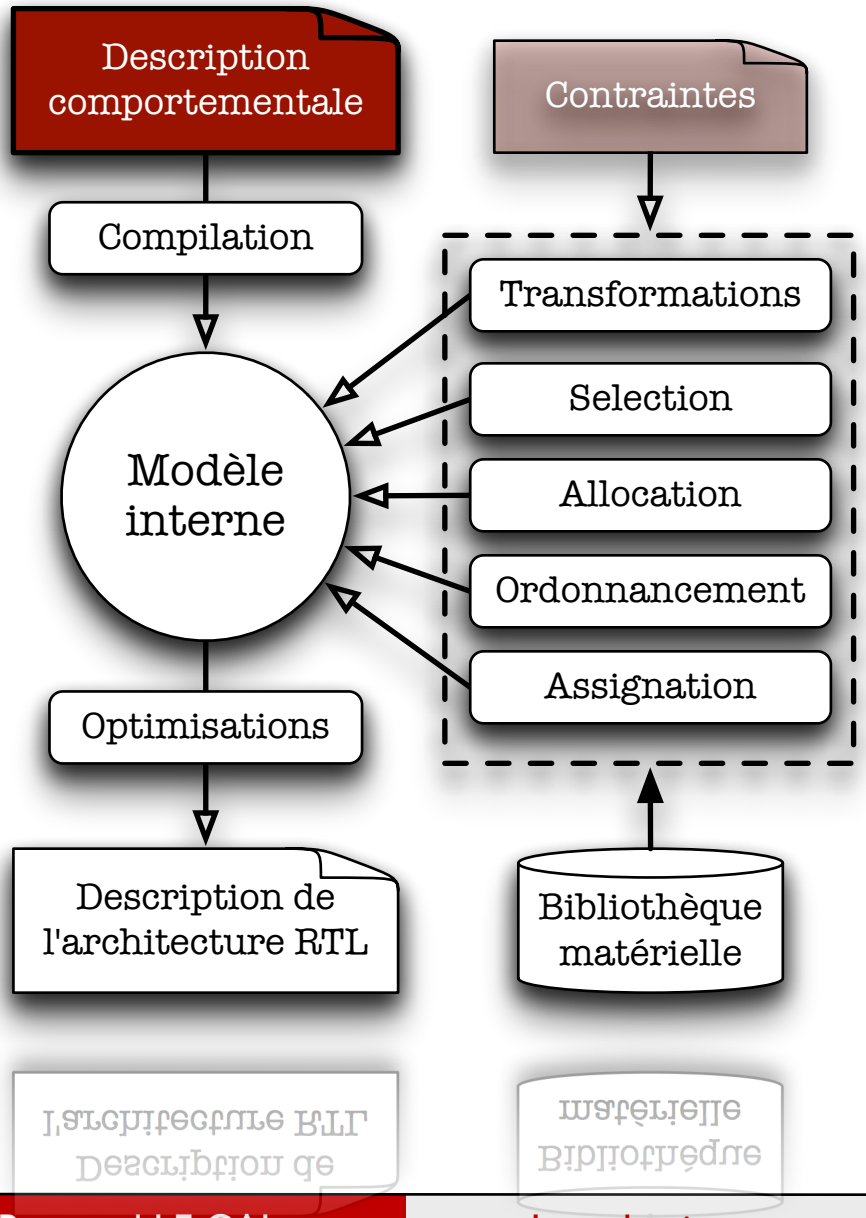


Architecture la plus rapide



Architecture la plus économe en surface

Les points d'entrée algorithmiques



Les algorithmes développés par le concepteur doivent être exprimés de manière a être compris sans ambiguité

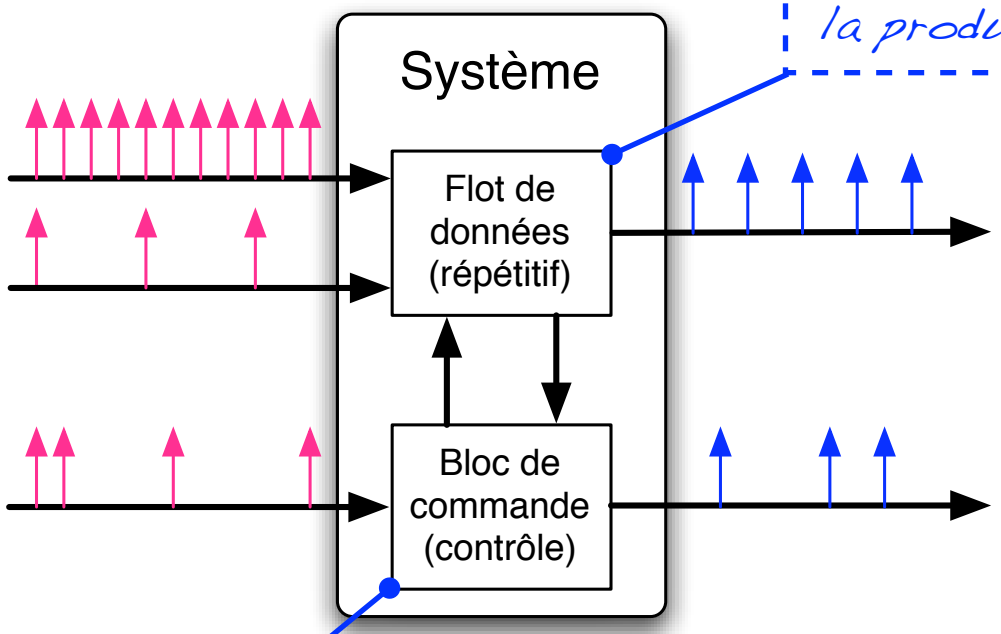
=> Quelle est la nature de mon algorithme ?

=> Quels sont les langages adaptés à sa description ?

=> Comment exprimer sans ambiguité le comportement attendu ?

Composition d'un système actuel

On ne décrit pas toutes les applications de la même manière car elles peuvent posséder des *caractéristiques intrinsèques différentes*.



Les algorithmes type flot de données sont réguliers dans la consommation et la production des données

Les applications de contrôle sont généralement irrégulières dans la réception des événements et la génération des commandes associées

La description algorithmique d'une application doit permettre d'exprimer sans ambiguïté le comportement attendu

Les points d'entrée algorithmiques

- Afin de spécifier le comportement attendu d'une application il est nécessaire de l'exprimer au travers d'un langage,
  - Langage C pour les informaticiens,
  - Langage MatLab pour les traiteurs du signal,
  - Langage VHDL/SystemC pour les électroniciens,
- Un langage permet de spécifier formellement les traitements que doit réaliser le système,
  - Traitement numérique (un produit de matrices),
  - Contrôle commande (gestion d'un feu rouge),
  - Systèmes mixtes (compression MPEG),

Exemples de codes sources algorithmiques

Ces codes sources sont dis algorithmiques car ils incorporent aucune relation avec le matériel (temps, dynamique, ressources, etc.)

```
int SommeMatrice(int matrice[64,64]){
    int sum = 0;
    for(int y=0; y<64; y++)
        for(int x=0; x<64; x++)
            sum += matrice[y, x];
    return sum;
}
```

Combien de ressources met on en oeuvre pour réaliser ce calcul ?

```
int PGDC(int a, int b){
    while( a != b ){
        if( a > b )
            a = a - b;
        else
            b = b - a;
    }
    return b;
}
```

Comment réalise t'on une boucle while dans un FPGA / ASIC ?

Description comportementale de la 2d-DCT 8x8

Editing (/Users/legal/Developpement/GraphLab/samples/QoS/DCT2d/dct2d.m)

File Edit

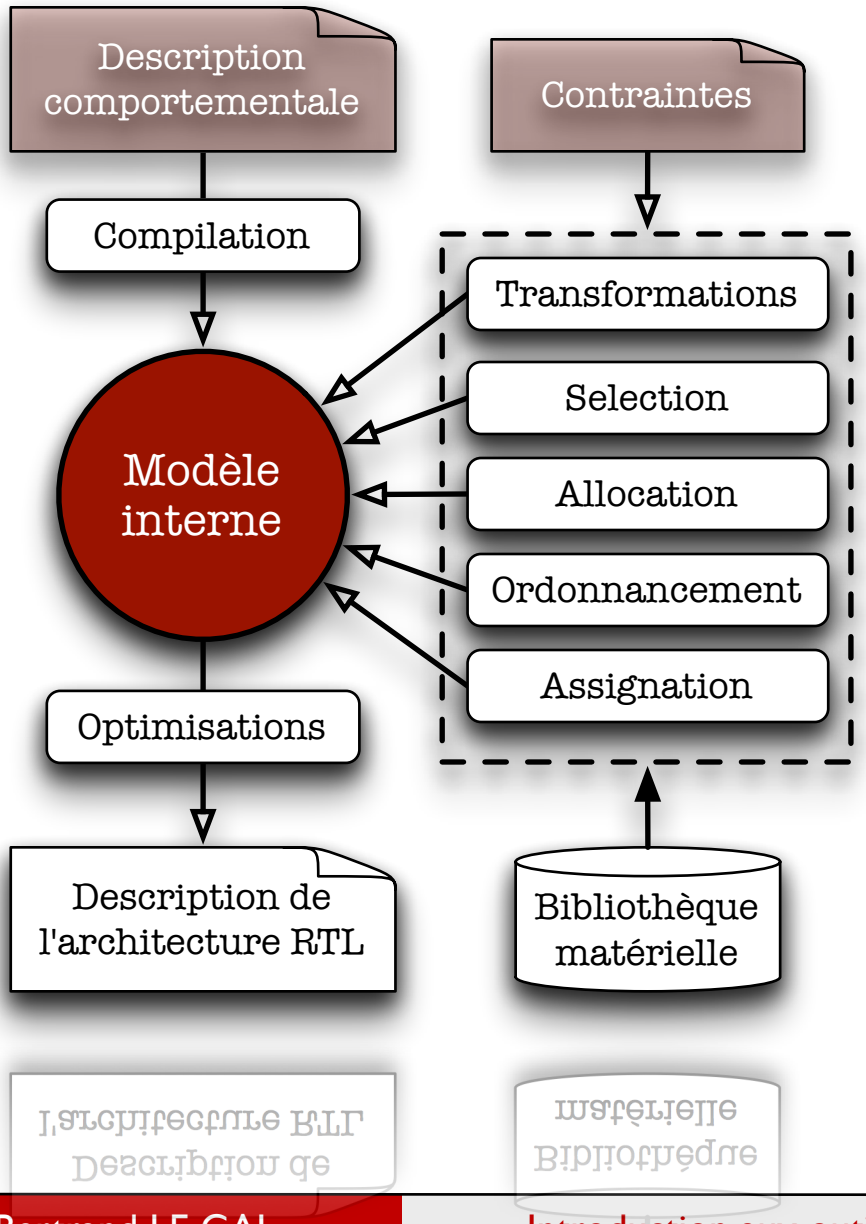
```
function [Y] = dct2d( X )

% CONSTANT DECLARATION
N=8;
pi = 3.141596;
dyn = 12; % bits per coefficient

% XIT OF THE COEFFICIENT ARRAY
ref = sqrt(2/N);
for i=1:N - 1
   C(i)= Qn(ref  *  cos(i * pi/16), dyn);
end

% THE COMPUTATION CORE
for y =1:8
  T(y,0 + 1) = C(4) * (X(y,1) + X(y,2)  +        (X(y,3) + X(y,4)) +        (X(y,5) + X(y,6)) +        (X(y,7) + X(y,8)));
  T(y,2 + 1) = C(2) * (X(y,1) + X(y,8)) + C(6) * (X(y,2) + X(y,7)) - C(6) * (X(y,3) + X(y,6)) - C(2) * (X(y,4) + X(y,5));
  T(y,4 + 1) = C(4) * (X(y,1) + X(y,8)) - C(4) * (X(y,2) + X(y,7)) - C(4) * (X(y,3) + X(y,6)) + C(4) * (X(y,4) + X(y,5));
  T(y,6 + 1) = C(6) * (X(y,1) + X(y,8)) - C(2) * (X(y,2) + X(y,7)) + C(2) * (X(y,3) + X(y,6)) - C(6) * (X(y,4) + X(y,5));

  T(y,1 + 1) = C(1) * (X(y,1) - X(y,8)) + C(3) * (X(y,2) - X(y,7)) + C(5) * (X(y,3) - X(y,6)) + C(7) * (X(y,4) - X(y,5));
  T(y,3 + 1) = C(3) * (X(y,1) - X(y,8)) - C(7) * (X(y,2) - X(y,7)) - C(1) * (X(y,3) - X(y,6)) - C(5) * (X(y,4) - X(y,5));
  T(y,5 + 1) = C(5) * (X(y,1) - X(y,8)) - C(1) * (X(y,2) - X(y,7)) + C(7) * (X(y,3) - X(y,6)) + C(3) * (X(y,4) - X(y,5));
  T(y,7 + 1) = C(7) * (X(y,1) - X(y,8)) - C(5) * (X(y,2) - X(y,7)) + C(3) * (X(y,3) - X(y,6)) - C(1) * (X(y,4) - X(y,5));
end

for x =1:8
    Y(0 + 1,x) = C(4) * (T(1,x) + T(2,x) + T(3,x) + T(4,x) + T(5,x) + T(6,x) + T(7,x) + T(8,x));
    Y(2 + 1,x) = C(2) * (T(1,x) + T(8,x)) + C(6) * (T(2,x) + T(7,x)) - C(6) * (T(3,x) + T(6,x)) - C(2) * (T(4,x) + T(5,x));
    Y(4 + 1,x) = C(4) * (T(1,x) + T(8,x)) - C(4) * (T(2,x) + T(7,x)) - C(4) * (T(3,x) + T(6,x)) + C(4) * (T(4,x) + T(5,x));
    Y(6 + 1,x) = C(6) * (T(1,x) + T(8,x)) - C(2) * (T(2,x) + T(7,x)) + C(2) * (T(3,x) + T(6,x)) - C(6) * (T(4,x) + T(5,x));

    Y(1 + 1,x) = C(1) * (T(1,x) - T(8,x)) + C(3) * (T(2,x) - T(7,x)) + C(5) * (T(3,x) - T(6,x)) + C(7) * (T(4,x) - T(5,x));
    Y(3 + 1,x) = C(3) * (T(1,x) - T(8,x)) - C(7) * (T(2,x) - T(7,x)) - C(1) * (T(3,x) - T(6,x)) - C(5) * (T(4,x) - T(5,x));
    Y(5 + 1,x) = C(5) * (T(1,x) - T(8,x)) - C(1) * (T(2,x) - T(7,x)) + C(7) * (T(3,x) - T(6,x)) + C(3) * (T(4,x) - T(5,x));
    Y(7 + 1,x) = C(7) * (T(1,x) - T(8,x)) - C(5) * (T(2,x) - T(7,x)) + C(3) * (T(3,x) - T(6,x)) - C(1) * (T(4,x) - T(5,x));
end

```

Text read from file

Nombre de bits générique

Calcul des coefficients

Les modèles de représentation formels



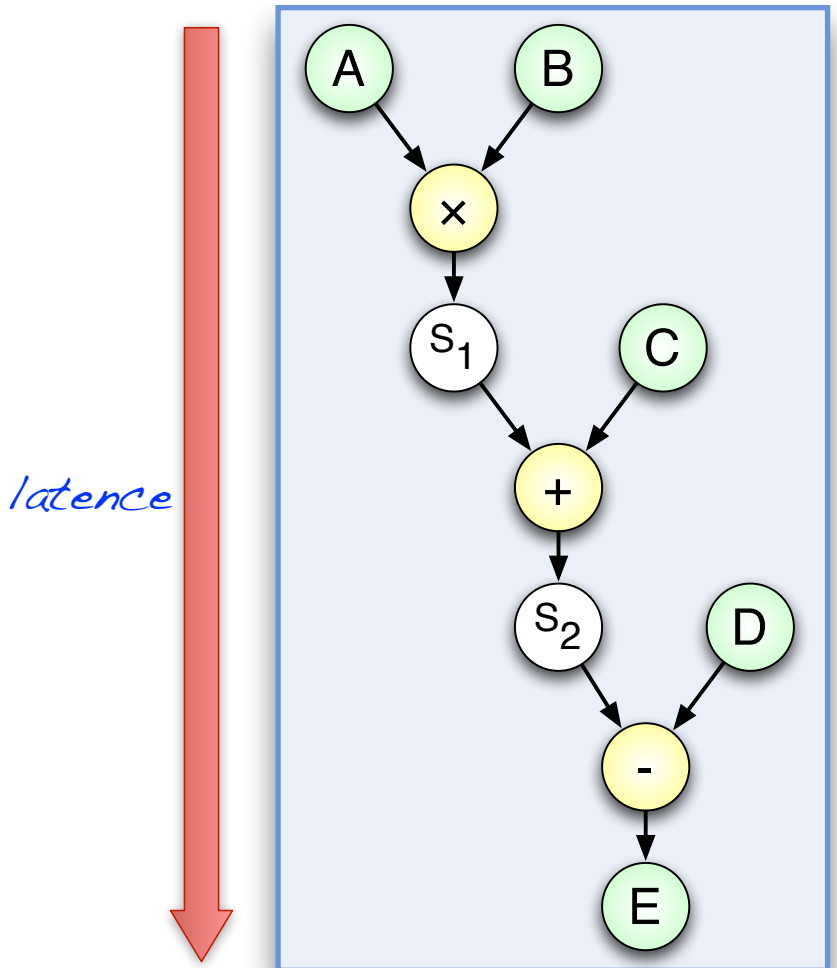
Les langages de descriptions sont compréhensibles par les concepteurs humains mais peu adaptés aux traitements réalisés par les outils

=> Quelles sont les sémantiques présentes dans la description comportementale ?

=> Quels sont les traitements à réaliser ?

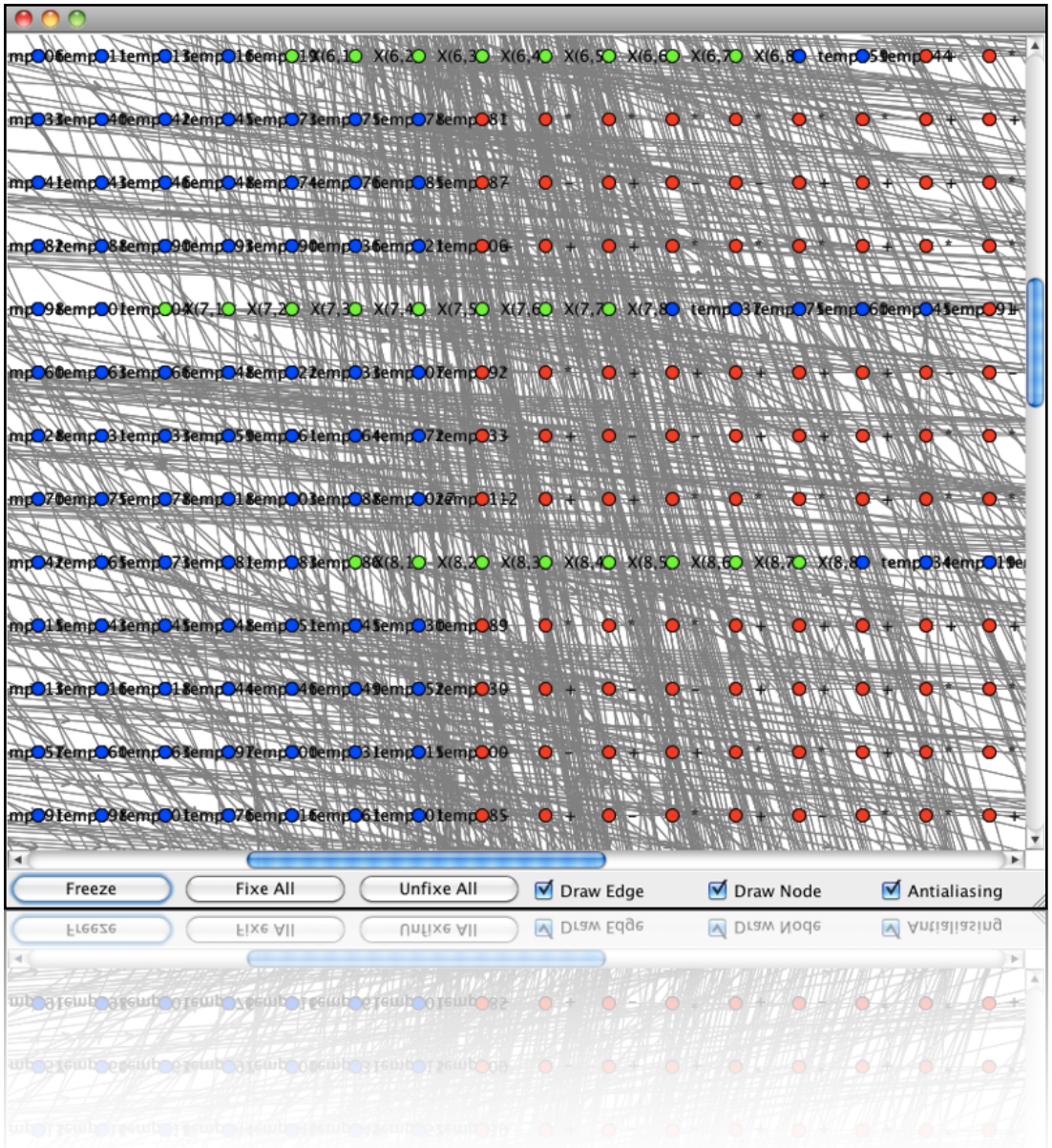
Modèle formel calcul - Exemple du «Data Flow Graph»

- Un graphe flot de données (DFG) est un graphe bi-partite orienté acyclique DFG(V, E) où l'ensemble fini des noeuds noté V = {v0 , ... , vn } représente les données et les opérations.
- L'ensemble noté E avec $E \subseteq V \times V$ est un ensemble d'arcs reliant les noeuds. Ces arcs représentent des dépendances de données contraignant l'exécution des noeuds.
- Les noeuds notés v0 et vn sont respectivement : le noeud source et le noeud puits du graphe.



Calcul de la latence minimum de l'implantation matérielle : T(x)+T(+)+T(-)

Modèle formel de la 2d-DCT 8x8



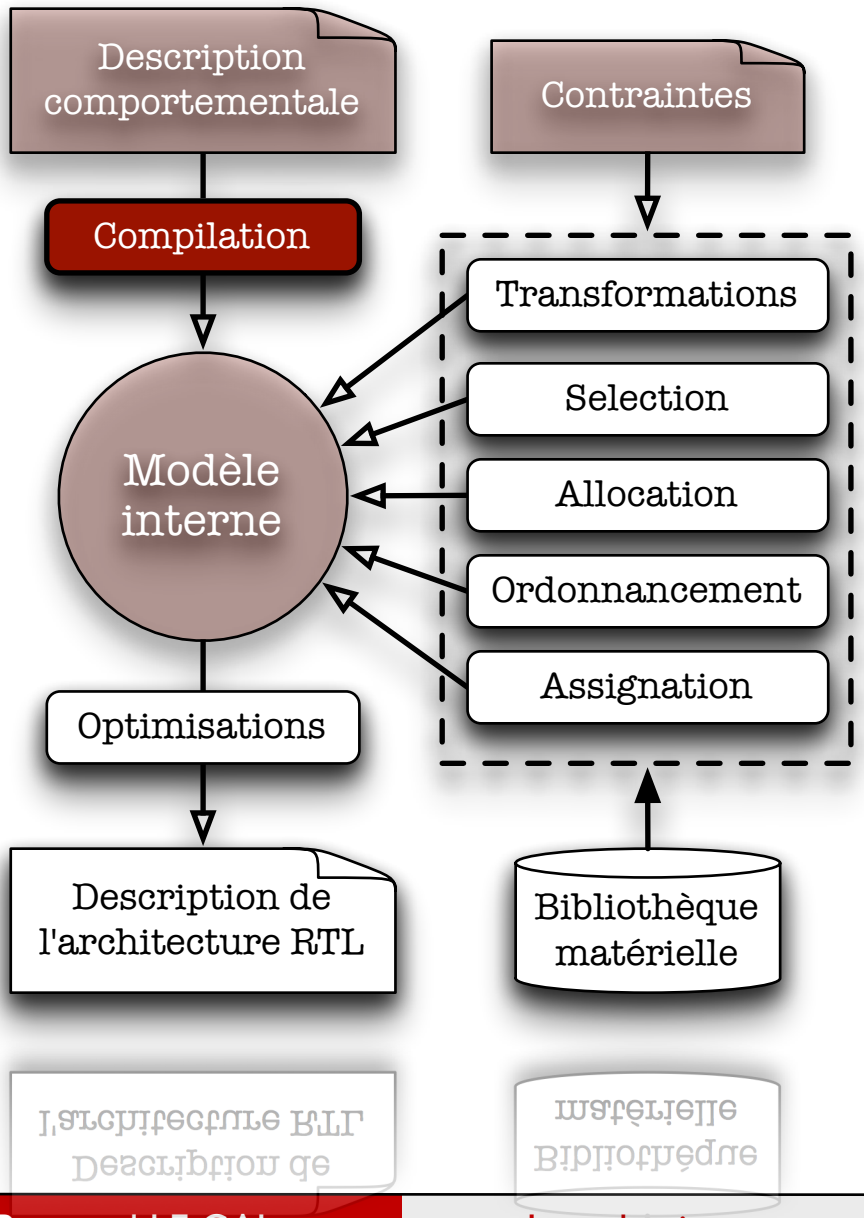
Voici un morceaux du modèle formel modélisant la 2d-DCT 8x8.

Le modèle est composé de 2814 noeuds répartis en:

- 1445 noeuds variables
- 1367 noeuds opérations

Ce modèle est adapté au traitement informatique du problème et non à la lisibilité humaine...

La compilation de la description algorithmique



Les langages de descriptions sont compréhensibles par les concepteurs mais comment aboutit on à un modèle formel exploitable par l'outil de HLS

=> Quelles sont les règles de transformation ?

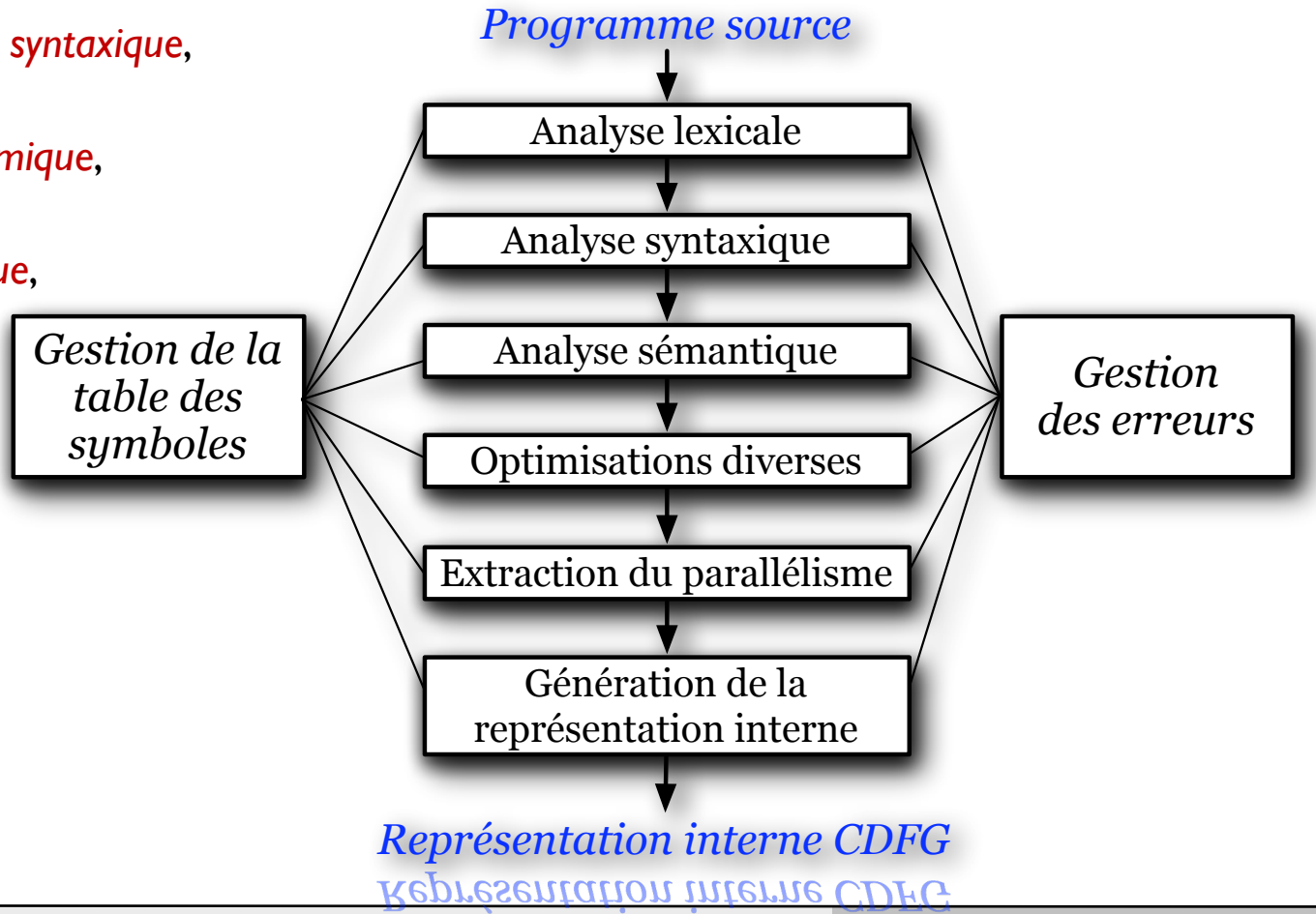
Objectifs de la phase de compilation algorithmique

- La phase de compilation de la description algorithmique vise à remplir plusieurs fonctions :
  - Vérifier la validité sémantique et syntaxique de la description algorithmique,
    - Vérifier une utilisation correcte du langage,
    - Différent de la validité fonctionnelle,
  - Optimiser la description algorithmique afin de la rendre plus optimale pour ses futures utilisations
    - Supprimer le code mort qui est inutile,
    - Transformer certaines instructions,
    - Changer la forme d'écriture des boucles, etc...
  - Transformer le code algorithmique sous forme d'un modèle formel pour la suite des opérations,

La phase de compilation de la description

- La phase de compilation de la description algorithmique vise à remplir plusieurs fonctions :

1. *Vérifier la validité sémantique et syntaxique*,
2. *Optimiser la description algorithmique*,
3. *Transformer le code algorithmique*,



Extraction du parallélisme d'une application

- 2 instructions nommées A et B peuvent être exécutées dans un ordre quelconque (A avant B et B avant A) à condition de ne pas être dépendante l'une de l'autre,
- Les fausses dépendances peuvent être cassées à l'aide de transformations du code source,

Considérons :

Expression A X = C - D;
Expression B Y = E - D;

Posons :

R(a) : ensemble des données lues dans l'expression A;
W(a) : ensemble des données modifiées dans l'expression A;

Les opérations A et B sont indépendantes (Conditions de Bernstein) si :

1. $W(a) \cap R(b) = \varnothing$ // Dépendance vraie
2. $R(a) \cap W(b) = \varnothing$ // Anti-dépendance ou consommateur-producteur,
3. $W(a) \cap W(b) = \varnothing$ // Dépendance de sortie ou producteur-producteur,

Quelques exemples de codes algorithmiques

```
a = b + c
d = e + c
```

```
a = b + c
d = c + x
```

```
a = b - c
a = 2 * a
```

```
a = b - c
d = b * c
```

```
a = b - c
d = 2 * a
```

```
a = b - c
a = 2 * y
```

```
z = b - c
c = 2 * a
```

```
a = b - c
b = 2 * y
c = 3 - a
```

Quelques exemples de codes algorithmiques

```
a = b + c
d = e + c
```

Pas de dépendance

```
a = b + c
d = c + x
```

```
a = b - c
a = 2 * a
```

```
a = b - c
d = b * c
```

```
a = b - c
d = 2 * a
```

```
a = b - c
a = 2 * y
```

```
z = b - c
c = 2 * a
```

```
a = b - c
b = 2 * y
c = 3 - a
```

Quelques exemples de codes algorithmiques

```
a = b + c
d = e + c
```

Pas de dépendance

```
a = b + c
d = c + x
```

```
a = b - c
a = 2 * a
```

Double dépendance

```
a = b - c
d = b * c
```

```
a = b - c
d = 2 * a
```

```
a = b - c
a = 2 * y
```

```
z = b - c
c = 2 * a
```

```
a = b - c
b = 2 * y
c = 3 - a
```

Quelques exemples de codes algorithmiques

```
a = b + c
d = e + c
```

Pas de dépendance

```
a = b + c
d = c + x
```

```
a = b - c
a = 2 * a
```

Double dépendance

```
a = b - c
d = b * c
```

```
a = b - c
d = 2 * a
```

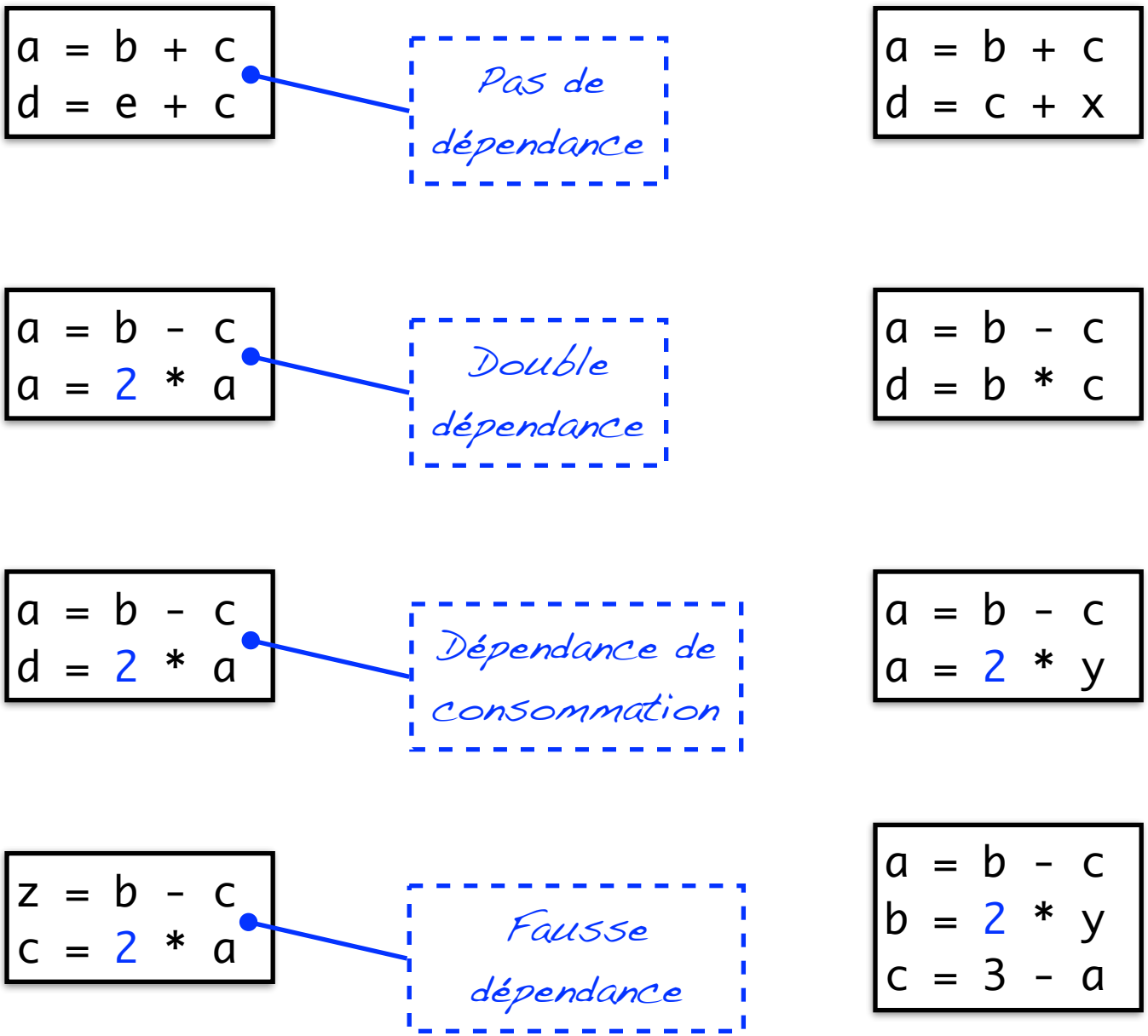
Dépendance de consommation

```
a = b - c
a = 2 * y
```

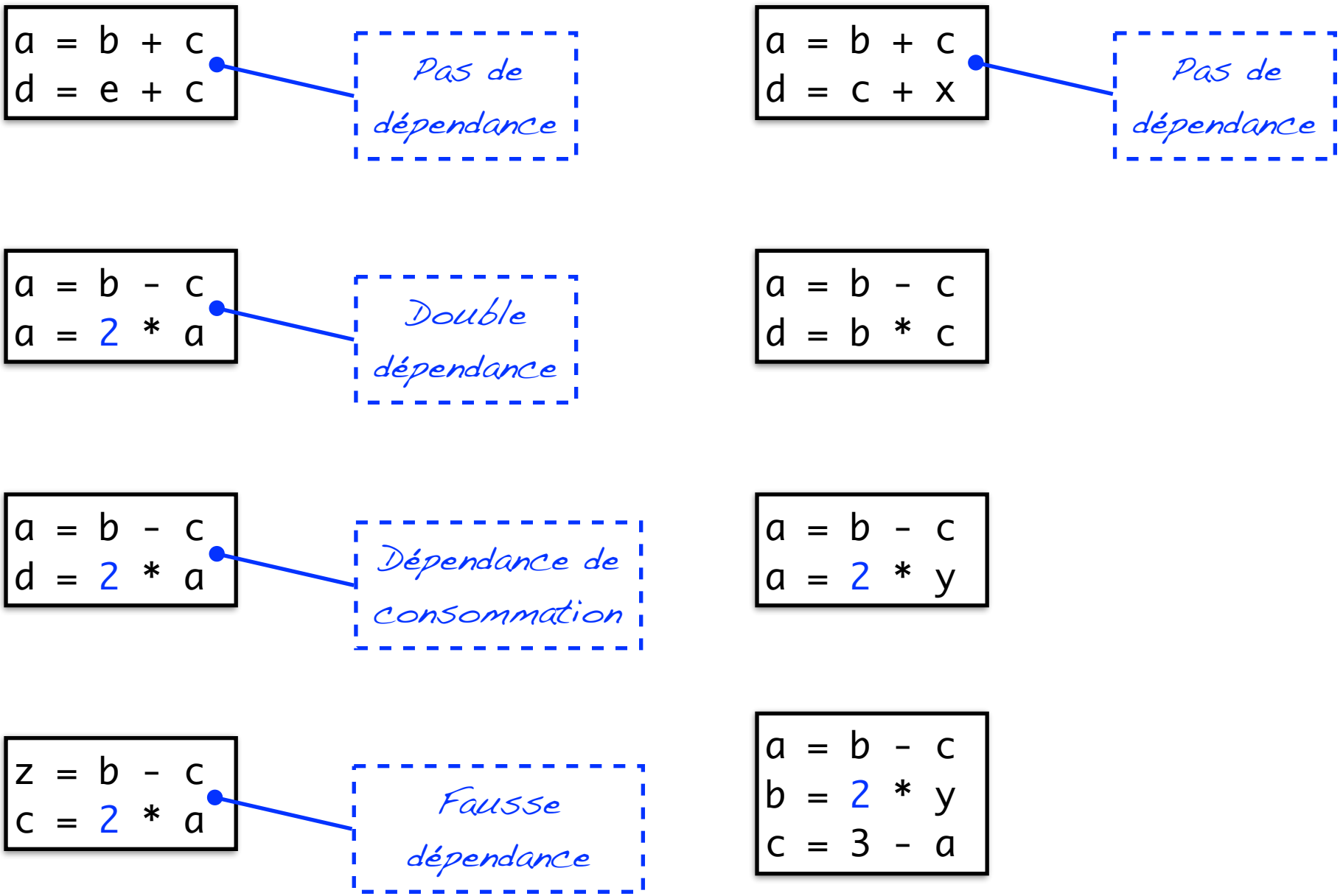
```
z = b - c
c = 2 * a
```

```
a = b - c
b = 2 * y
c = 3 - a
```

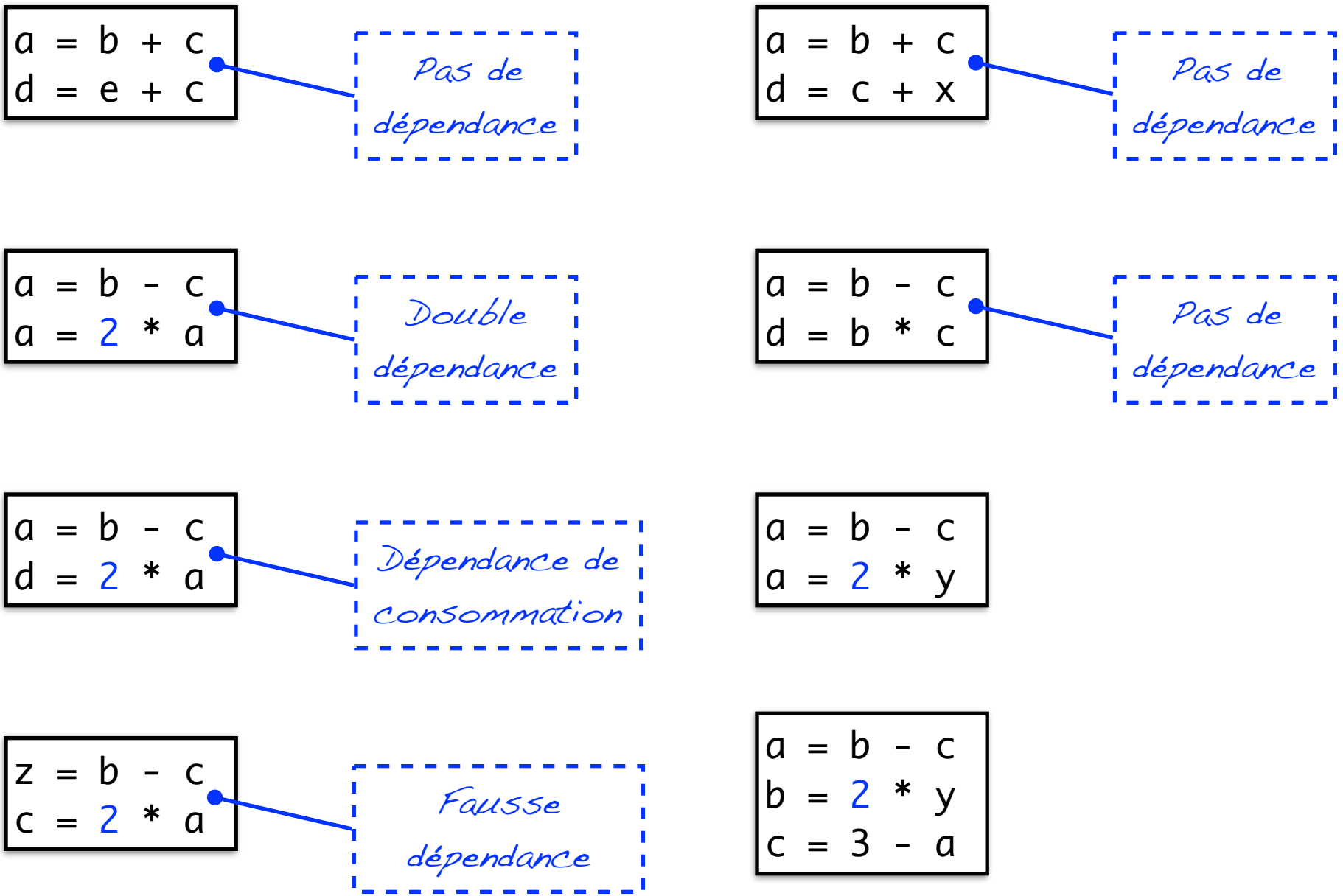
Quelques exemples de codes algorithmiques



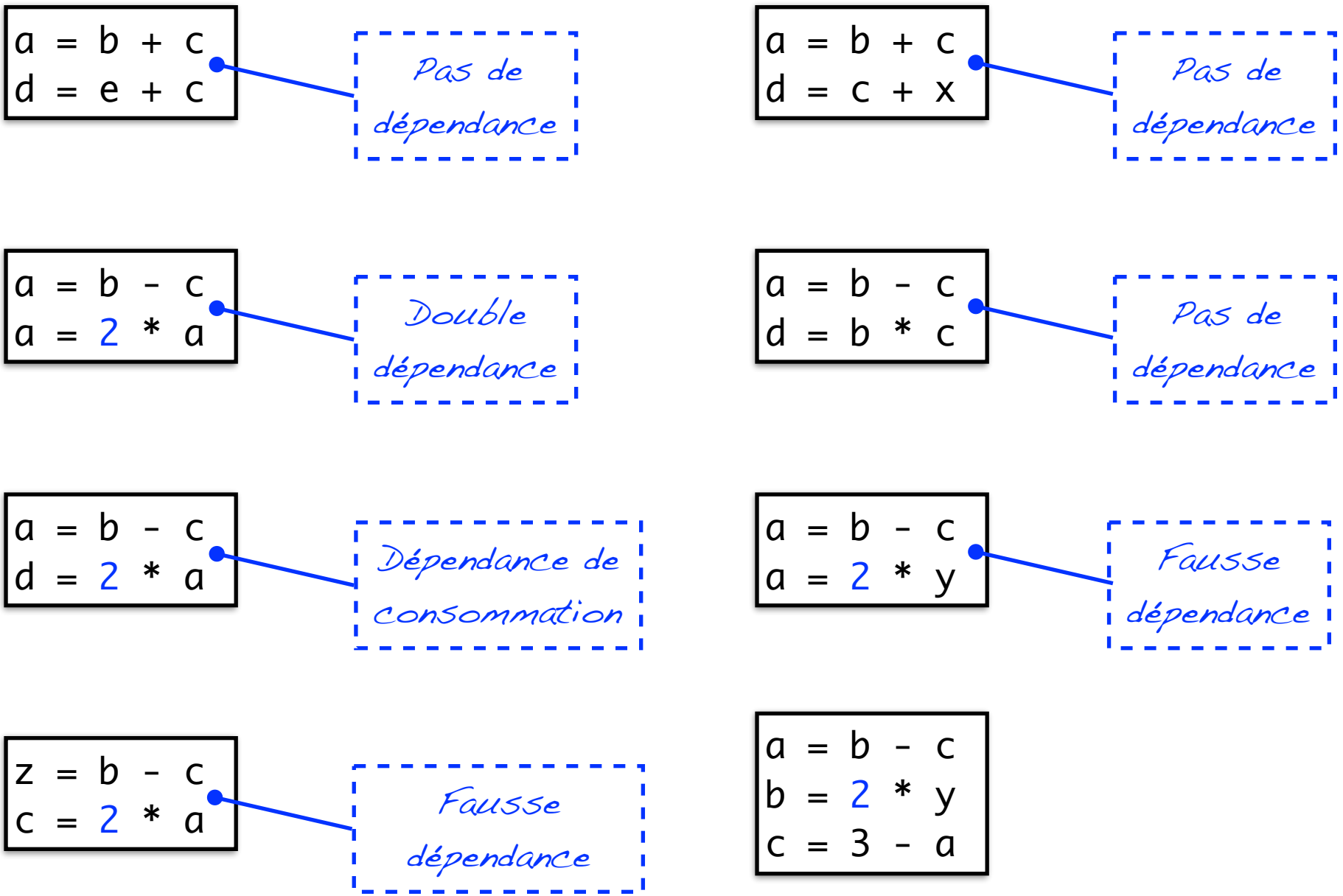
Quelques exemples de codes algorithmiques



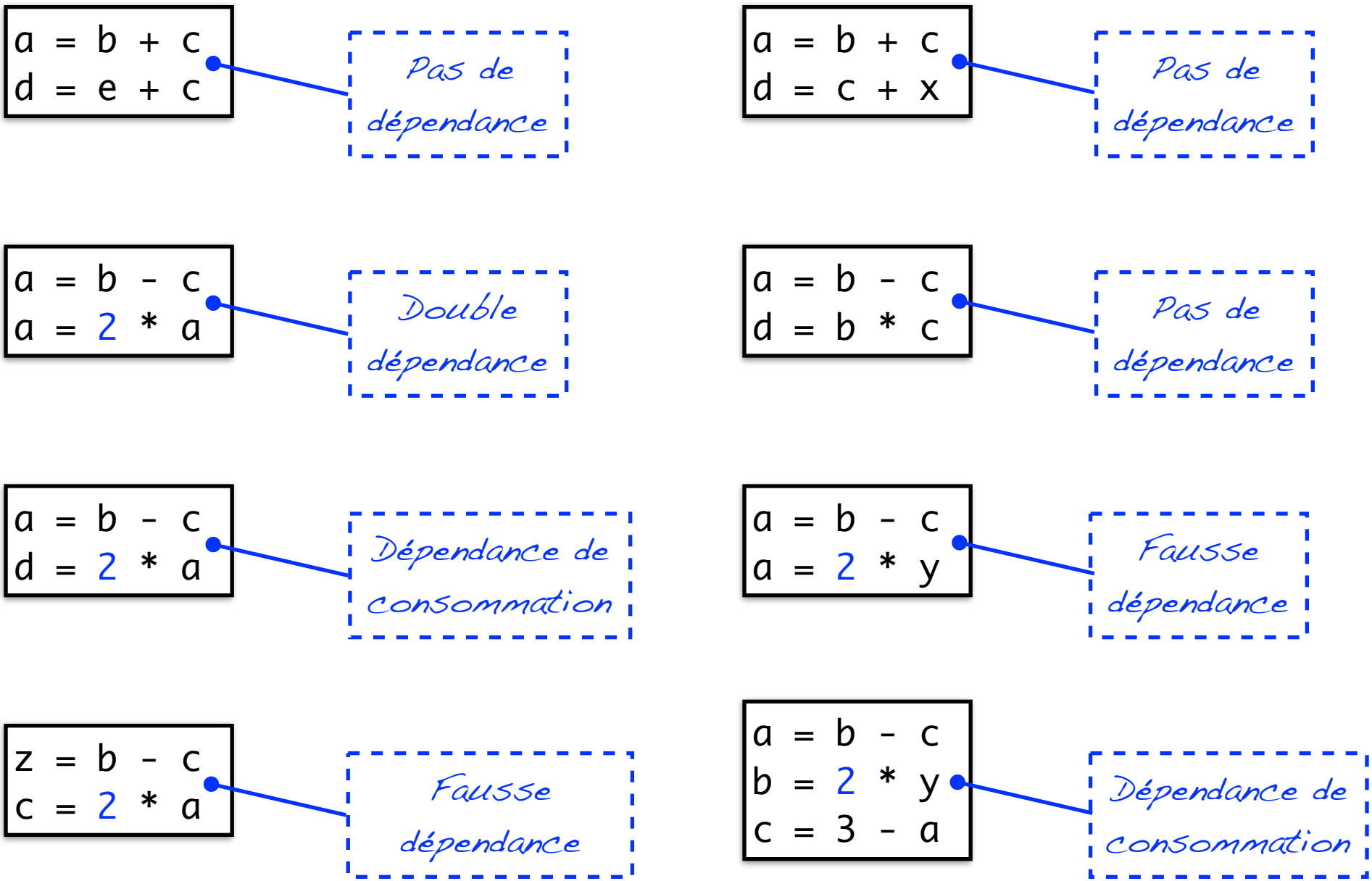
Quelques exemples de codes algorithmiques



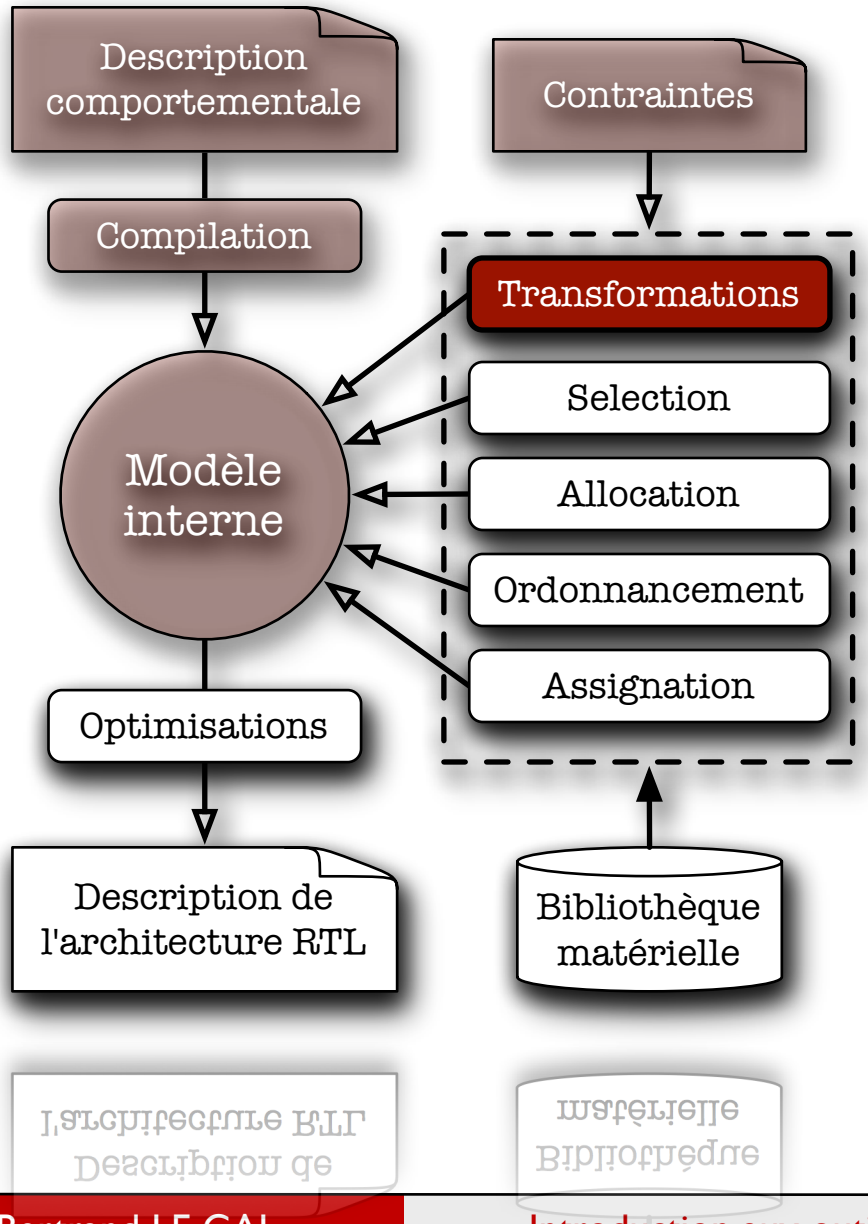
Quelques exemples de codes algorithmiques



Quelques exemples de codes algorithmiques



Transformations formelles du modèle



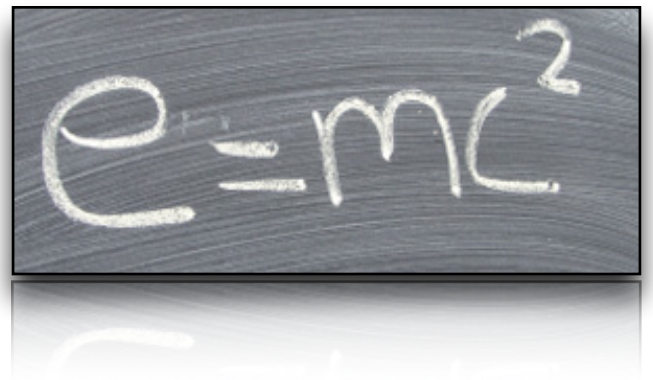
A partir de la description comportementale nous avons produit un modèle interne spécifiant le comportement à implanter

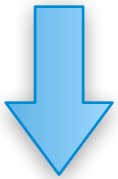
=> Peut'on améliorer les caractéristiques du modèle formel ?

=> Comment optimiser la description générique faite par le concepteur ?

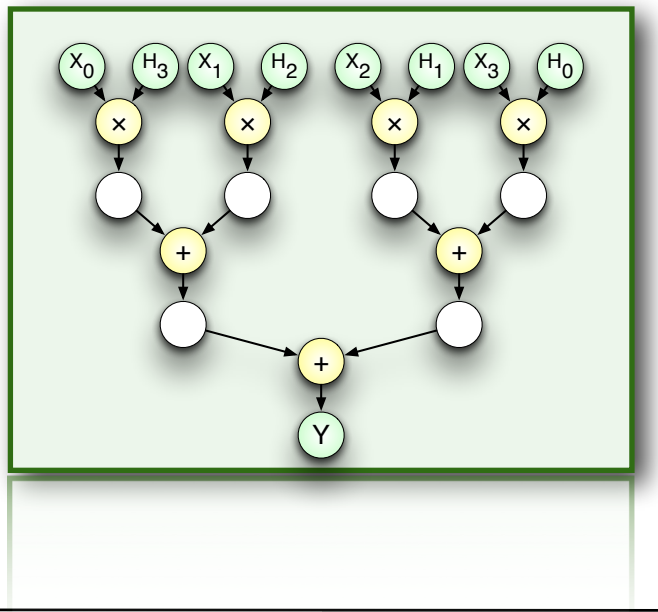
Transformations sur le modèle formel

- L'objectif de cette étape est:
  - D'améliorer les caractéristiques de la description algorithmique,
  - Rendre la description invariante du style d'écriture du code source,
- Différentes techniques
  - La propagation des constantes,
  - Factorisation des calculs identiques,
  - Suppression des calculs invariants,
  - Le remplacement d'instructions,
  - Déroulage partiel ou total des boucles,
- Conséquences:
  - Amélioration ou dégradation de certains paramètres du circuit final.





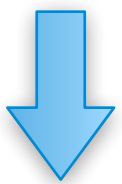
Transformation



Transformations sur le modèle formel

- L'objectif de cette étape est:
  - D'améliorer les caractéristiques de la description algorithmique,
  - Rendre la description invariante du style d'écriture du code source,
- Différentes techniques
  - La propagation des constantes,
  - Factorisation des calculs identiques,
  - Suppression des calculs invariants,
  - Le remplacement d'instructions,
  - Déroulage partiel ou total des boucles,
- Conséquences:
  - Amélioration ou dégradation de certains paramètres du circuit final.

```
int a = 2 * sin(-π/2);
int b = (2 + a) / 2 + 6;
int c = z - b /2 + 1;
```



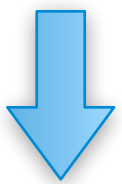
Propagation des expressions constantes

```
int a = -2;
int b =  6;
int c = z - 1;
```

Transformations sur le modèle formel

- L'objectif de cette étape est:
  - D'améliorer les caractéristiques de la description algorithmique,
  - Rendre la description invariante du style d'écriture du code source,
- Différentes techniques
  - La propagation des constantes,
  - Factorisation des calculs identiques,
  - Suppression des calculs invariants,
  - Le remplacement d'instructions,
  - Déroulage partiel ou total des boucles,
- Conséquences:
  - Amélioration ou dégradation de certains paramètres du circuit final.

```
int a = 2 * sin(delta * π);
int b = 6 + sin(delta * π);
int c = sin(delta * π)^2;
```



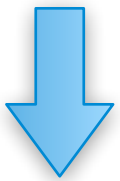
Optimisation des sous expressions communes

```
int t = sin(delta * π);
int a = 2 * t;
int b = 6 + t;
int c = t^2;
```

Transformations sur le modèle formel

- L'objectif de cette étape est:
  - D'améliorer les caractéristiques de la description algorithmique,
  - Rendre la description invariante du style d'écriture du code source,
- Différentes techniques
  - La propagation des constantes,
  - Factorisation des calculs identiques,
  - Suppression des calculs invariants,
  - Le remplacement d'instructions,
  - Déroulage partiel ou total des boucles,
- Conséquences:
  - Amélioration ou dégradation de certains paramètres du circuit final.

```
for(i=0; i<16; i++){
   a[i] = x + y;
}
```



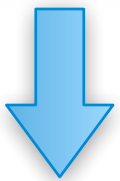
La somme des données X et Y est invariable

```
int t = x + y;
for(i=0; i<16; i++){
   a[i] = t;
}
```

Transformations sur le modèle formel

- L'objectif de cette étape est:
  - D'améliorer les caractéristiques de la description algorithmique,
  - Rendre la description invariante du style d'écriture du code source,
- Différentes techniques
  - La propagation des constantes,
  - Factorisation des calculs identiques,
  - Suppression des calculs invariants,
  - Le remplacement d'instructions,
  - Déroulage partiel ou total des boucles,
- Conséquences:
  - Amélioration ou dégradation de certains paramètres du circuit final.

```
for(i=0; i<16; i++){
   a[i] = 4*i;
}
i = (i * 4);
i = (i * 3);
```



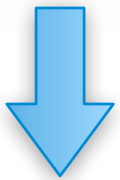
Transformation de certaines opérations

```
t = 0;
for(i=0; i<16; i++){
   a[i] = t;
   t = t+4;
}
i = (i << 2);
i = (i << 1) + i;
```

Transformations sur le modèle formel

- L'objectif de cette étape est:
  - D'améliorer les caractéristiques de la description algorithmique,
  - Rendre la description invariante du style d'écriture du code source,
- Différentes techniques
  - La propagation des constantes,
  - Factorisation des calculs identiques,
  - Suppression des calculs invariants,
  - Le remplacement d'instructions,
  - Déroulage partiel ou total des boucles,
- Conséquences:
  - Amélioration ou dégradation de certains paramètres du circuit final.

```
y = x[0] * h[3];
for(i=1; i<4; i++){
   y = y + x[i] * h[3-i];
}
y = y / 4;
```



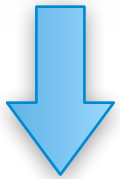
Déroulage intégral

```
y0 =       x[0] * h[3];
y1 = y0 + x[1] * h[2];
y2 = y1 + x[2] * h[1];
y3 = y2 + x[3] * h[0];
y  = y3 / 4;
```

Transformations sur le modèle formel

- L'objectif de cette étape est:
  - D'améliorer les caractéristiques de la description algorithmique,
  - Rendre la description invariante du style d'écriture du code source,
- Différentes techniques
  - La propagation des constantes,
  - Factorisation des calculs identiques,
  - Suppression des calculs invariants,
  - Le remplacement d'instructions,
  - Déroulage partiel ou total des boucles,
- Conséquences:
  - Amélioration ou dégradation de certains paramètres du circuit final.

```
y = x[0] * h[3];
for(i=1; i<4; i++){
   y = y + x[i] * h[3-i];
}
y = y / 4;
```



Déroulage partiel d'un facteur 2

```
y = x[0] * h[3];
for(i=1; i<4; i+=2){
   y = y + x[i]   * h[3-i];
   y = y + x[i+1] * h[2-i];
}
y = y / 4;
```

Optimisations des opérations arithmétiques

- Quelles sont les optimisations arithmétiques,
  - Addition,
    - B = (A + 0) = (0 + A) = A
  - Soustraction,
    - B = (A - 0) = A
    - B = (A - A) = 0
  - Multiplications,
    - B = (A x 0) = 0
    - B = (A x 1) = (1 x A) = A
  - Divisions,
    - B = (A / 1) = A
    - B = (0 / A) = 0
    - B = (A / A) = 1
  - Opérations complexes (MAC)...

- Quelles sont les optimisations logiques,
  - Fonction AND,
    - B = (A . 0) = (0 .A) = 0
    - B = (A . 1) = (1 .A) = 1
  - Fonction OR,
    - B = (A + 0) = (A + 1) = (1 + A) = (A + 0) = A
  - Fonction XOR,
    - B = (A xor 0) = (0 xor A) = A
    - B = (A xor 1) = (1 xor A) = 0
  - Fonction NAND / NOR,
    - idem...

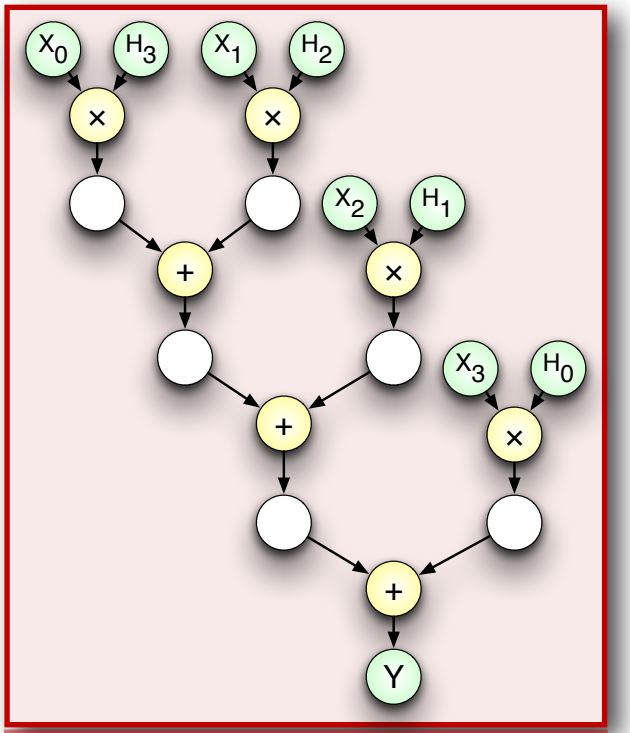
Liste des fonctions usuelles à propager (cas constant)

- Fonctions mathématiques usuelles à traiter
  - *min*/*max*(cste, cste) = cste,
  - *sqr*/*sqrt*(cste) = cste,
  - abs(cste) = cste,
  - Qn(cste, cste) = cste,
  - *sin*/*cos*/*tan*(cste) = cste,
  - log(cste) = cste,
  - log2(cste) = cste,
  - exp(cste) = cste,
  - pow(cste, cste) = cste
  - select(cond, cste, cste) = cste
- Optimisations facultatives (cibles dépendantes)
  - Addition
    - B = (A + A) = (2 * A) = (A << 1)
  - Multiplication,
    - B = (2 * A) = (A + A) = (A << 1)
  - Multiplication 2n,
    - B = (2^n * A) = (A * 2^n) = (A << n)
  - Division 2n,
    - B = (A / 2^n) = (A >> n)
  - D'autres optimisations visibles ?

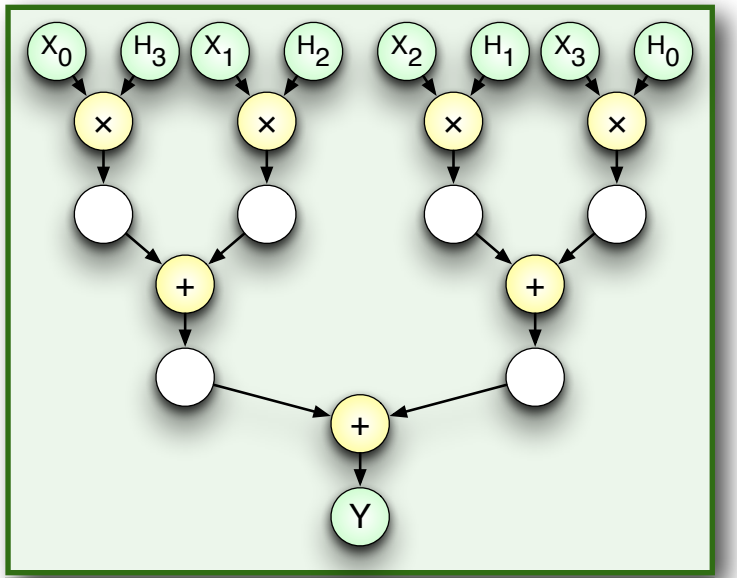
Impact du style d'écriture de l'algorithme

Filtre FIR de 512 points

- Ecriture classique : Tcc = T(×) + 511.T(+)
  - Application Numérique = 513 cycles
- Ecriture "raffinée" : Tcc = T(×) + 9.T(+)
  - Application Numérique = 11 cycles



$Y = X_0.H_3+X_1.H_2+X_2.H_1+X_3.H_0$



$Y = (X_0.H_3+X_1.H_2) + (X_2.H_1+X_3.H_0)$

Ecrire intelligemment sa description algorithmique ne produit pas toujours des circuits rapides mais cela y contribue si l'on met les moyens matériels afin d'exploiter le parallélisme,

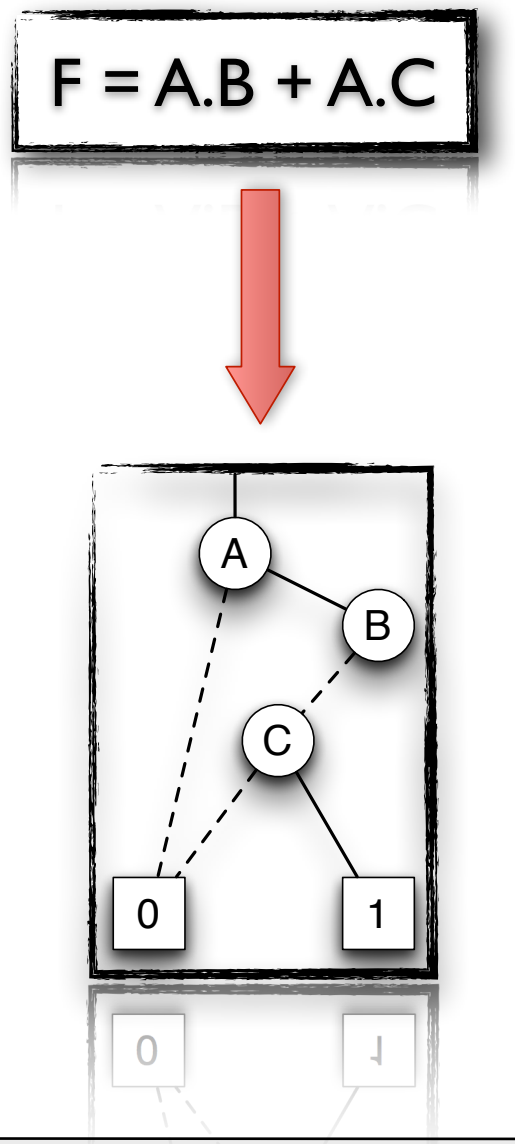
Réduction de la complexité calculatoire

M. Ciesielski, S. Askar, D. Gomez-Prado, J. Guillot, and E. Boutillon. Data-flow transformations using Taylor expansion diagrams. *In Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, April 16 - 20, 2007*.

$$F = A.B + A.C$$

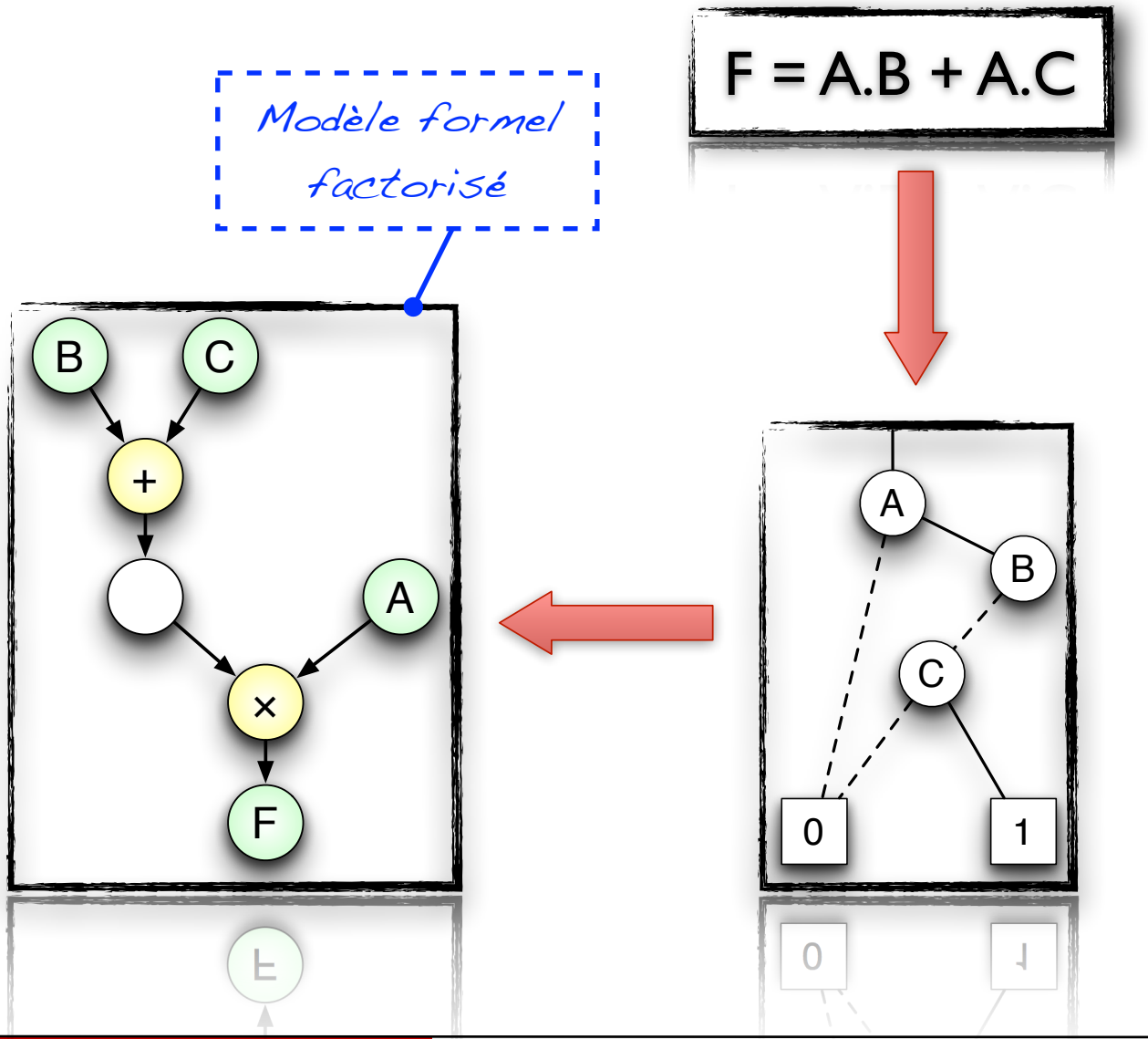
Réduction de la complexité calculatoire

M. Ciesielski, S. Askar, D. Gomez-Prado, J. Guillot, and E. Boutillon. Data-flow transformations using Taylor expansion diagrams. *In Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, April 16 - 20, 2007*.



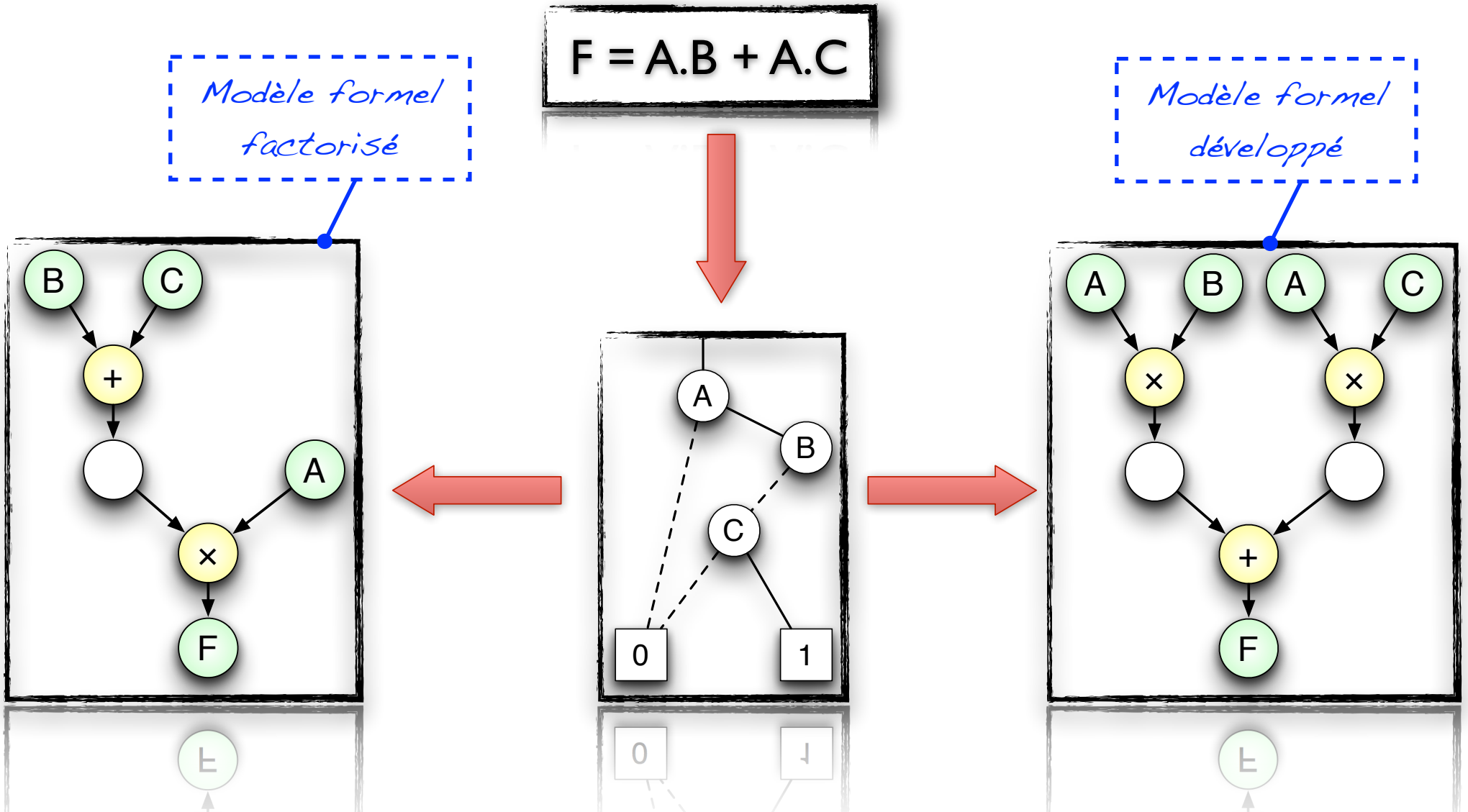
Réduction de la complexité calculatoire

M. Ciesielski, S. Askar, D. Gomez-Prado, J. Guillot, and E. Boutillon. Data-flow transformations using Taylor expansion diagrams. *In Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, April 16 - 20, 2007*.

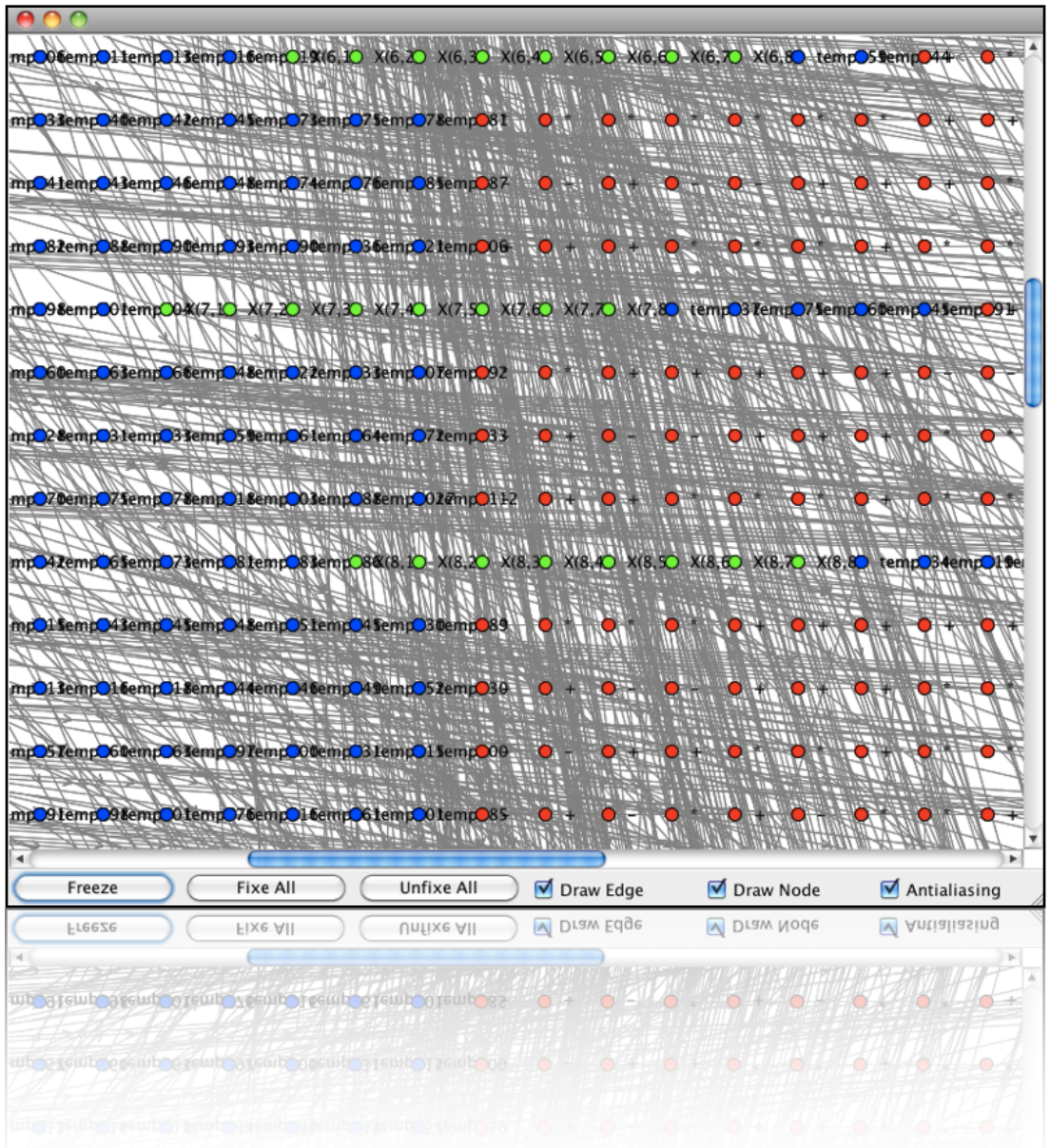


Réduction de la complexité calculatoire

M. Ciesielski, S. Askar, D. Gomez-Prado, J. Guillot, and E. Boutillon. Data-flow transformations using Taylor expansion diagrams. *In Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, April 16 - 20, 2007*.



Modèle formel optimisé de la 2d-DCT 8x8



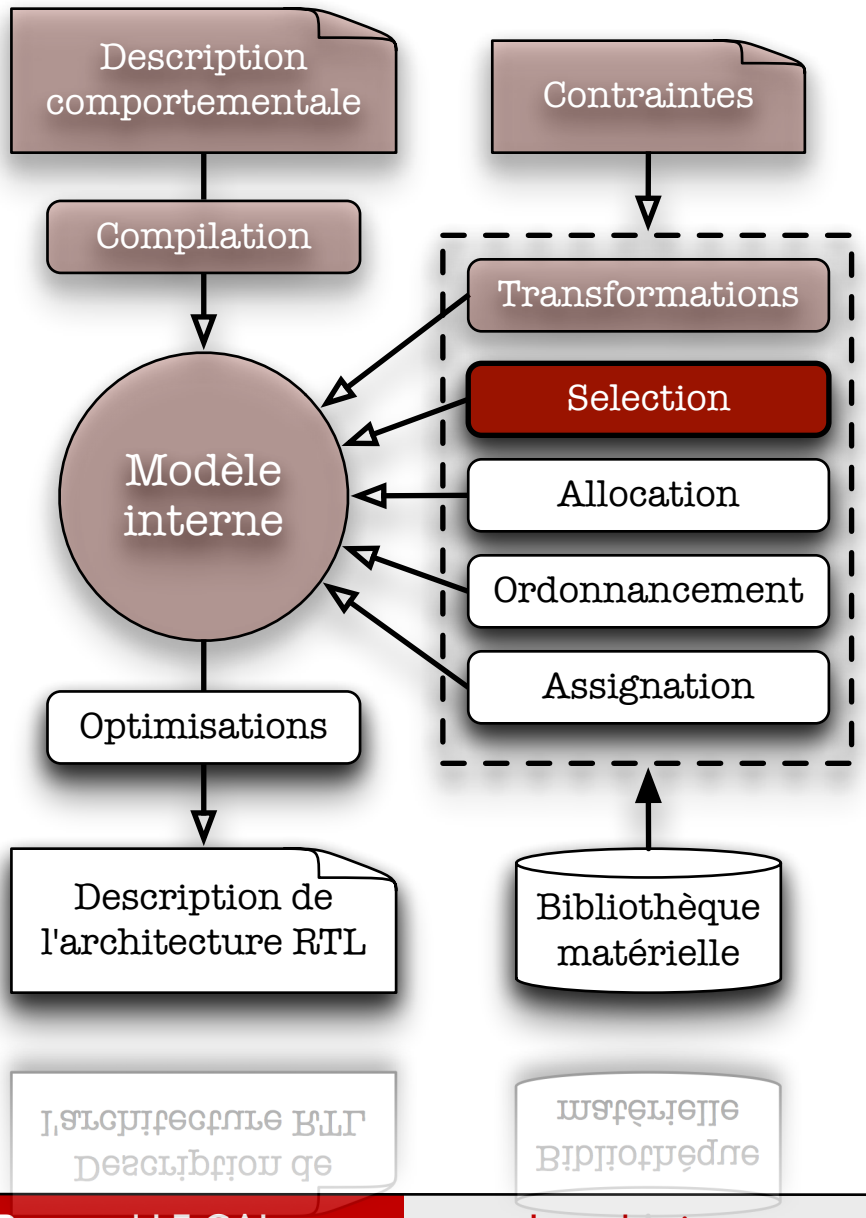
L'ensemble des calculs des coefficients a été réalisé.

Le sous expressions communes ont été factorisées :

- coeff = cos(cste)
- fp = 2^n * coeff

Cela permet de supprimer les calculs «inutiles» tout en conservant une flexibilité forte du code MatLab.

La sélection des ressources matérielles



En fonction du modèle, il faut déterminer les ressources matérielles nécessaires pour réaliser les calculs,

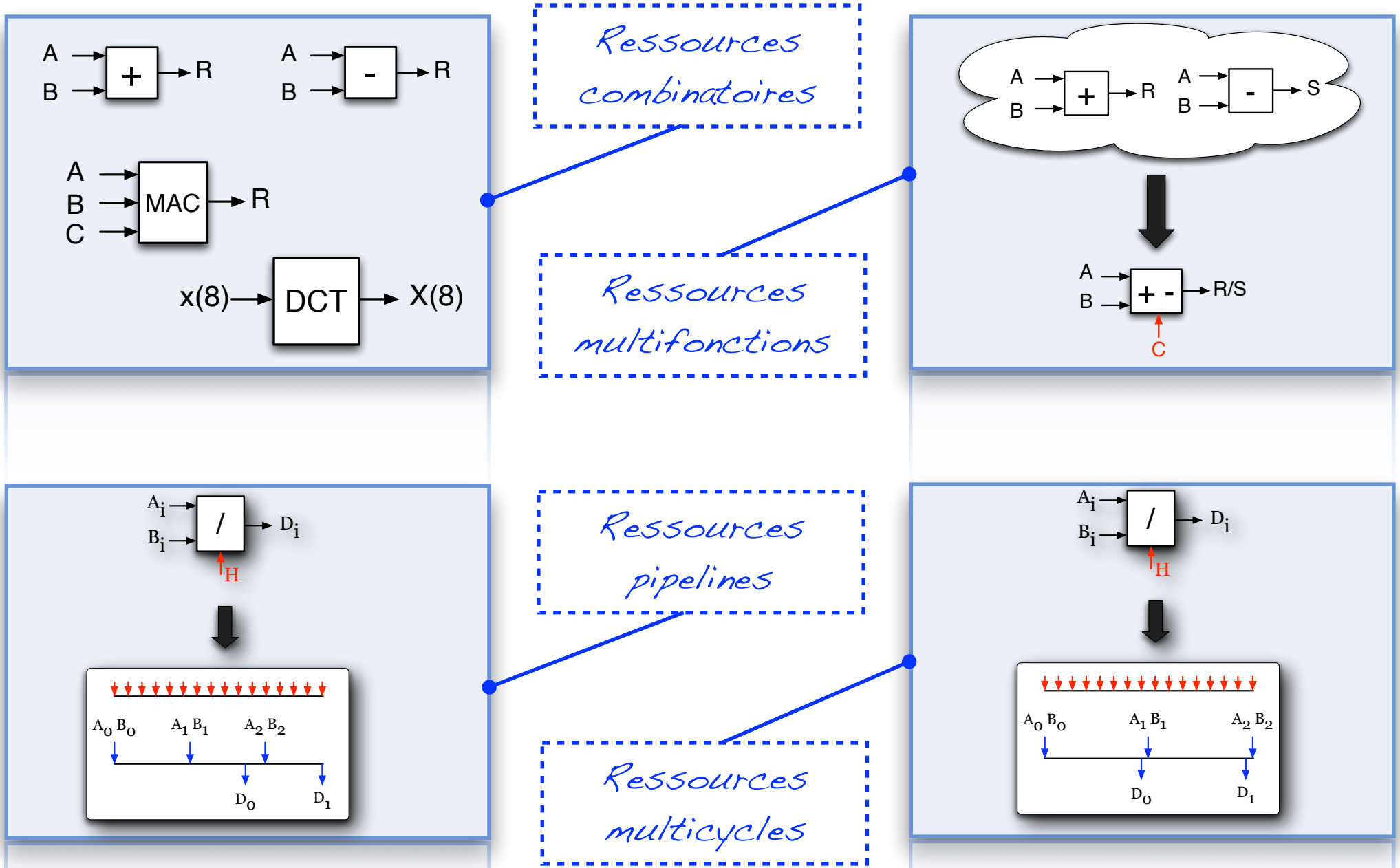
=> Pour chaque opération contenue dans le graphe doit posséder une implantation matérielle,

=> Les caractéristiques matérielles sont différentes en fonction des cibles technologiques (FPGA, ASIC)

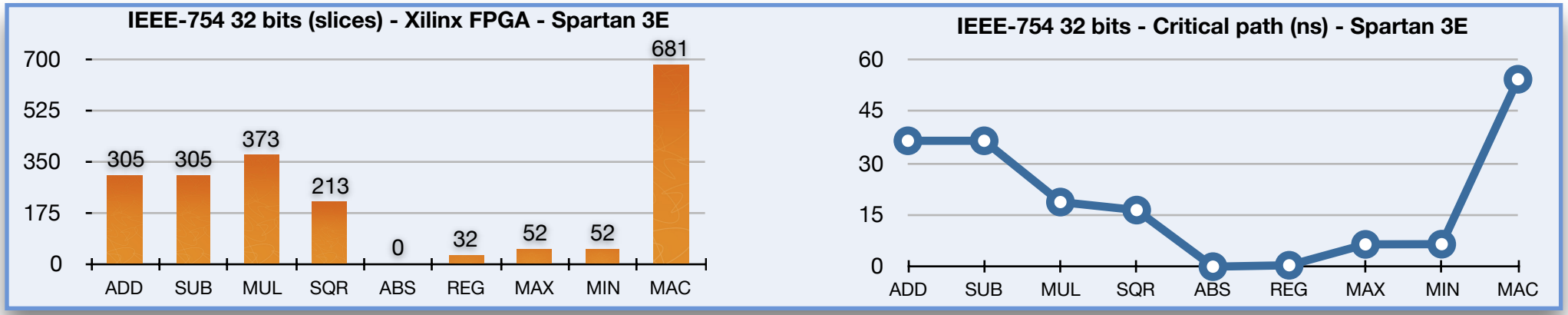
La sélection des opérateurs

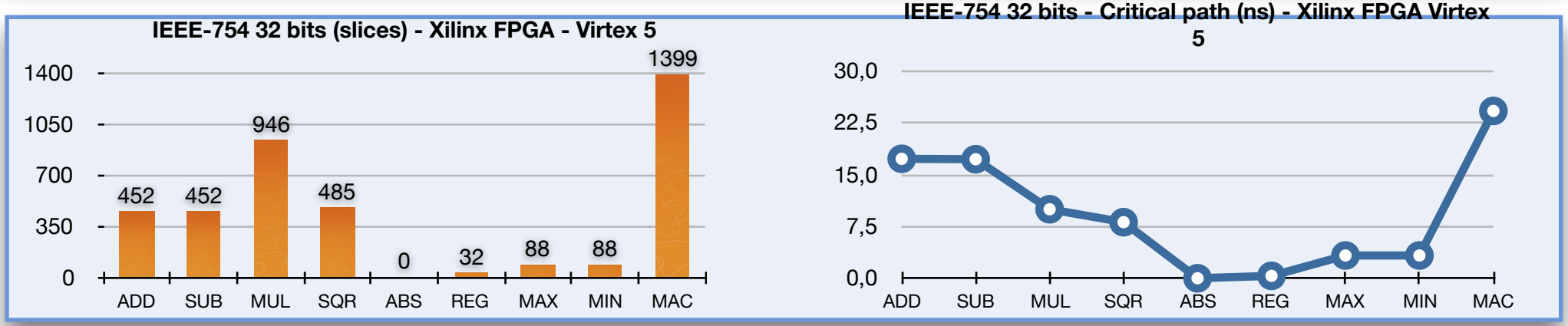
- Il est nécessaire de posséder plus ou moins d'informations sur les ressources matérielles à disposition,
  - La ou les fonctions réalisées par la ressource (+, -, x, MAC, FFT, etc...),
  - Les performances temporelles (latence, cadence => pipeline),
  - La consommation d'énergie (statique, dynamique),
  - Le coût matériel de l'opérateur (surface ou LUTs),
  - Le nombre de ressources de ce type disponibles sur la cible,
  - La dynamique (nombre de bits) de l'opérateur,
  - Le nombre d'entrées et de sorties du composant (sont elles commutatives ?),
- Toutes ces informations vont permettre de réaliser le choix des composants à mettre en oeuvre dans le «futur» circuit.

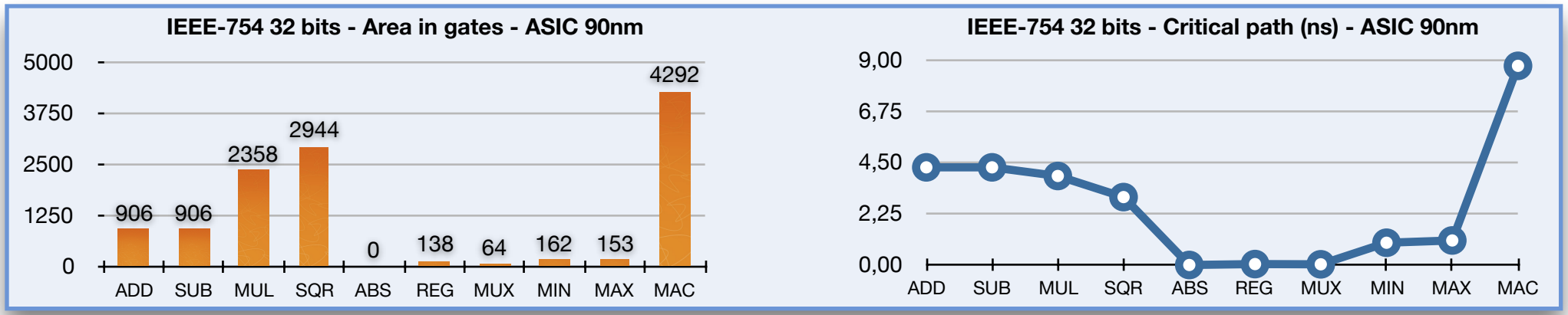
Différents types de ressources matérielles



Différences importantes entres les technologies



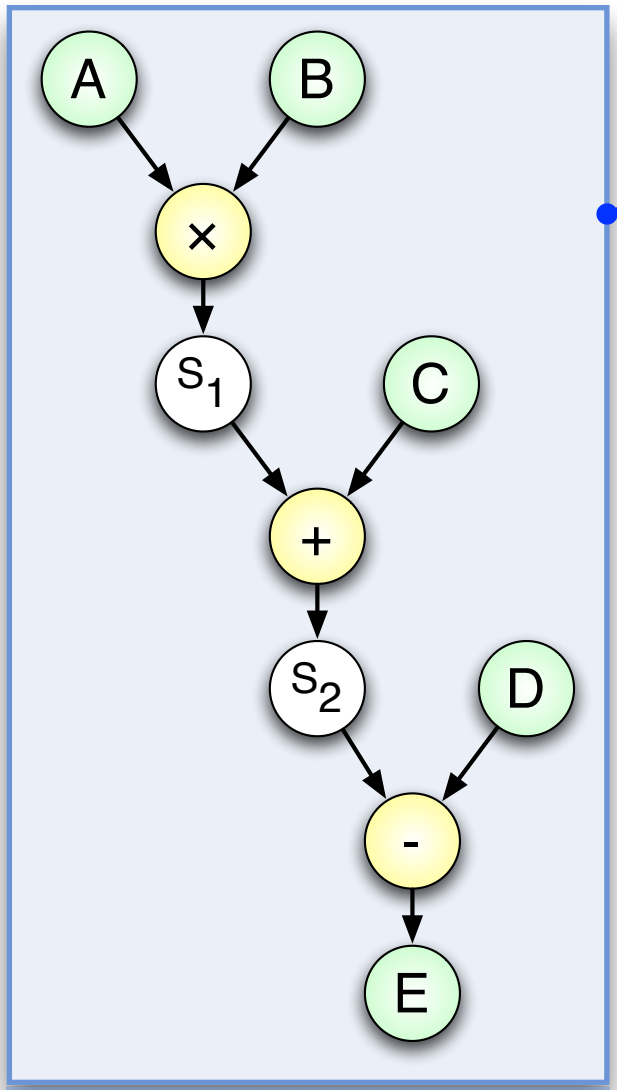




Liste des ressources matérielles intéressantes...

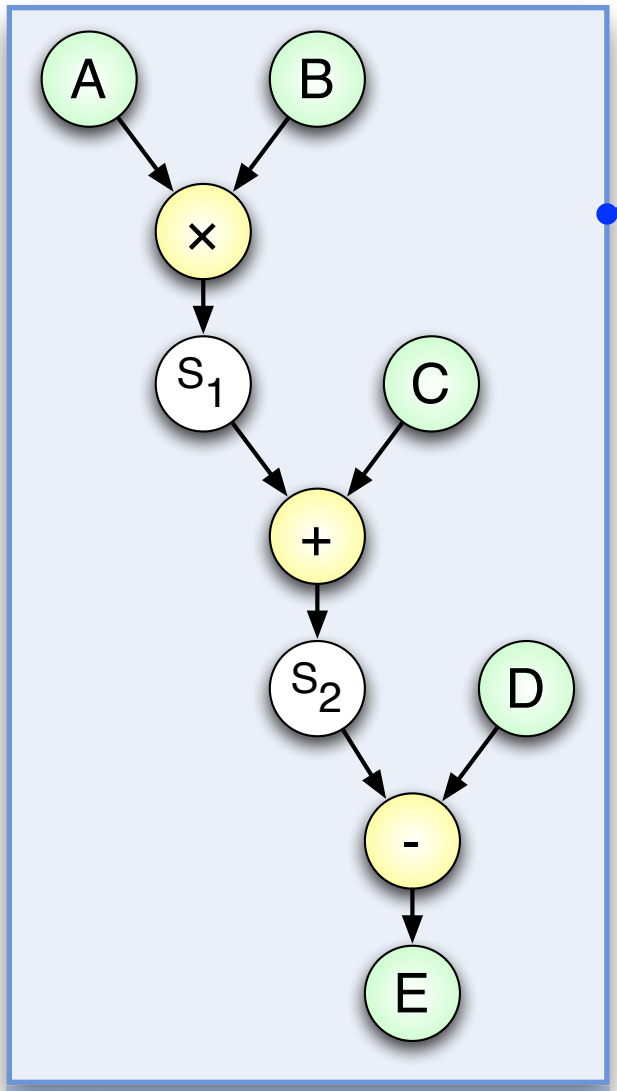
- Ressources matérielles disponibles
  - Arithmétique : Add, Sub, Mul, Div, Mac
  - Logique : And, Nand, Or, Nor, Not,
  - Autres : Square, SquareRoot, Shift (left, right), Abs, ACS, CMove
- Bibliothèques d'opérateurs
  - Nombres entiers 8 => N bits,
  - Nombres flottants 32 bits
  - Nombres flottants 64 bits
  - {Bibliothèques} sécurisée FS/FT
- Cibles technologiques caractérisées
  - Technologie FPGA : Xilinx (Spartan 3E, Virtex-5) et Altera (Cyclone 3, Stratix III)
  - Technologie ASIC : 65nm et 90nm (bibliothèque ST)

Sélection des ressources matérielles - Exemple



Dans le modèle formel on peut identifier 3 types d'opérations (+ × -)

Sélection des ressources matérielles - Exemple

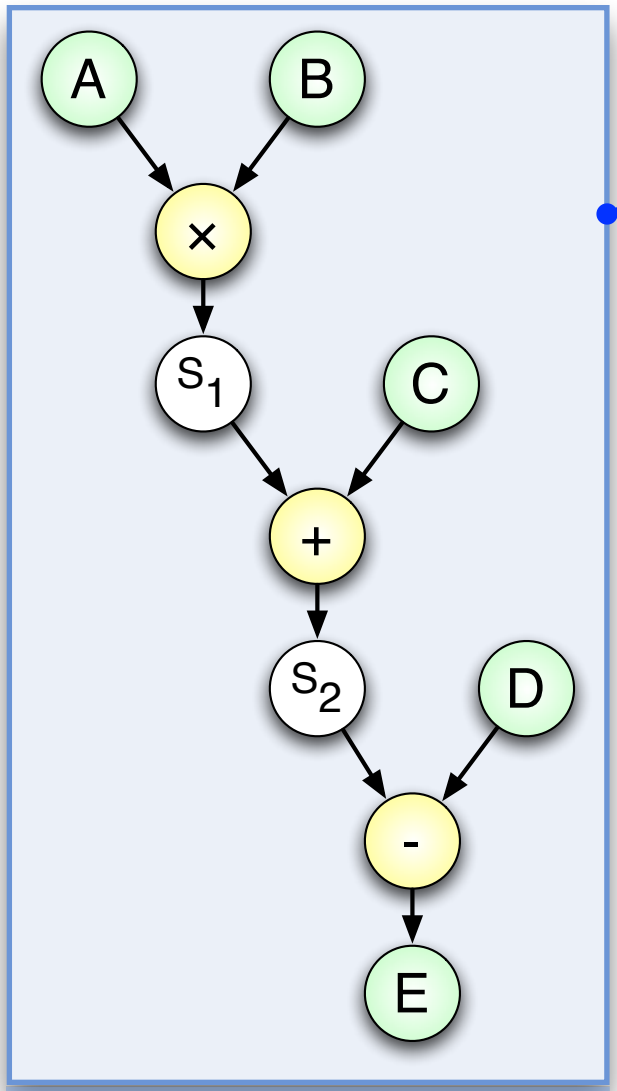


Dans le modèle formel on peut identifier 3 types d'opérations (+ × -)

Afin de pouvoir implémenter le modèle formel nous devrons sélectionner :

- Un additionneur (+)
- Un soustracteur (-)
- Un multiplieur (×)

Sélection des ressources matérielles - Exemple



Dans le modèle formel on peut identifier 3 types d'opérations (+ × -)

Afin de pouvoir implémenter le modèle formel nous devrons sélectionner :

- Un additionneur (+)
- Un soustracteur (-)
- Un multiplieur (×)

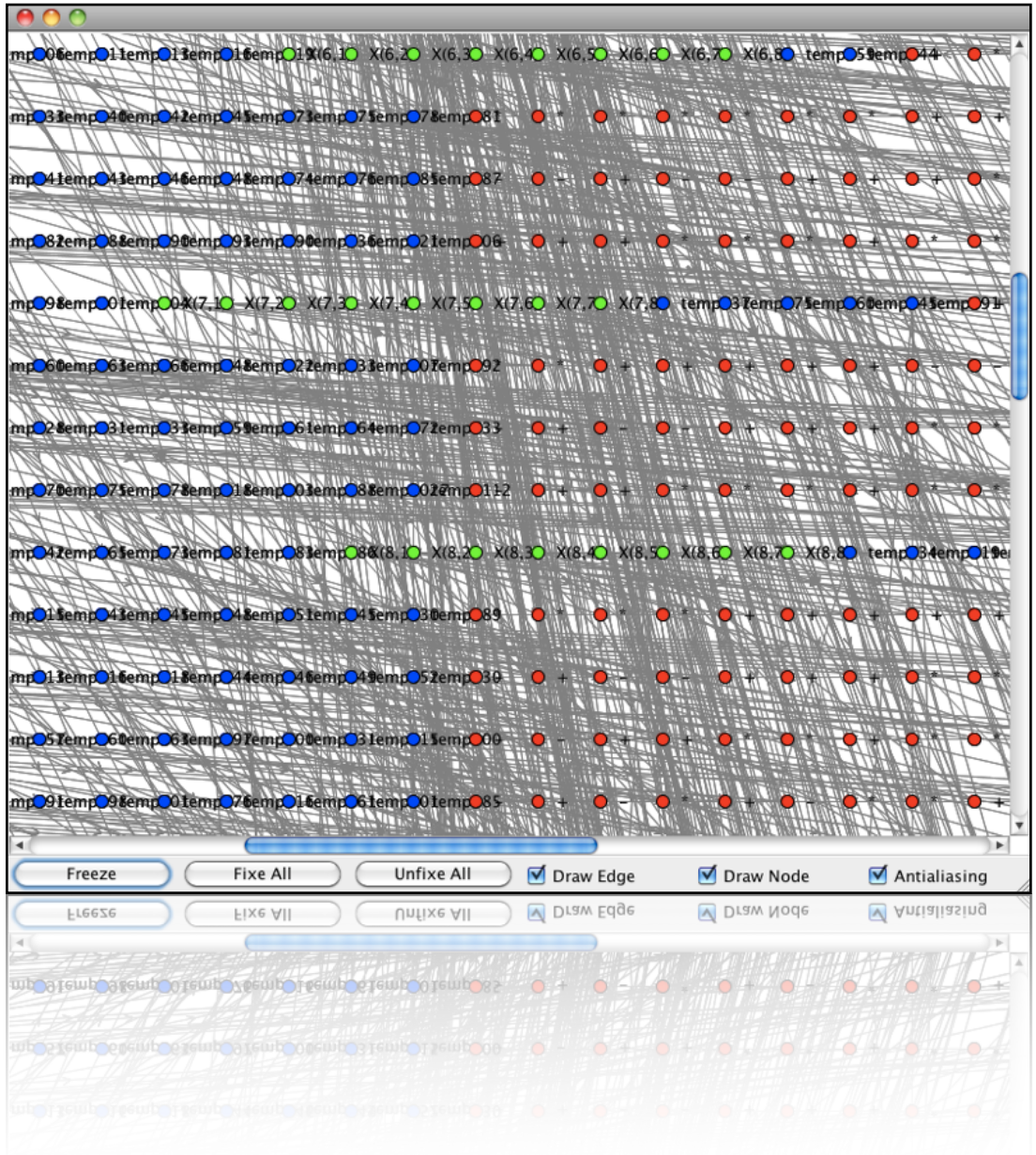
Ou une ALU (+ - ×)

Ou un additionneur-soustracteur (+ -) et un multiplieur (×)

La sélection des opérateurs

- Synthèse sous contrainte temporelle
  - L'outil doit choisir dans la bibliothèque les ressources qui seront utilisées dans la suite du flot de synthèse,
- Synthèse sous contrainte matérielle
  - Le concepteur doit faire lui même le choix des ressources matérielles à utiliser pour les étapes de synthèse du circuit,
- Il faut trouver un métrique de décision afin de sélectionner une ressource matérielle parmi N disponibles :
  - Sous contrainte temporelle, on essaye de minimiser la surface en choisissant les moins coûteux (ou des variantes en fonction des contraintes),
  - L'analyse est souvent mono-critère : si l'utilisateur souhaite un circuit économe en énergie, on choisira les opérateurs low power...
- Une alternative utilisée consiste à ne placer qu'un opérateur de chaque type en bibliothèque...

Modèle formel de la 2d-DCT 8x8 (Selection)



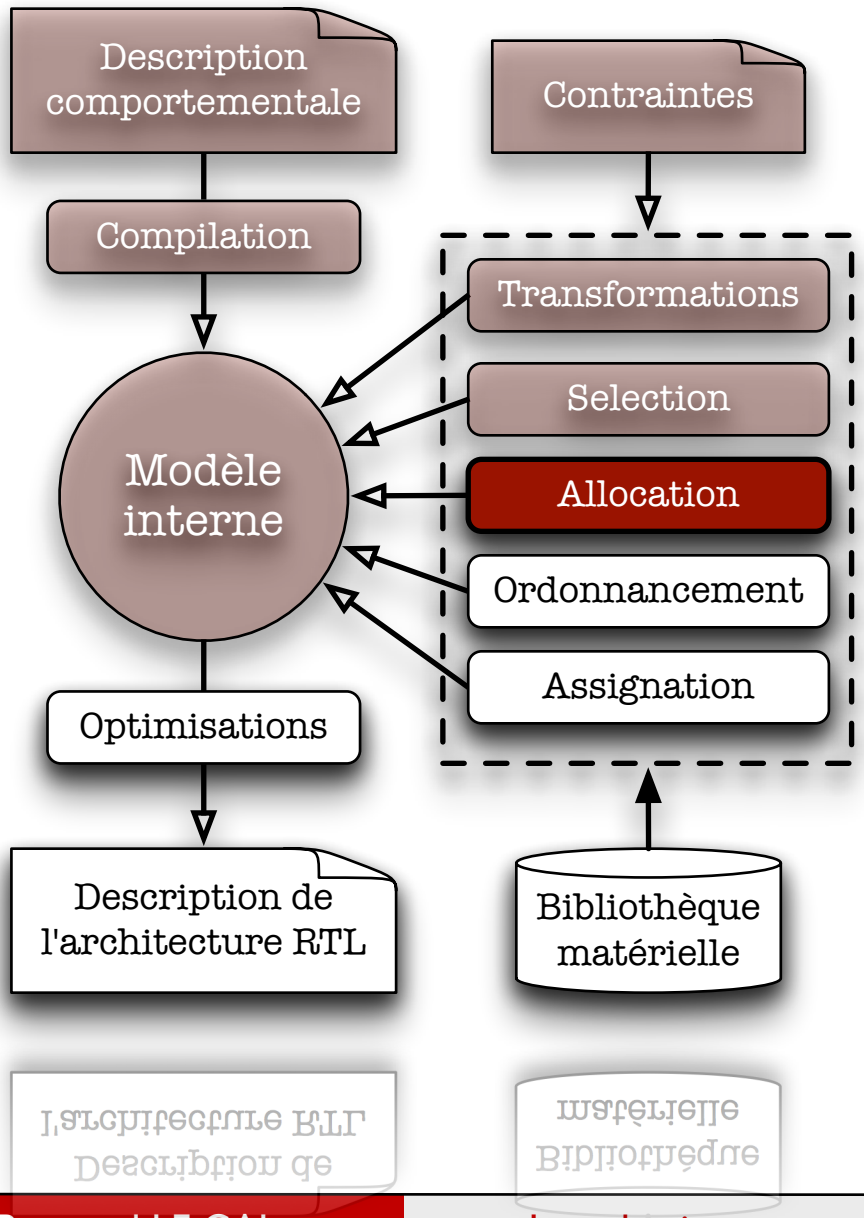
Dans la phase de sélection des ressources matérielles nous avons selectionné en bibliothèque:

- Une ressource de type (+)
- Une ressource de type (-)
- Une ressource de type (*)

Cette approche est la plus simple, nous aurions pu selectionner des opérateurs:

- multi-fonctions (+/-)
- pipeline (n tranches)

Allocation des ressources matérielles



On a déterminé les ressources matérielles utilisables en fonction de l'application et des contraintes

Combien de ressources matérielles doit on mettre en oeuvre :

=> afin de respecter la contrainte temporelle fournie par le concepteur,

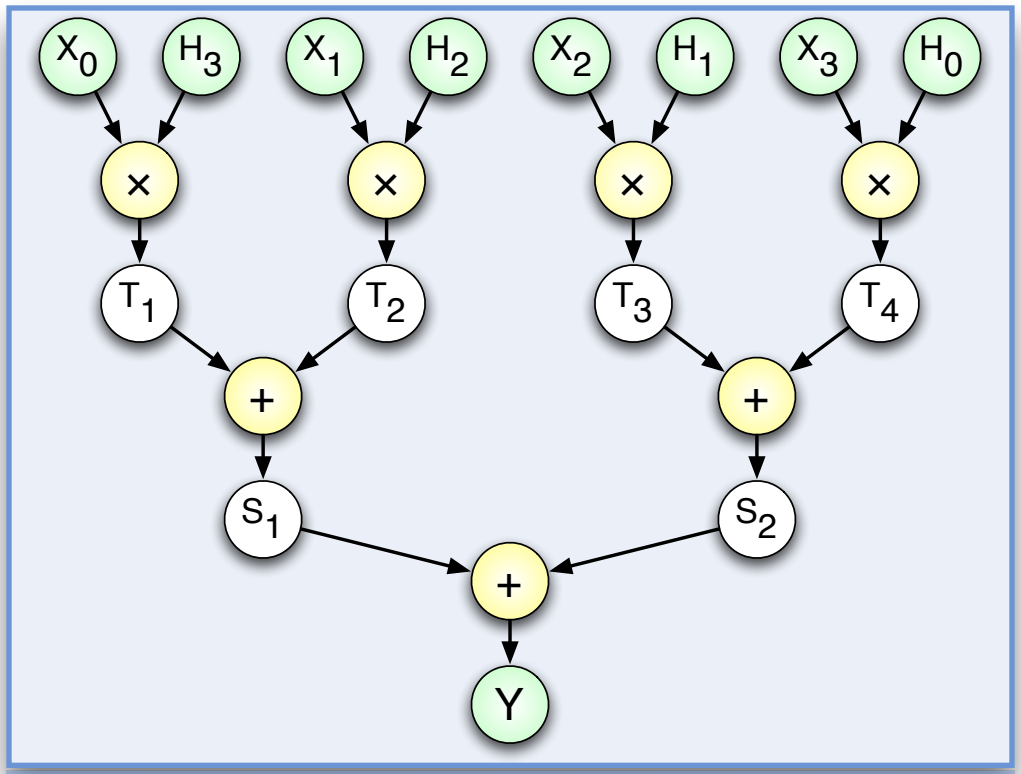
=> minimiser les autres paramètres,

Présentation des techniques d'allocation

- L'ensemble de ces techniques possèdent des cas d'utilisation particuliers ainsi qu'un couple d'avantages/inconvénients,
  - Allocation manuelle, dans ce cas on obtient l'architecture possédant la latence la plus faible vis-a-vis les ressources autorisées lors de la synthèse,
  - Allocation minimum, obtention de l'architecture la + rapide avec un coût matériel minimum,
  - Allocation maximum, obtention de l'architecture la plus rapide avec un coût matériel maximum,
  - Allocation moyenne, obtention de l'architecture dont le coût moyen doit permettre d'exploiter pleinement le parallélisme contenu dans l'algorithme,
  - Allocation par les forces, obtention de l'architecture dont le coût doit être optimal vis-a-vis de la contrainte de latence spécifiée pour la synthèse,
  - Allocation incrémentale, technique visant à faire succéder les étapes d'allocation et d'ordonnancement de manière dichotomique afin de minimiser une métrique,

- Lors de la compilation logicielle, l'étape d'allocation n'existe pas... Le compilateur fait avec les ressources disponibles sur le µ-processeur !

Techniques d'allocation des ressources



Alloc. mini

Dans cette application nous avons 2 types d'opérations : (+) et (x)

=> Un opérateur de chaque type dans le circuit final

Alloc. maxi

Dans cette application nous avons 2 types d'opérations : (+) et (x)

=> 4 opérateurs (x) et 2 opérateurs de type (+)

Alloc. moyenne

Dans cette application nous avons 2 types d'opérations : (+) et (x)

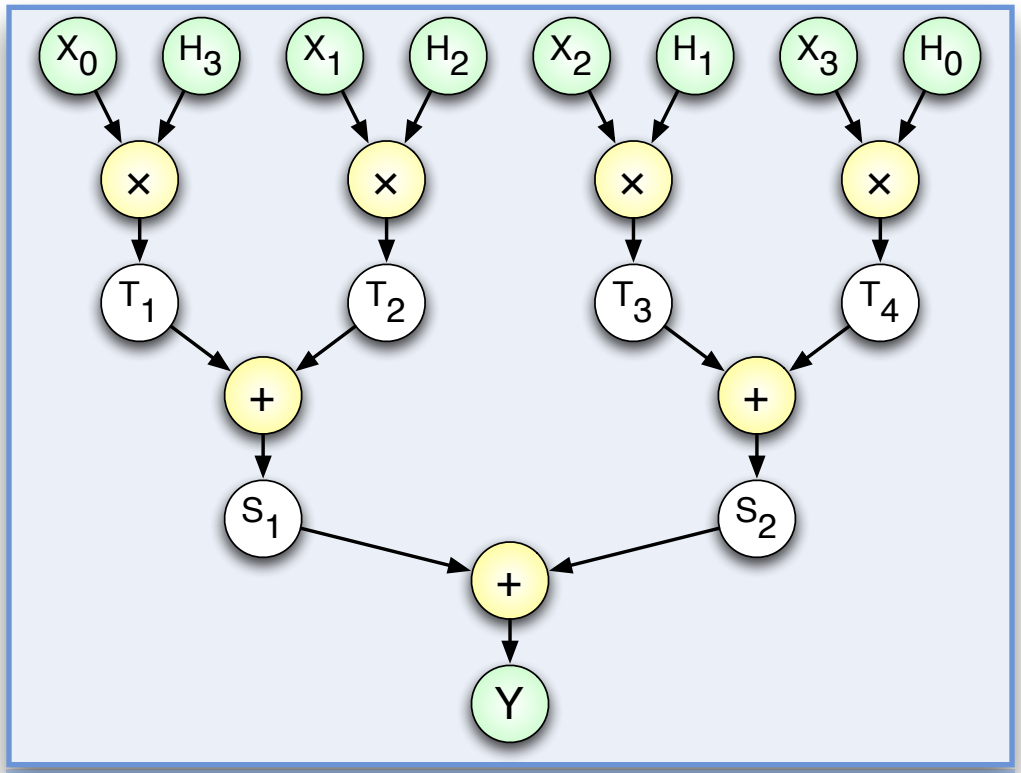
=> Allocation (+) = Nb(+) / latence

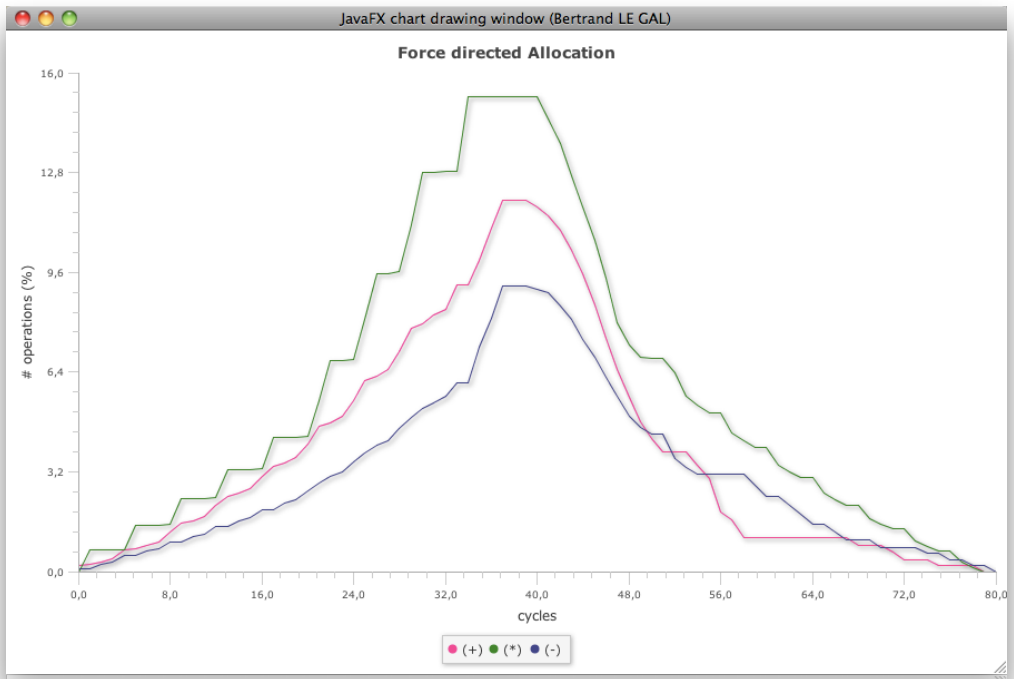
Allocation (x) = Nb(x) / latence

A.N. : latence = 6 => 1.(x) et 1.(+)

latence = 3 => 2.(x) et 1.(+)

Allocation orientée par les forces - FDS

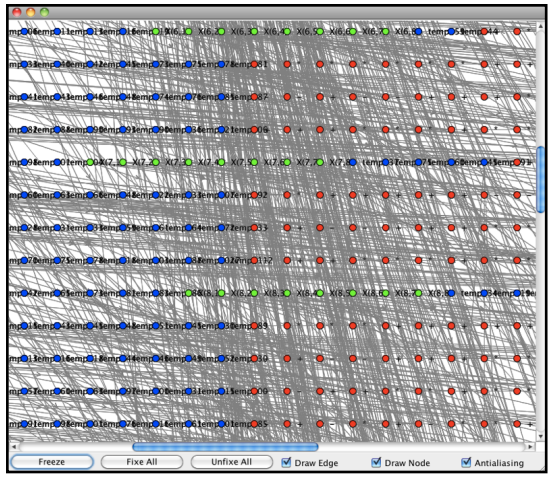




Calcul de la probabilité d'exécution d'une opération par cycle d'horloge.

En fonction de la somme des probabilités on va allouer plus ou moins de ressources

Exemple de la 2d-DCT 8x8 (Allocation)



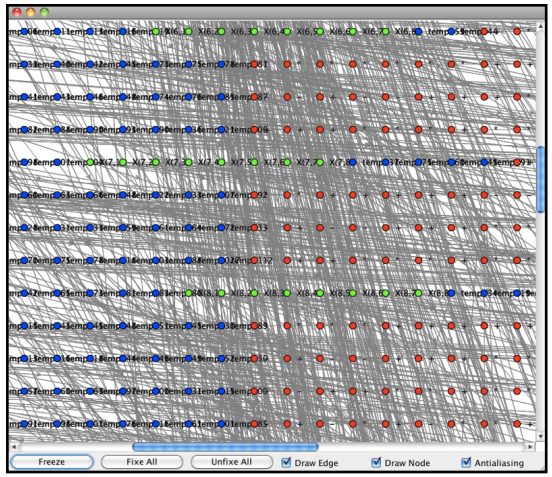
Allocation minimum :

- 1 ressource (+)
- 1 ressource (-)
- 1 ressource (*)

Allocation maximum :

- 32 ressource (+)
- 64 ressource (-)
- 40 ressource (*)

Exemple de la 2d-DCT 8x8 (Allocation)



Allocation minimum :

- 1 ressource (+)
- 1 ressource (-)
- 1 ressource (*)

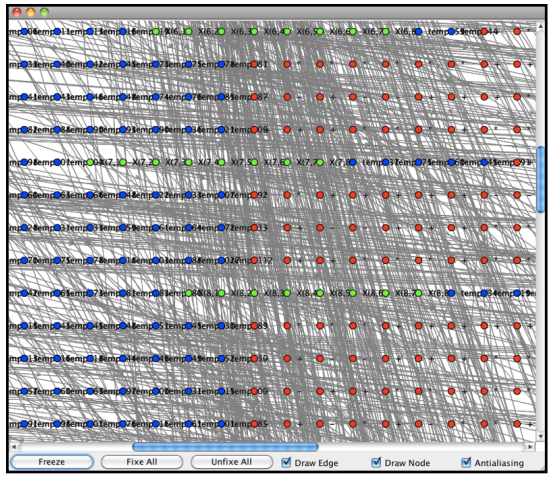
Allocation maximum :

- 32 ressource (+)
- 64 ressource (-)
- 40 ressource (*)

Allocation moyenne :

- 3 ressource (+)
- 3 ressource (-)
- 4 ressource (*)

Exemple de la 2d-DCT 8x8 (Allocation)



Allocation minimum :

- 1 ressource (+)
- 1 ressource (-)
- 1 ressource (*)

Allocation maximum :

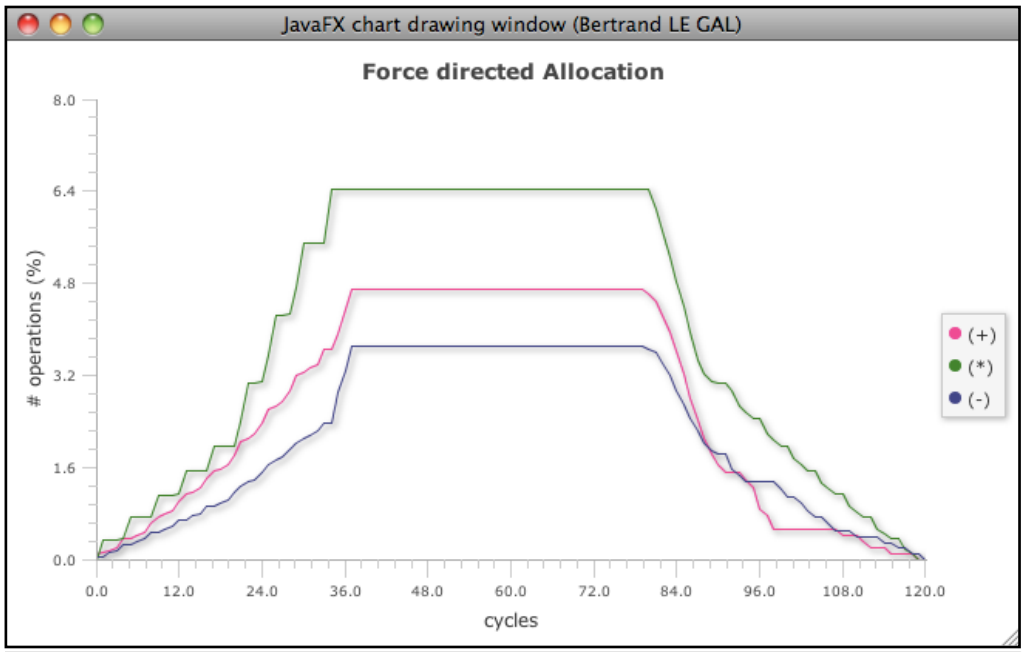
- 32 ressource (+)
- 64 ressource (-)
- 40 ressource (*)

Allocation moyenne :

- 3 ressource (+)
- 3 ressource (-)
- 4 ressource (*)

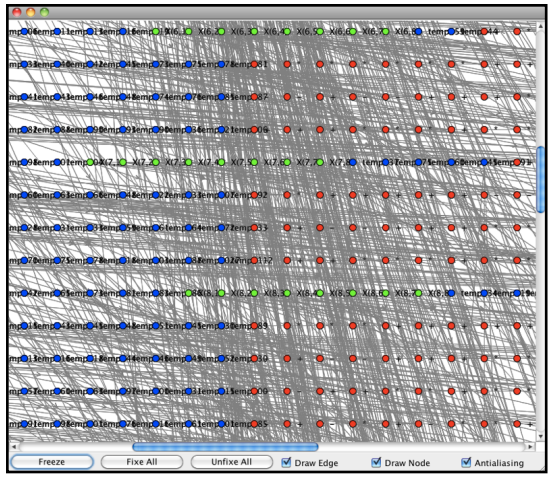
Allocation force directed :

- 5 ressource (+)
- 4 ressource (-)
- 7 ressource (*)



2d-DCT 8x8 : 120 cycles de latence

Exemple de la 2d-DCT 8x8 (Allocation)



Allocation minimum :

- 1 ressource (+)
- 1 ressource (-)
- 1 ressource (*)

Allocation maximum :

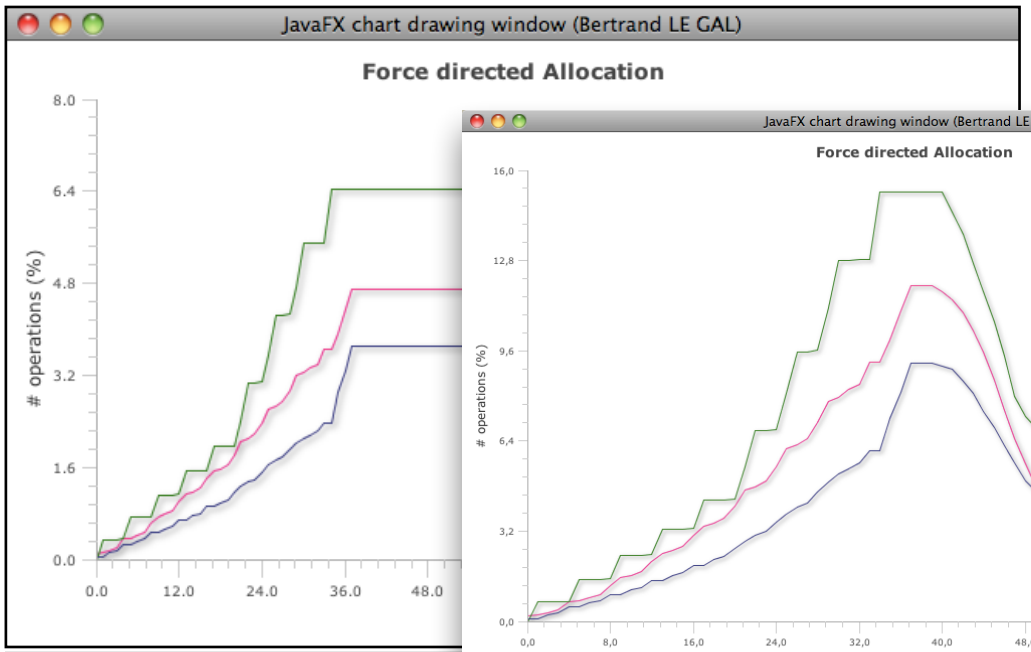
- 32 ressource (+)
- 64 ressource (-)
- 40 ressource (*)

Allocation moyenne :

- 3 ressource (+)
- 3 ressource (-)
- 4 ressource (*)

Allocation force directed :

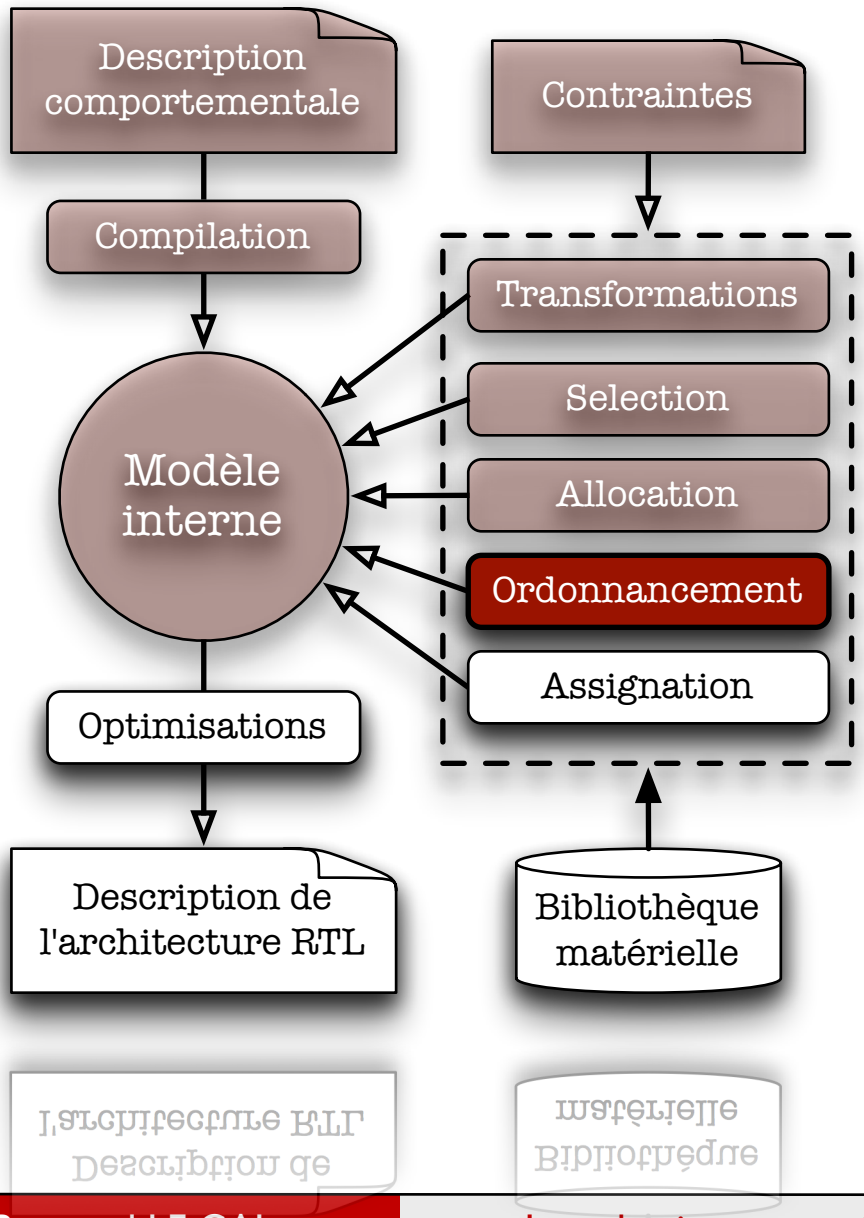
- 5 ressource (+)
- 4 ressource (-)
- 7 ressource (*)



2d-DCT 8x8 : 1

2d-DCT 8x8 : 80 cycles de latence

Ordonnancement des opérations



On connaît le nombre de ressources matérielles de chaque type dont nous disposons (peut évoluer)

Quand les calculs doivent ils être réalisé dans le temps impartit ?

=> Contrainte temporelle implique un délai à ne pas violer !

=> Contrainte matérielle implique une latence minimum du traitement !

Ordonnancement des opérations

- Définition : l'ordonnancement est l'étape durant laquelle on affecte à chaque opération du graphe une date d'exécution.
- Calcul pour chacune des opérations et des variables du graphe d'une date d'exécution,
- Cette étape sera effectuée différemment en fonction des contraintes de synthèse :
  - Ordonnancement temps contraint (latence ou de cadence),
  - Ordonnancement matériel contraint (surface),

- ***Approches locales***
  - ASAP / ALAP Scheduling,
  - Dynamic List Scheduling O(n^2),
  - Static List Scheduling O(n),
- ***Approches gloables***
  - Force Directed Scheduling O(n^3)
  - ILP based scheduling O(e^n)
  - Approches transformationnelles (post-ordonnancement),

ASAP (As Soon As Possible) / ALAP (As Late As Possible)

- En fonction des contraintes appliquées par le concepteur, les algorithmes d'ordonnancement sont différents.
- Ordonnancement ASAP
  - Cet ordonnancement à pour but d'assigner à chaque opération du graphe la date d'exécution au plus tôt (f(x) dates d'exécution des prédécesseurs de n).
- Ordonnancement ALAP
  - Cet ordonnancement à pour but d'assigner à chaque opération du graphe la date d'exécution au plus tard (f(x) dates d'exécution des successeurs de n).
- Ces techniques d'ordonnancement peuvent être utilisées :
  - Seules afin de créer un circuit,
  - Conjointement avec les méthodes que nous verrons après afin d'obtenir des informations sur les noeuds du modèle.

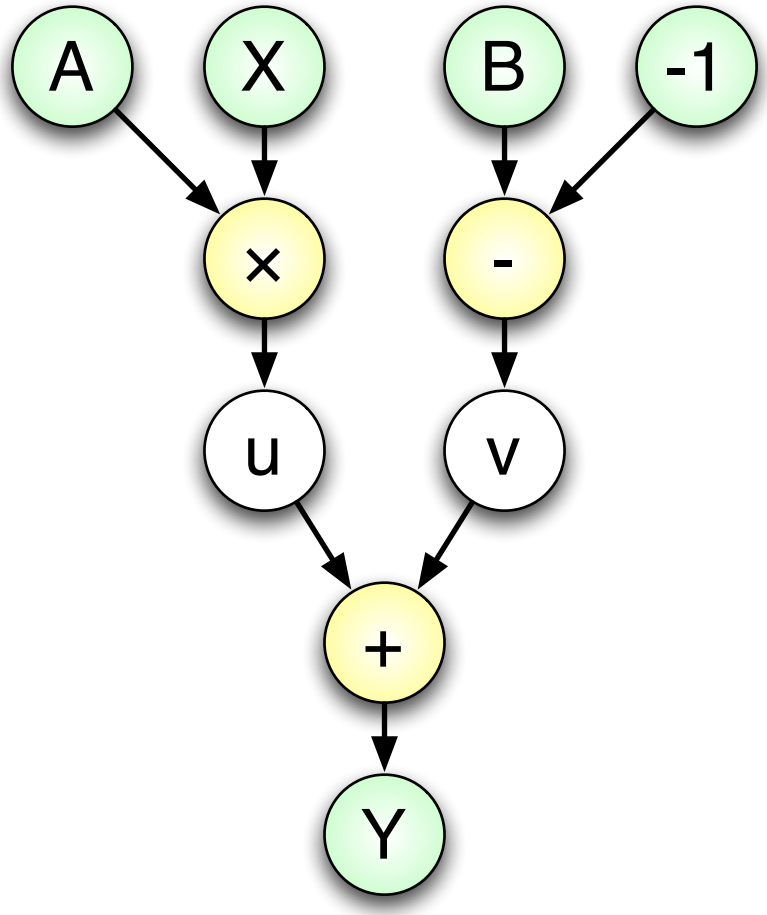
Algorithme ASAP - Implémentation classique

Algorithme ASAP()

```
N = ListeNoeuds( G )

Pour tous les noeuds ni de N faire
    Si Pred(n) != Null alors
        Ei = DateDe(ni)
        N  = N - {ni}
    Sinon
        Ei = 0
    Fin si
Fin pour

Tant que N != Vide faire
    Pour tous les noeuds ni dans N faire
        Si NoeudsOrdonnancés(Pred(ni)) = vrai alors
            Ei = MaxDate(Pred(ni)) + TempsCalcul(ni)
            N  = N - {ni}
        Fin si
    Fin pour
Fin tant que
```

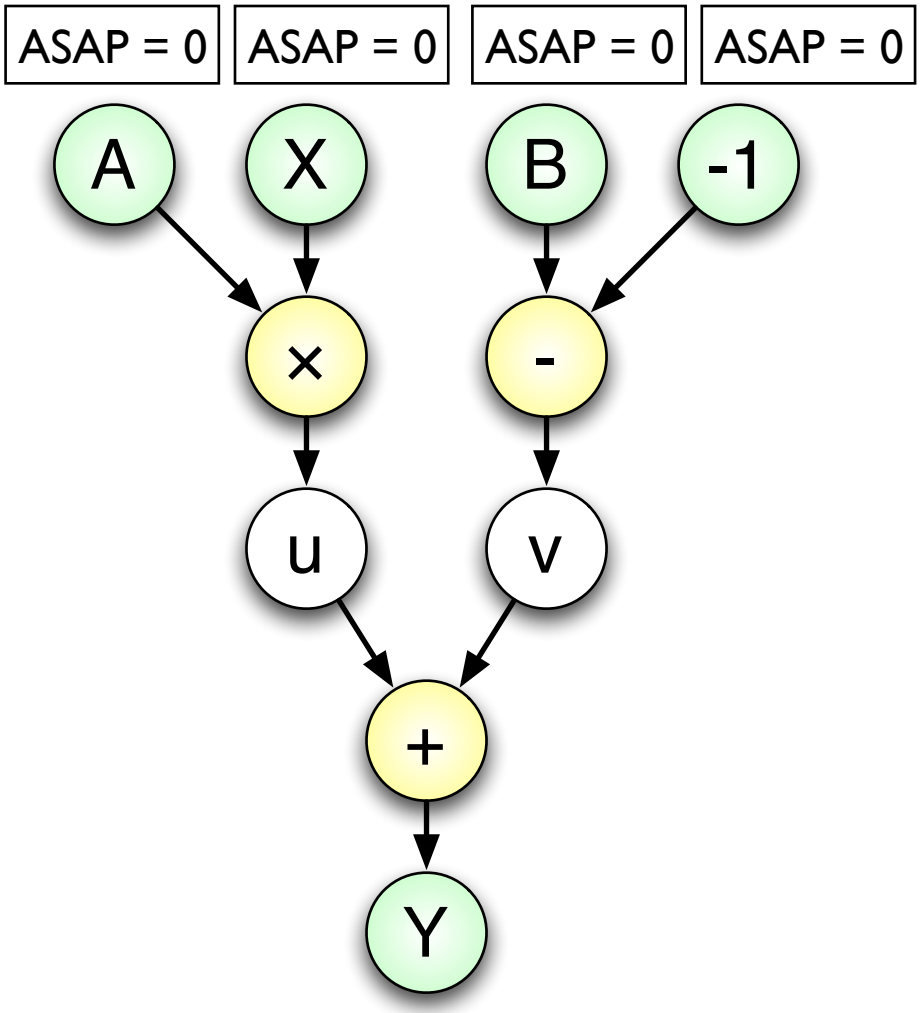
Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,



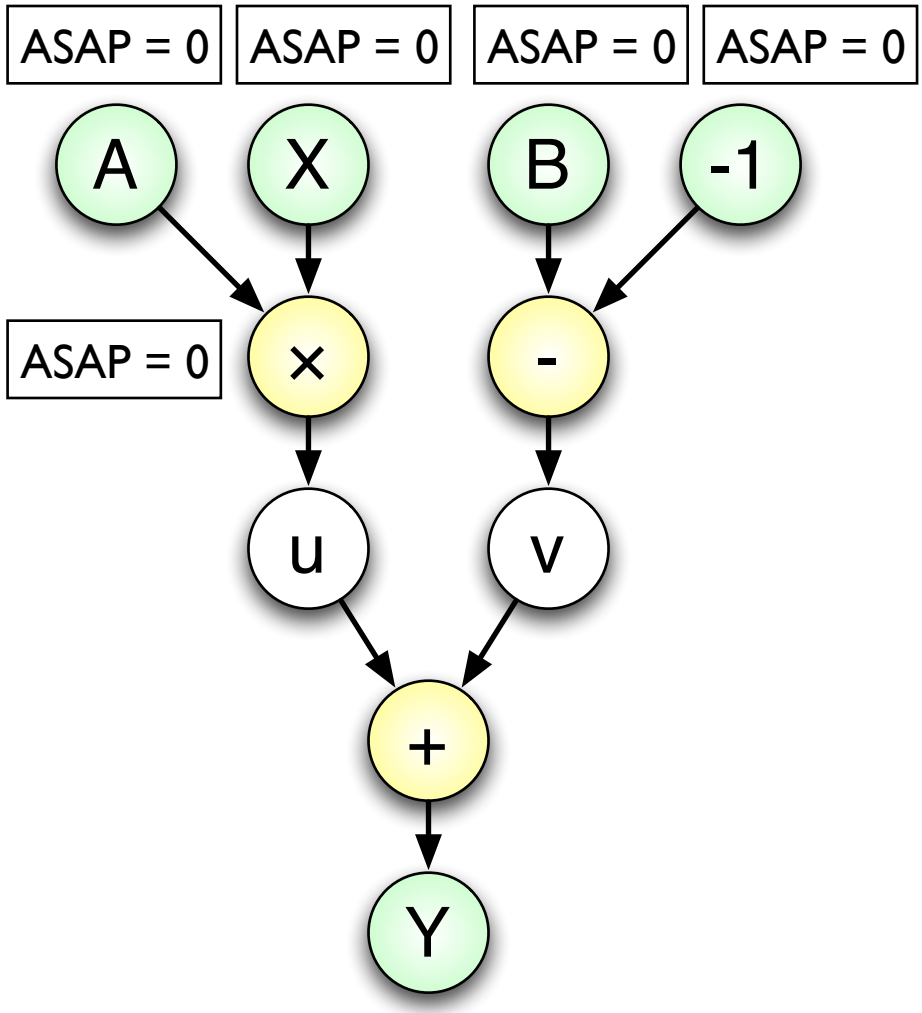
Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,



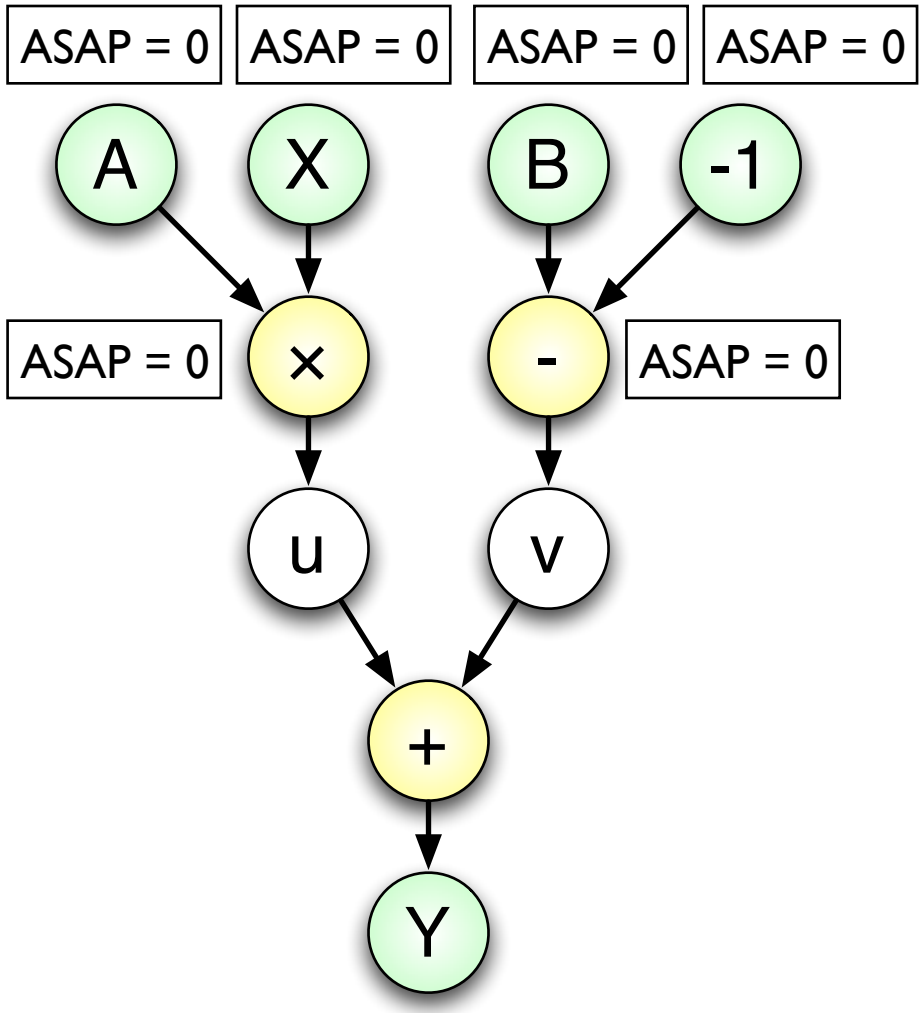
Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,



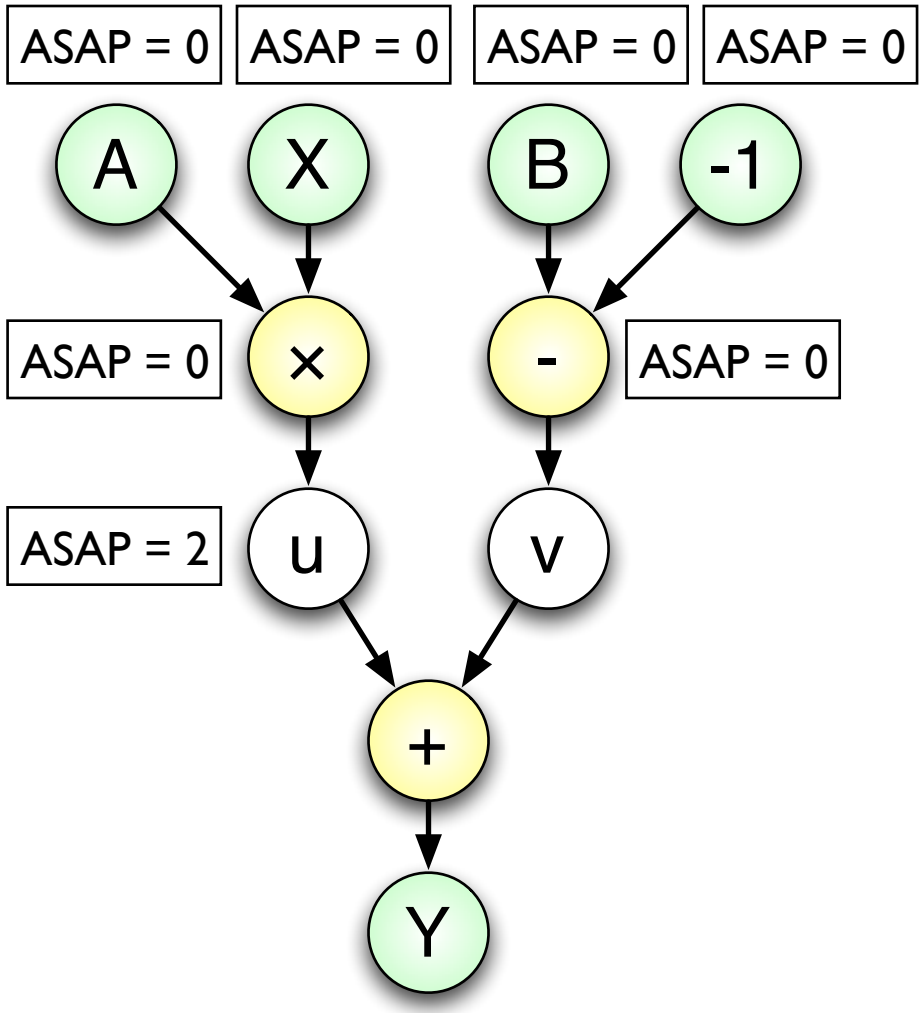
Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,



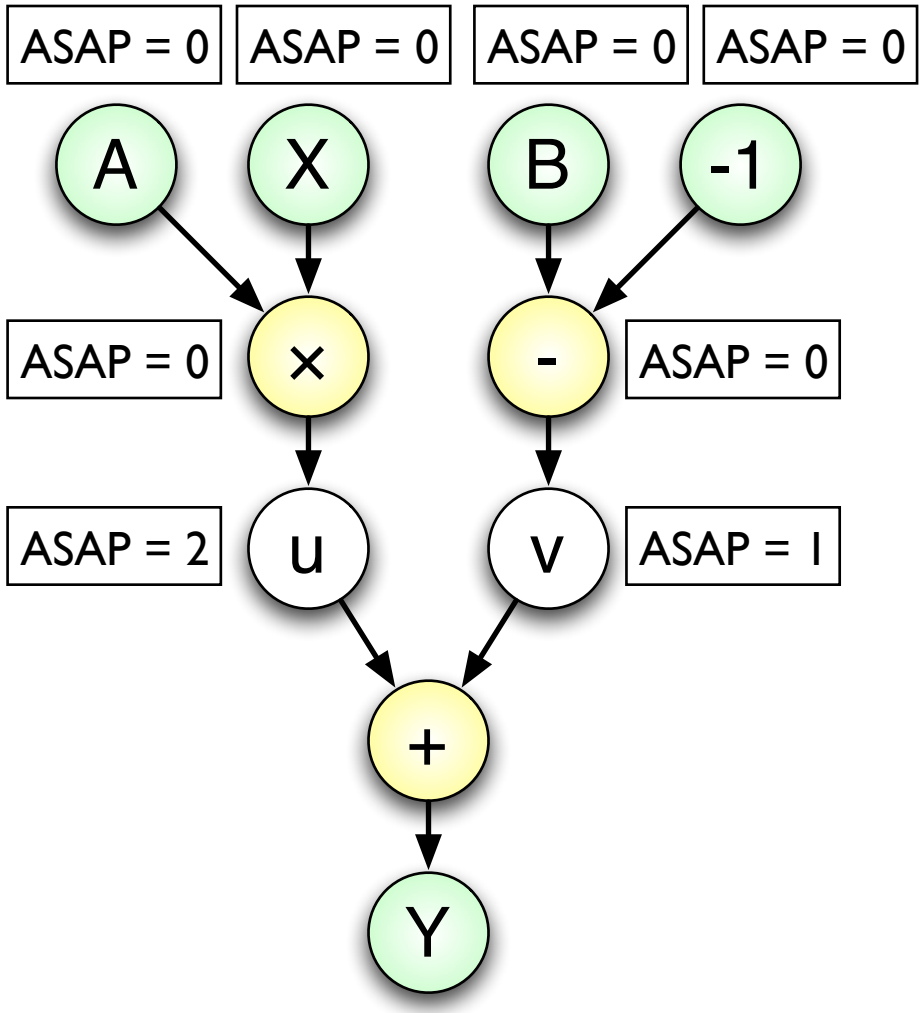
Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,



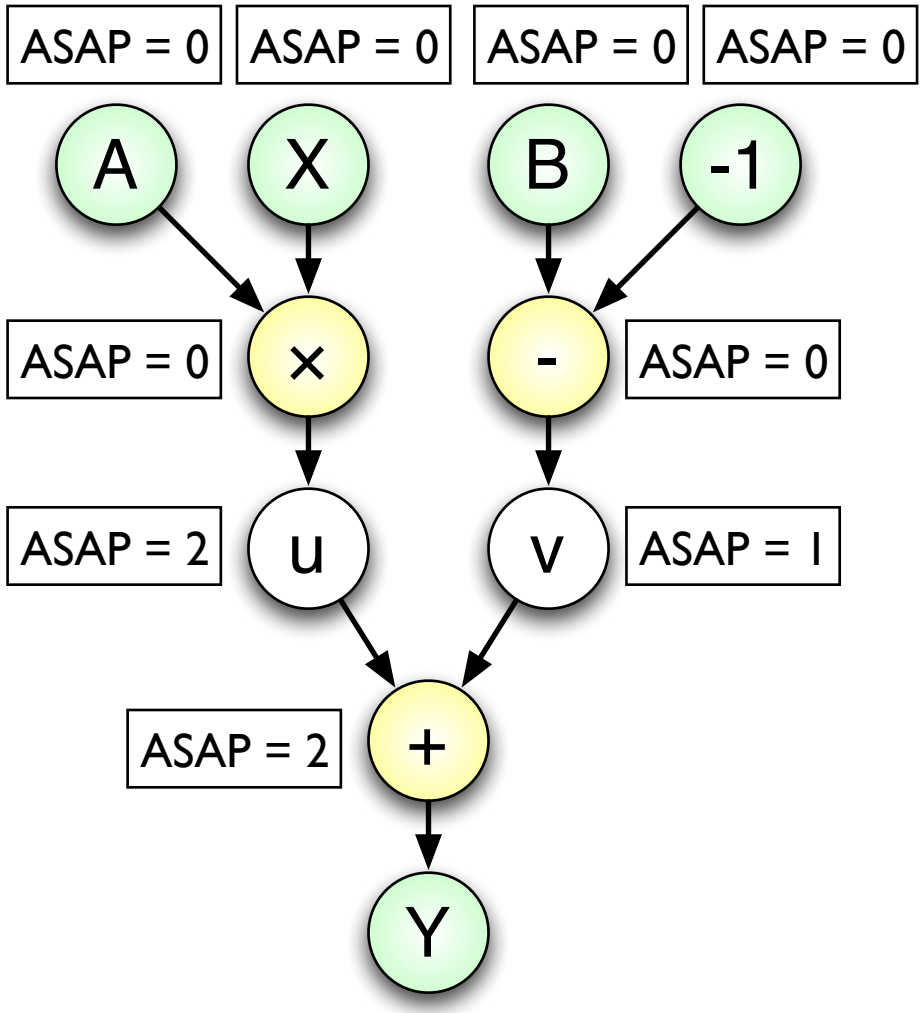
Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,



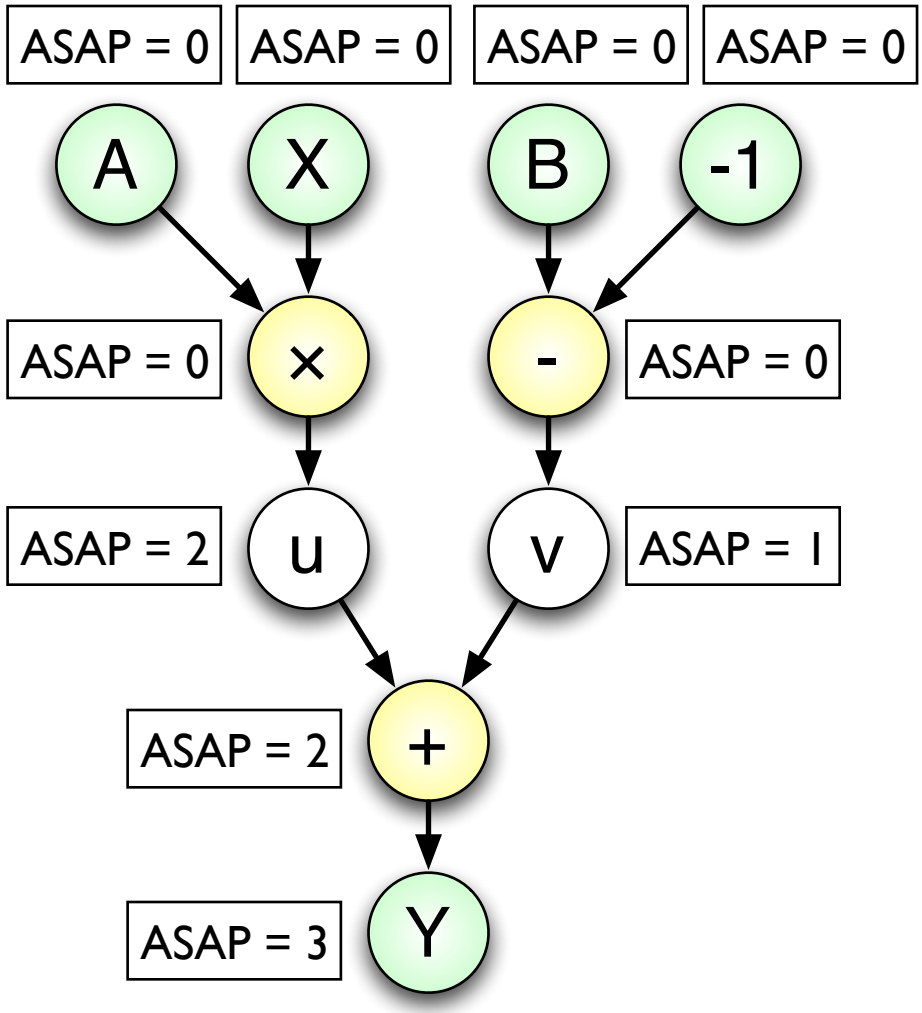
Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,

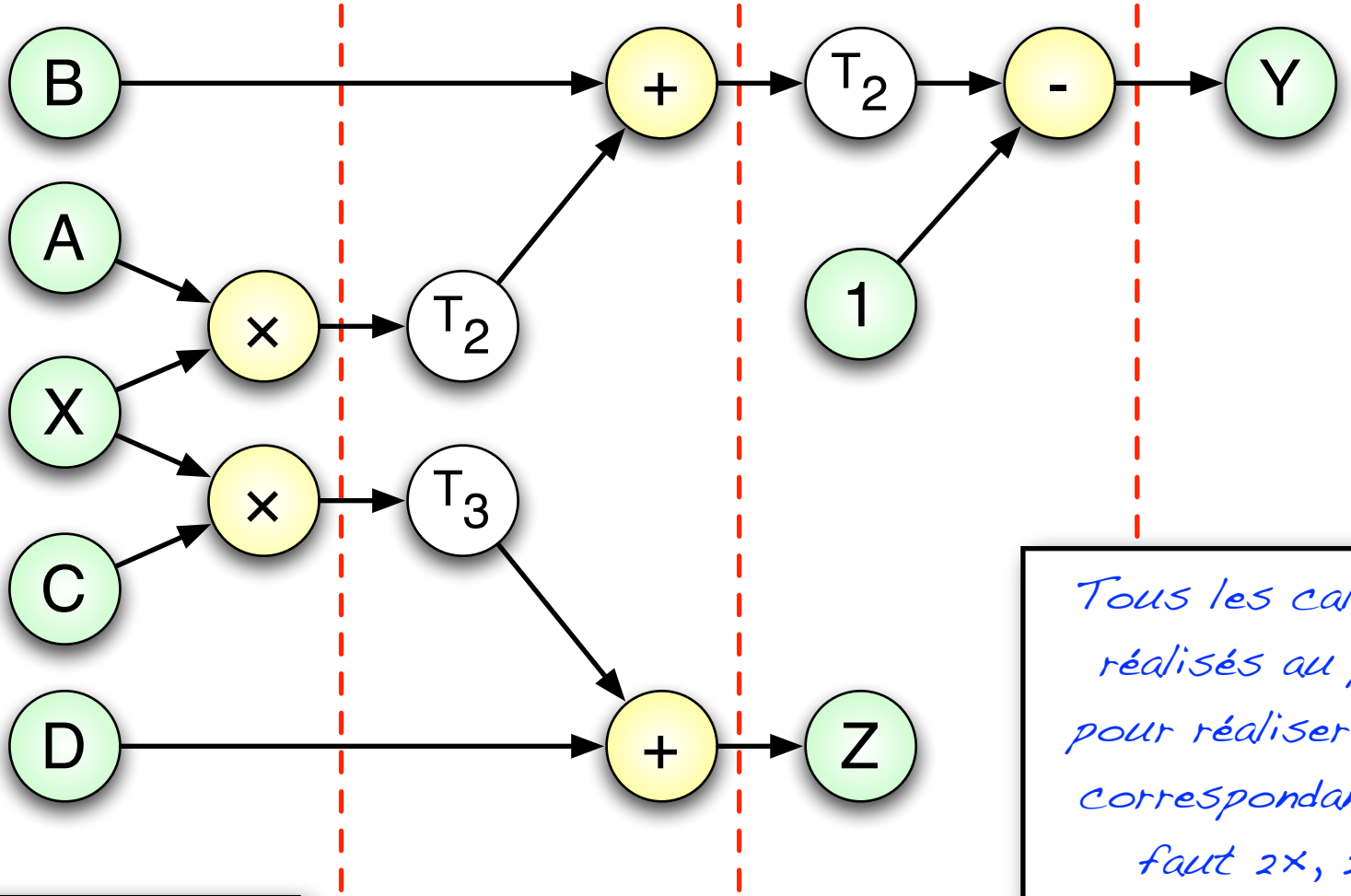


Exemple d'ordonnancement ASAP

- Dans le cadre de cet exemple d'ordonnancement ASAP nous allons faire les hypothèses suivantes :
  - Date de début = 0
  - T(×) = 2 cycles
  - T(+) = 1 cycle
  - Toutes les entrées sont disponibles dès le début de l'ordonnancement,



Ordonnancement ASAP - Exemple (1)

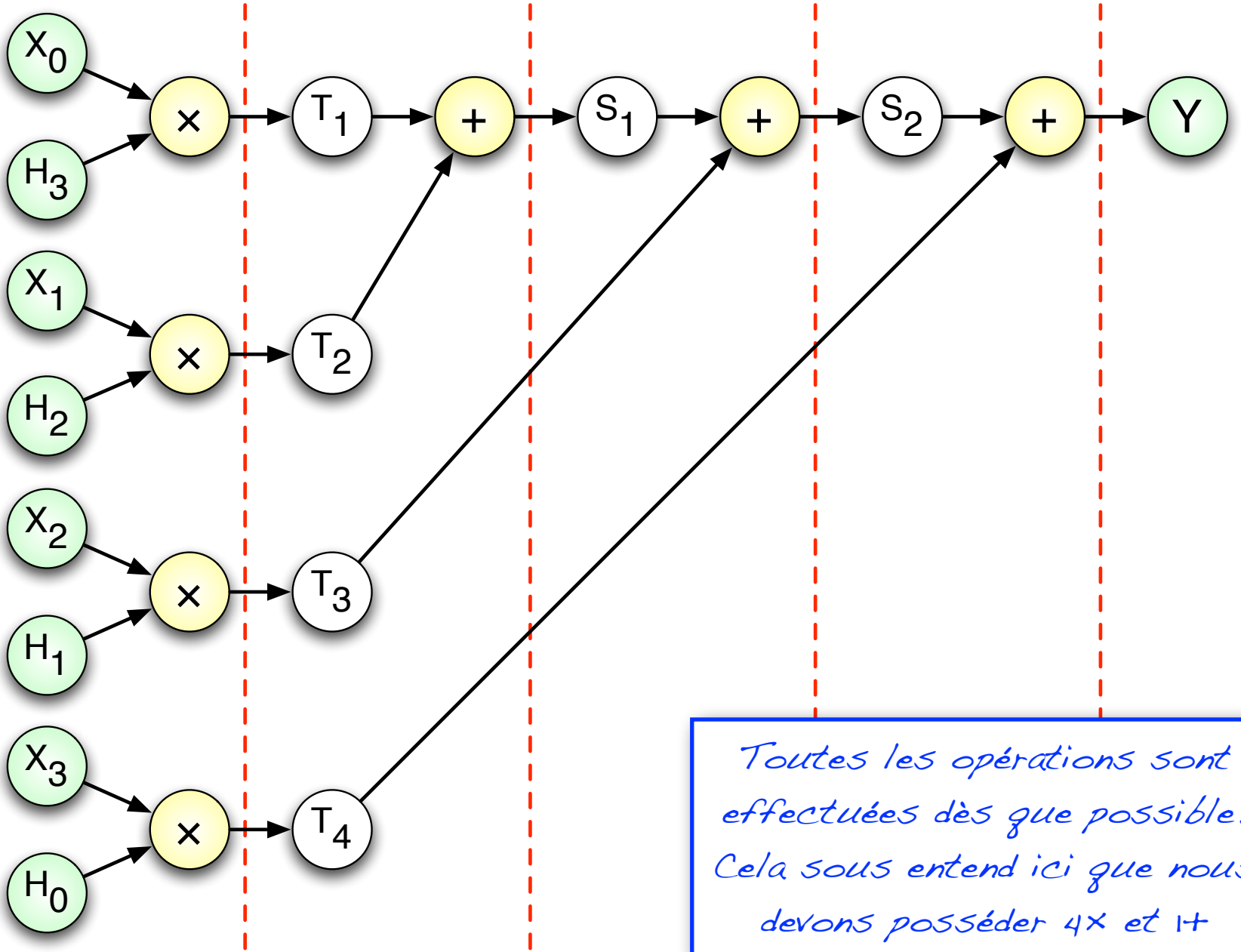


$Y = (A \cdot X) + B - 1$

$Z = (C \cdot X) + D$

Tous les calculs sont réalisés au plus tôt, pour réaliser le circuit correspondant, il nous faut 2×, 2+ et 1-

Ordonnancement ASAP - Exemple (2)



Exemple pédagogique - Ordonnancement ASAP

Décomposez l'algorithme sous forme de graphe puis ordonnancez le à l'aide des dates ASAP. Le temps des opérations (+, -, *) est de 1 cycle.

=> Quelle est la latence minimale de l'application ?

=> Combien de ressources matérielles sont nécessaires ?

$$y = (a - b)^2 + c - 1$$

$$z = c - b - a$$

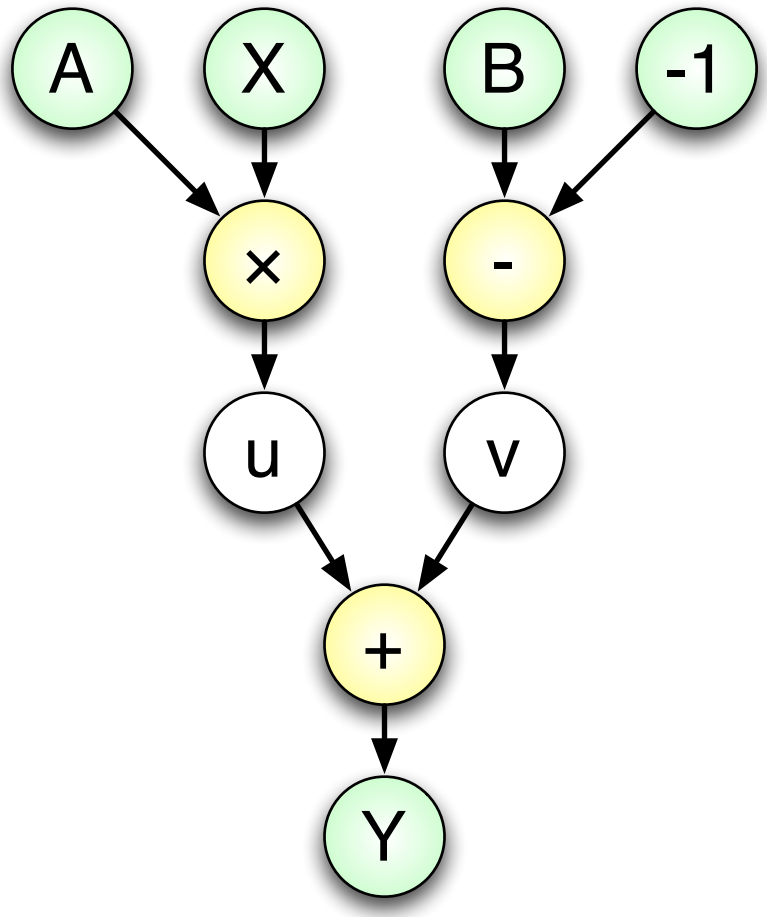
Algorithme ALAP - Implémentation classique

Algorithme ALAP()

```
N = ListeNoeuds( G )

Pour tous les noeuds ni de N faire
    Si Succ(n) != Null alors
        Li = T /* Date ASAP de la sortie du graphe */
        N  = N - {ni}
    Sinon
        Li = 0
    Fin si
Fin pour

Tant que N != Vide faire
    Pour tous les noeuds ni dans N faire
        Si NoeudsOrdonnancés(Succ(ni)) = vrai alors
            Li = MinDate(Succ(ni)) - TempsCalcul(ni)
            N  = N - {ni}
        Fin si
    Fin pour
Fin tant que
```

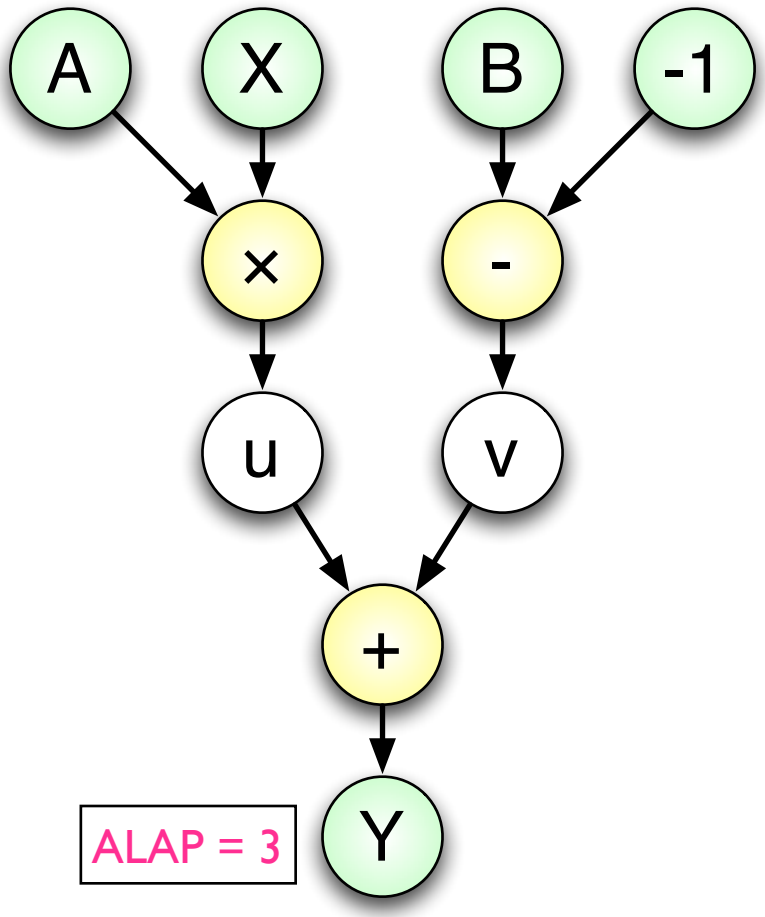
Exemple d'ordonnancement ALAP

- Dans le cadre de cet exemple d'ordonnancement ALAP nous allons faire les hypothèses suivantes :
  - Contrainte temporelle = 3
  - T(×) = 2 cycles
  - T(+) = 1 cycle



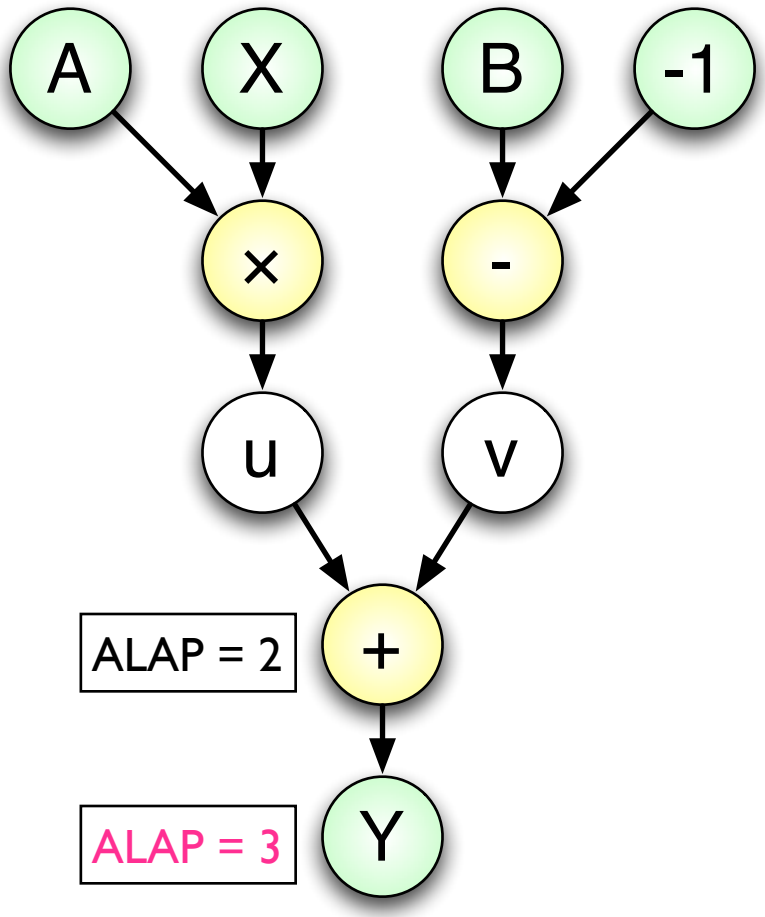
Exemple d'ordonnancement ALAP

- Dans le cadre de cet exemple d'ordonnancement ALAP nous allons faire les hypothèses suivantes :
  - Contrainte temporelle = 3
  - T(×) = 2 cycles
  - T(+) = 1 cycle



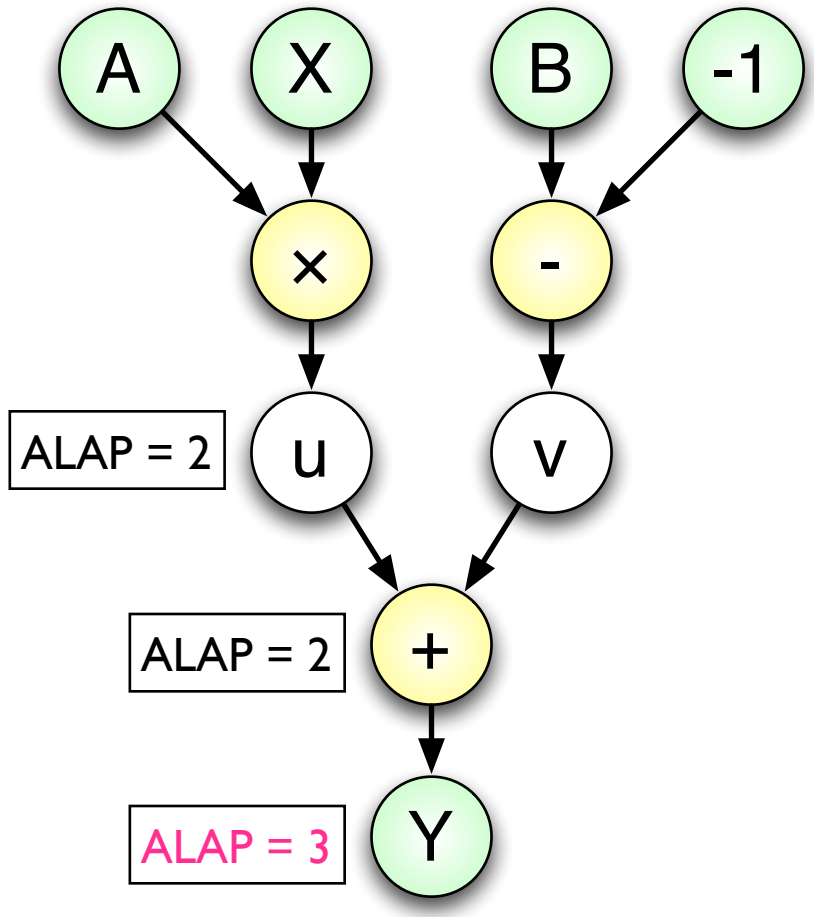
Exemple d'ordonnancement ALAP

- Dans le cadre de cet exemple d'ordonnancement ALAP nous allons faire les hypothèses suivantes :
  - Contrainte temporelle = 3
  - T(×) = 2 cycles
  - T(+) = 1 cycle



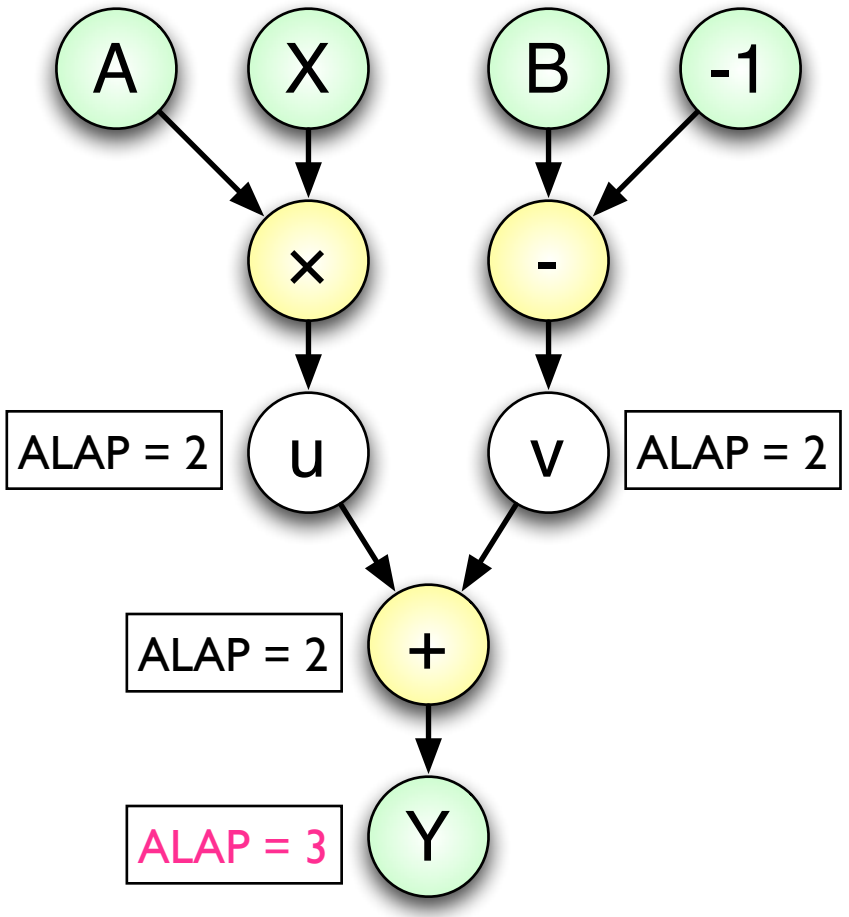
Exemple d'ordonnancement ALAP

- Dans le cadre de cet exemple d'ordonnancement ALAP nous allons faire les hypothèses suivantes :
  - Contrainte temporelle = 3
  - T(×) = 2 cycles
  - T(+) = 1 cycle



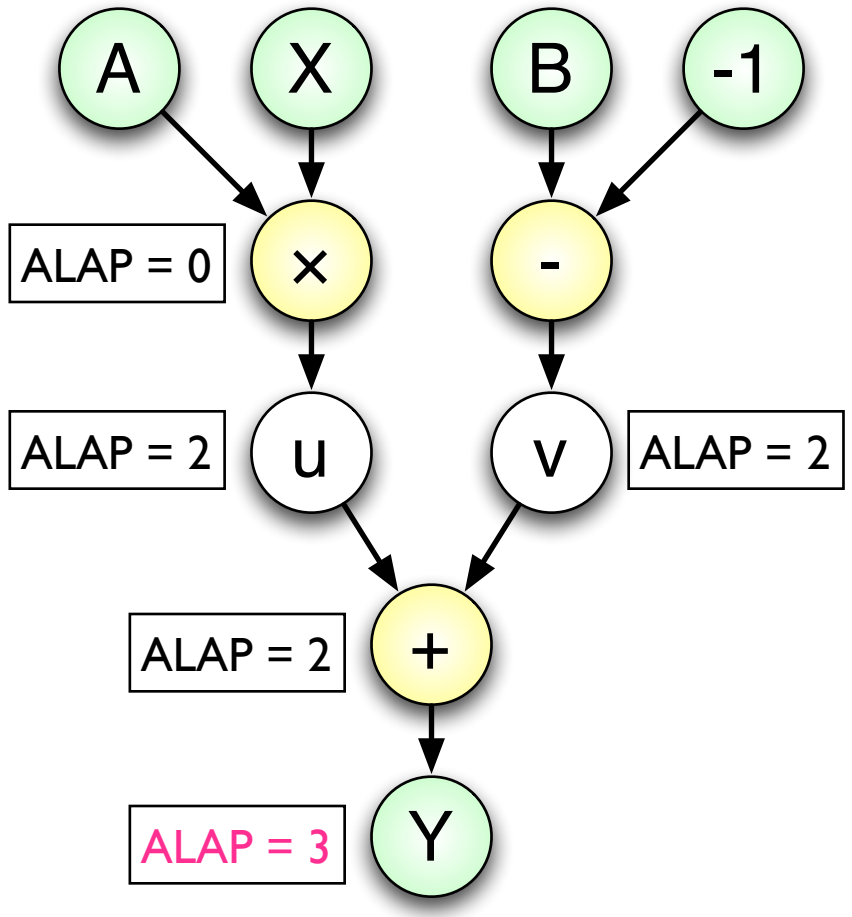
Exemple d'ordonnancement ALAP

- Dans le cadre de cet exemple d'ordonnancement ALAP nous allons faire les hypothèses suivantes :
  - Contrainte temporelle = 3
  - T(×) = 2 cycles
  - T(+) = 1 cycle



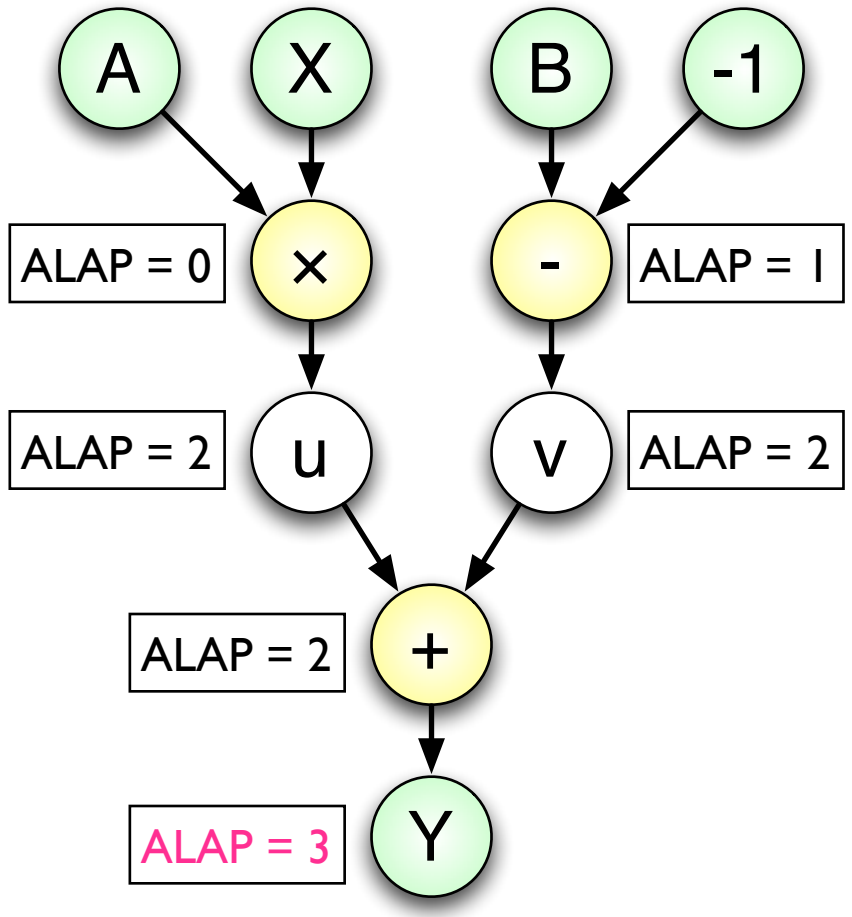
Exemple d'ordonnancement ALAP

- Dans le cadre de cet exemple d'ordonnancement ALAP nous allons faire les hypothèses suivantes :
  - Contrainte temporelle = 3
  - T(×) = 2 cycles
  - T(+) = 1 cycle

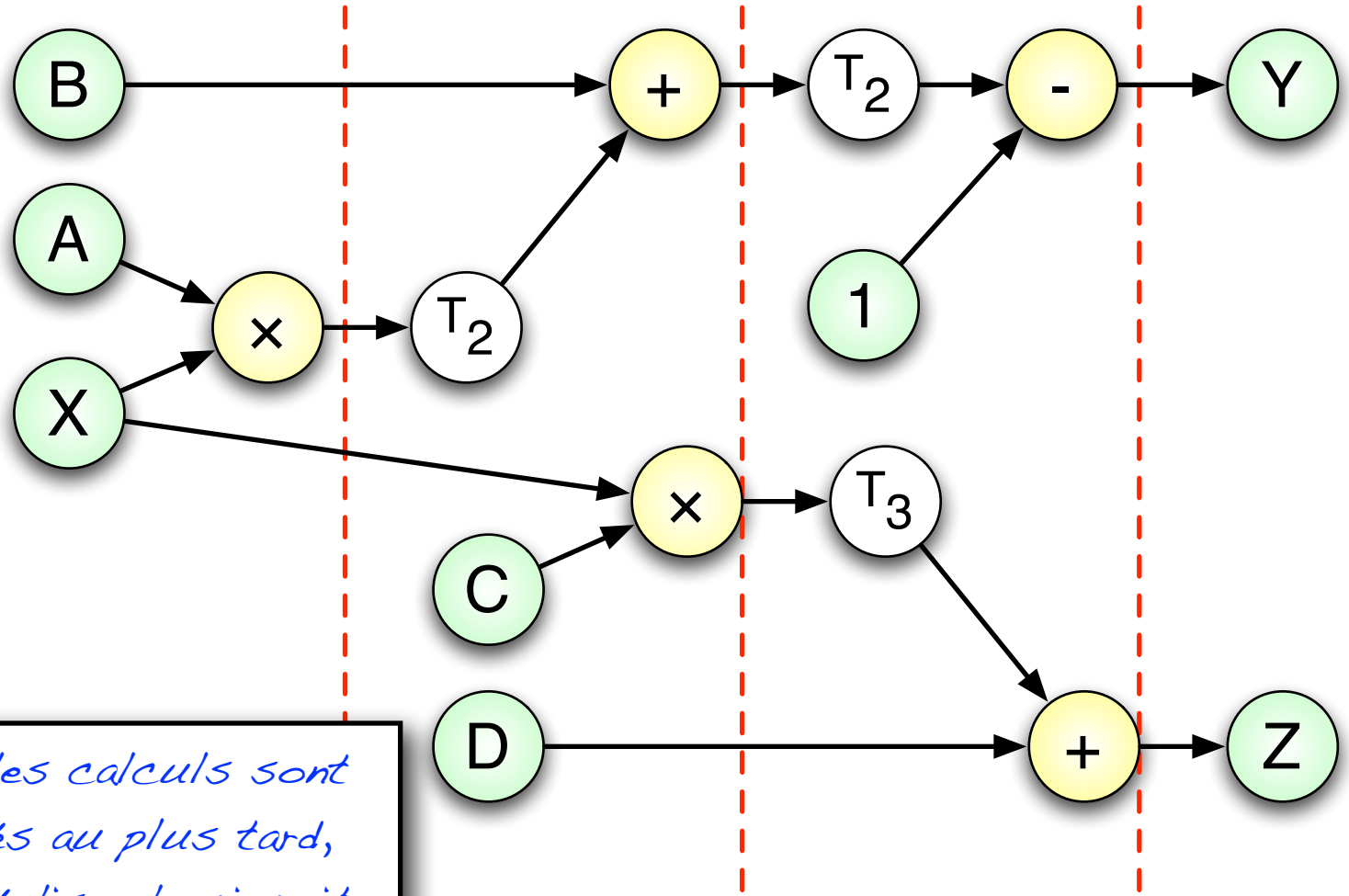


Exemple d'ordonnancement ALAP

- Dans le cadre de cet exemple d'ordonnancement ALAP nous allons faire les hypothèses suivantes :
  - Contrainte temporelle = 3
  - T(×) = 2 cycles
  - T(+) = 1 cycle



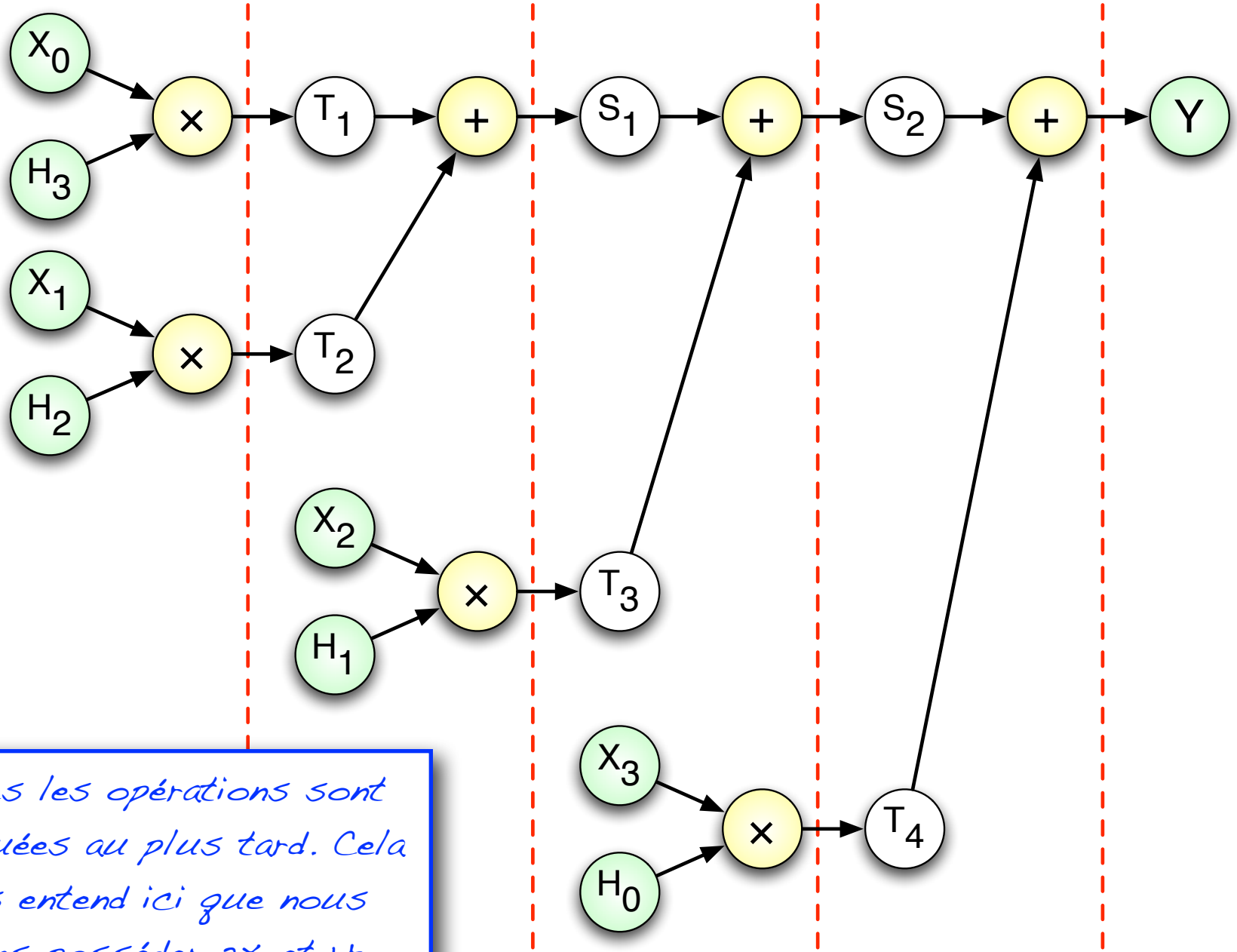
Ordonnancement ALAP - Exemple (1)



Tous les calculs sont réalisés au plus tard, pour réaliser le circuit correspondant, il nous faut 1×, 1+ et 1−

La date au plus tard utilisée correspond à la durée du chemin critique contenu dans le graphe

Ordonnancement ALAP - Exemple (2)



Toutes les opérations sont effectuées au plus tard. Cela sous entend ici que nous devons posséder 2× et 1+

Exemple pédagogique - Ordonnancement ALAP

Décomposez l'algorithme sous forme de graphe puis ordonnancez le à l'aide des dates ALAP. Le temps des opérations (+, -, *) est de 1 cycle.

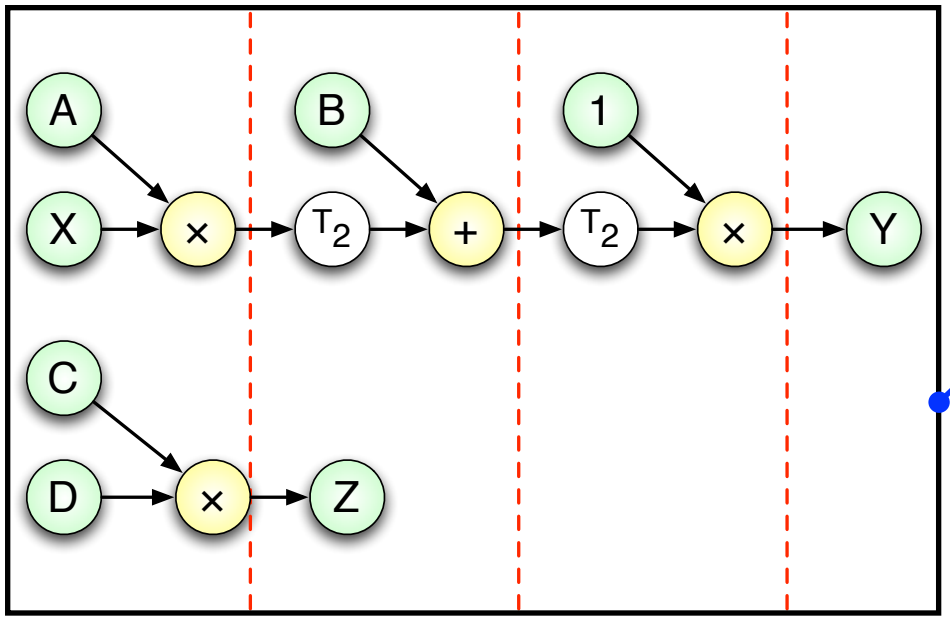
=> Quelle est la latence minimale de l'application ?

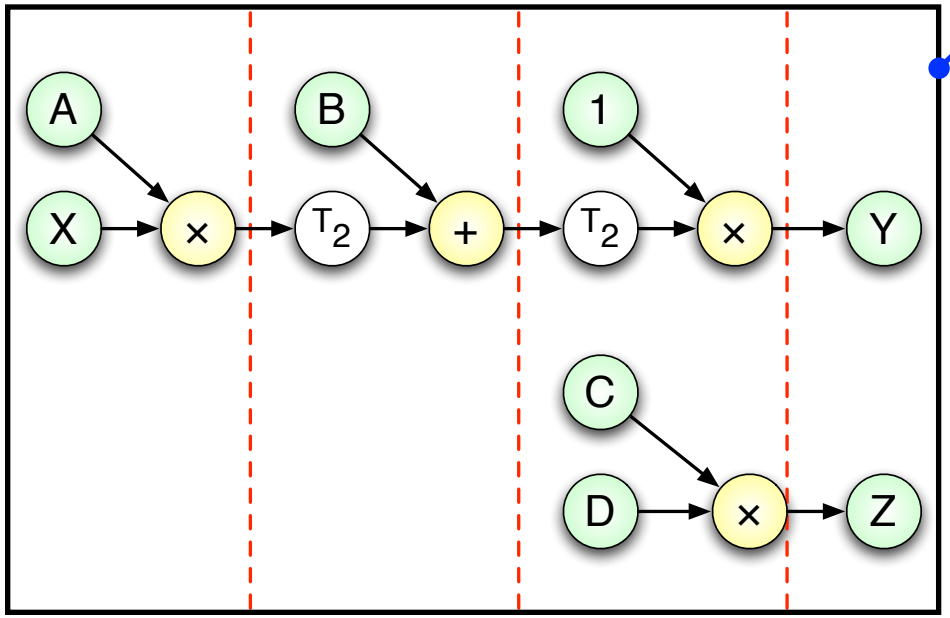
=> Combien de ressources matérielles sont nécessaires ?

$$y = (a - b)^2 + c - 1$$

$$z = c - b - a$$

La mobilité des opérations





Ordonnancement ASAP des opérations

Ordonnancement ALAP des opérations

L'opération Z=C+D est mobile entre sa date ASAP et ALAP. Sa mobilité = ALAP - ASAP

Cette opération peut être librement ordonnancée au cycles (1,2 ou 3). Les ressources matérielles nécessaires seront alors différentes.

Ordonnancement par liste de priorité

- A partir de ces dates ASAP/ALAP, on peut extraire de nouvelles informations :

$$Mobilite = ALAP(n_i) - ASAP(n_i)$$

$$Urgence = ALAP(v_n) - ALAP(v_i)$$

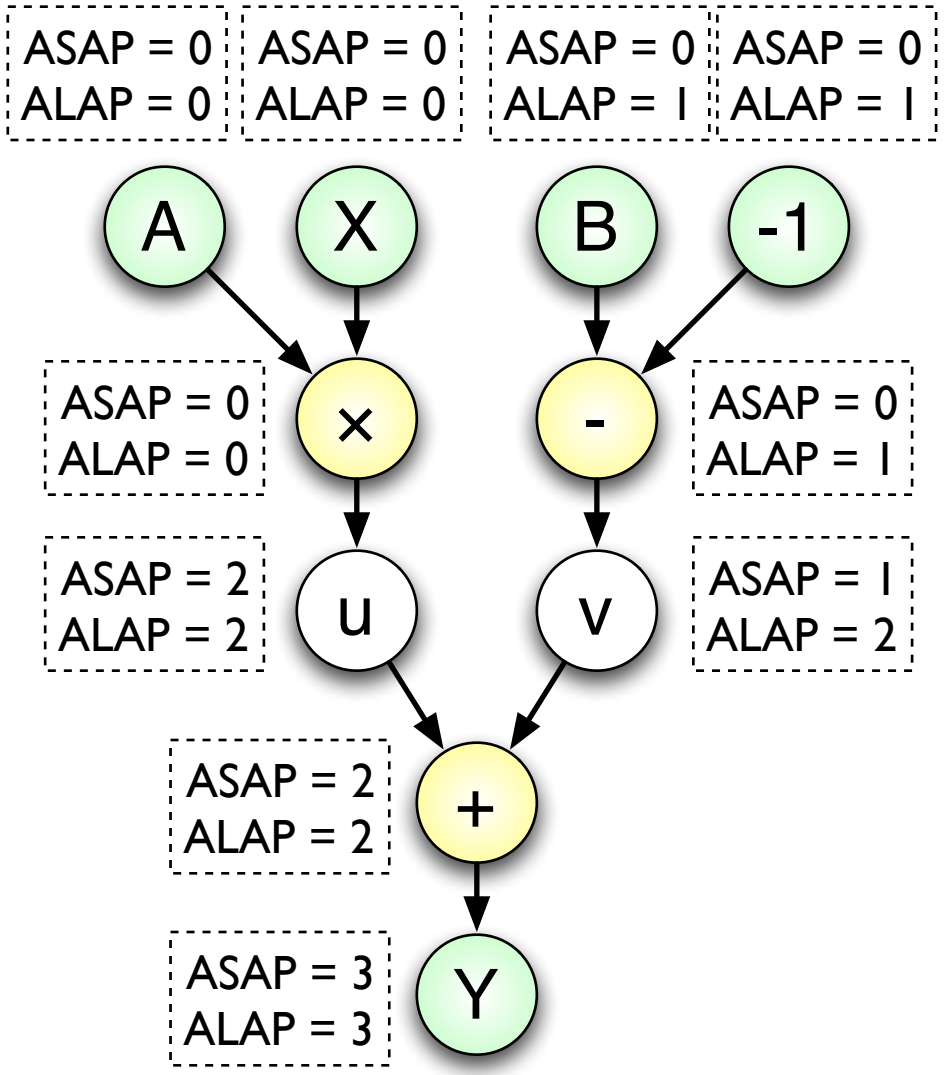
- Afin d'améliorer la qualité de l'ordonnancement, on va utiliser des techniques basées sur des listes :
  - On va placer toutes les opérations à réaliser dans une liste,
  - On va trier les opérations dans la liste en fonction d'un critère de priorité,
  - On va sélectionner les opérations prêtes à être ordonnancés dans la liste sous réserve d'avoir assez de ressources matérielles disponibles,
  - On libère l'ensemble des ressources et on incrémente le compteur de cycles d'horloge écoulés + mise à jour des opérations éligibles,
  - On réitère (3) tant que la liste n'est pas vide.

Produit de matrice (8×8) => 2000 noeuds, produit de matrice (16×16) => 15500 noeuds

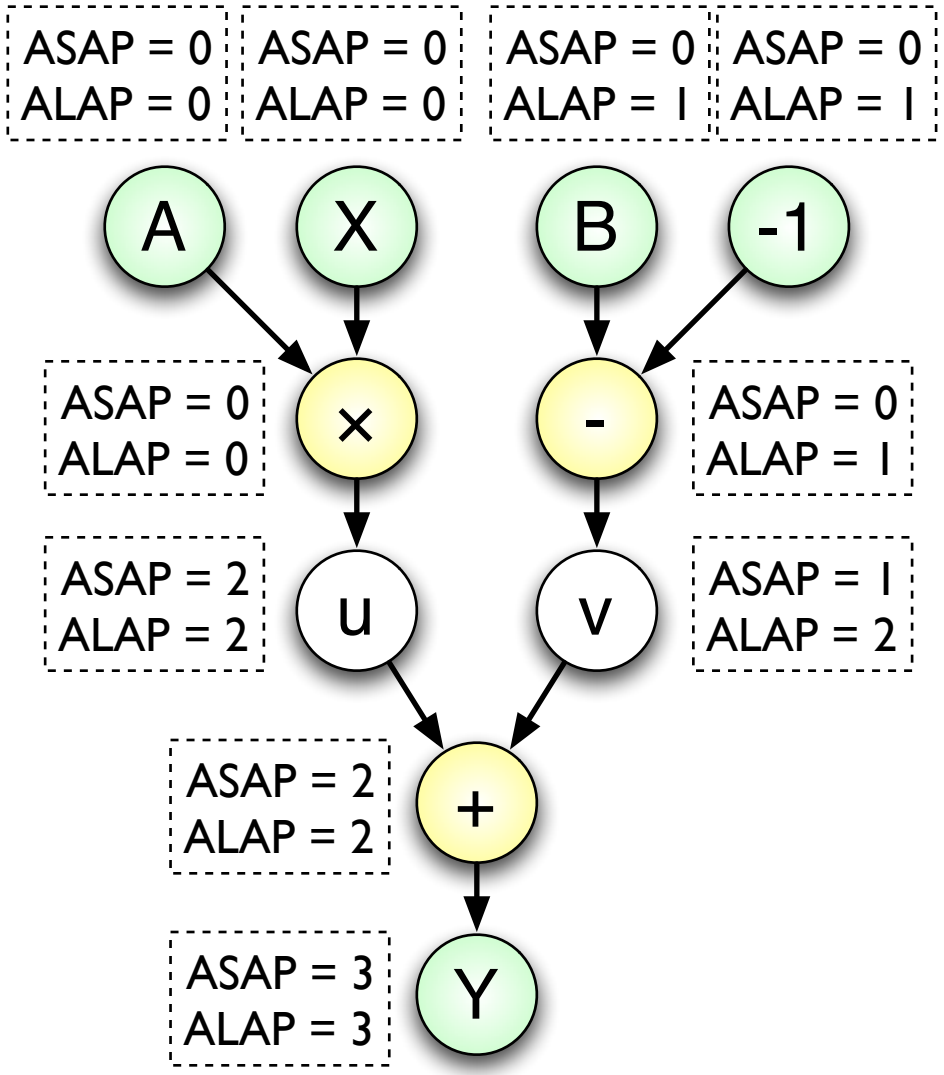
Ordonnancement par liste - Surface contrainte

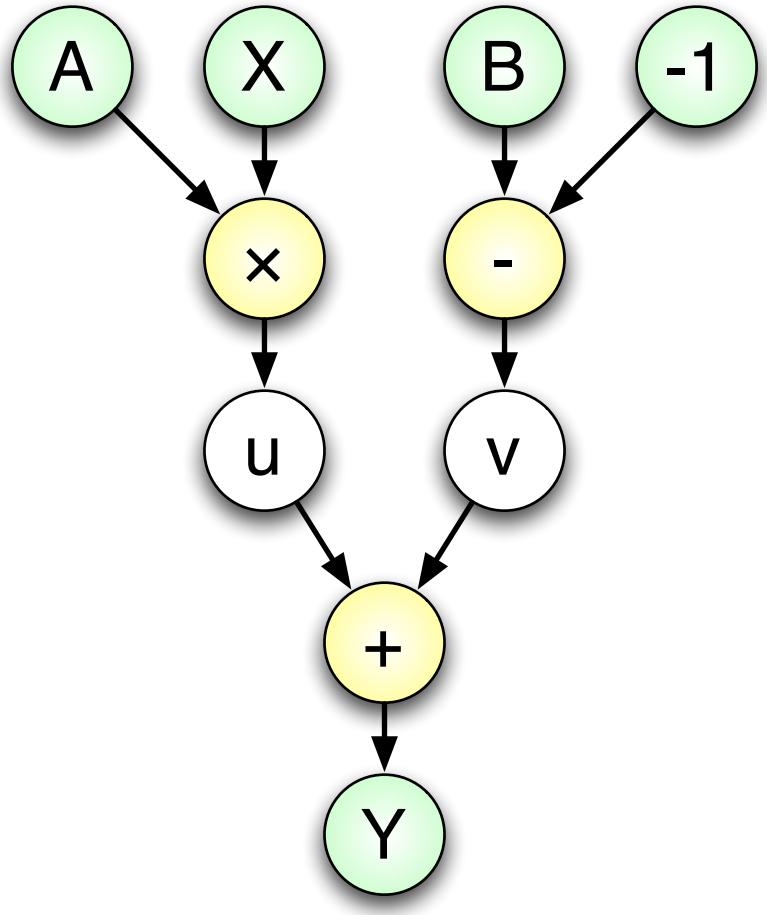
- Il existe plusieurs variantes d'ordonnancement par liste
  - Liste statique : la liste des opérations à exécuter est triée uniquement au départ de l'ordonnancement à l'aide d'une fonction de coût,
  - Liste dynamique : la liste des opérations à exécuter est triée a chaque cycle de l'ordonnancement afin de pouvoir prendre en considération les choix effectués,
- Dans les 2 cas, les décisions prises ne sont jamais remise en cause par la suite => réduction de la complexité de l'algorithmique,
  - Ces 2 variantes fourniront des résultats différents car la liste est adapté à chaque décision (dynamique),
- La complexité de l'algorithme à de l'importance, cela Impacte directement sur le temps de calcul de l'outil,
  - Produit de matrice (8×8) est de 2000 noeuds
  - Produit de matrice (16×16) est de 15500 noeuds

Calcul de l'urgence et de la mobilité des noeuds

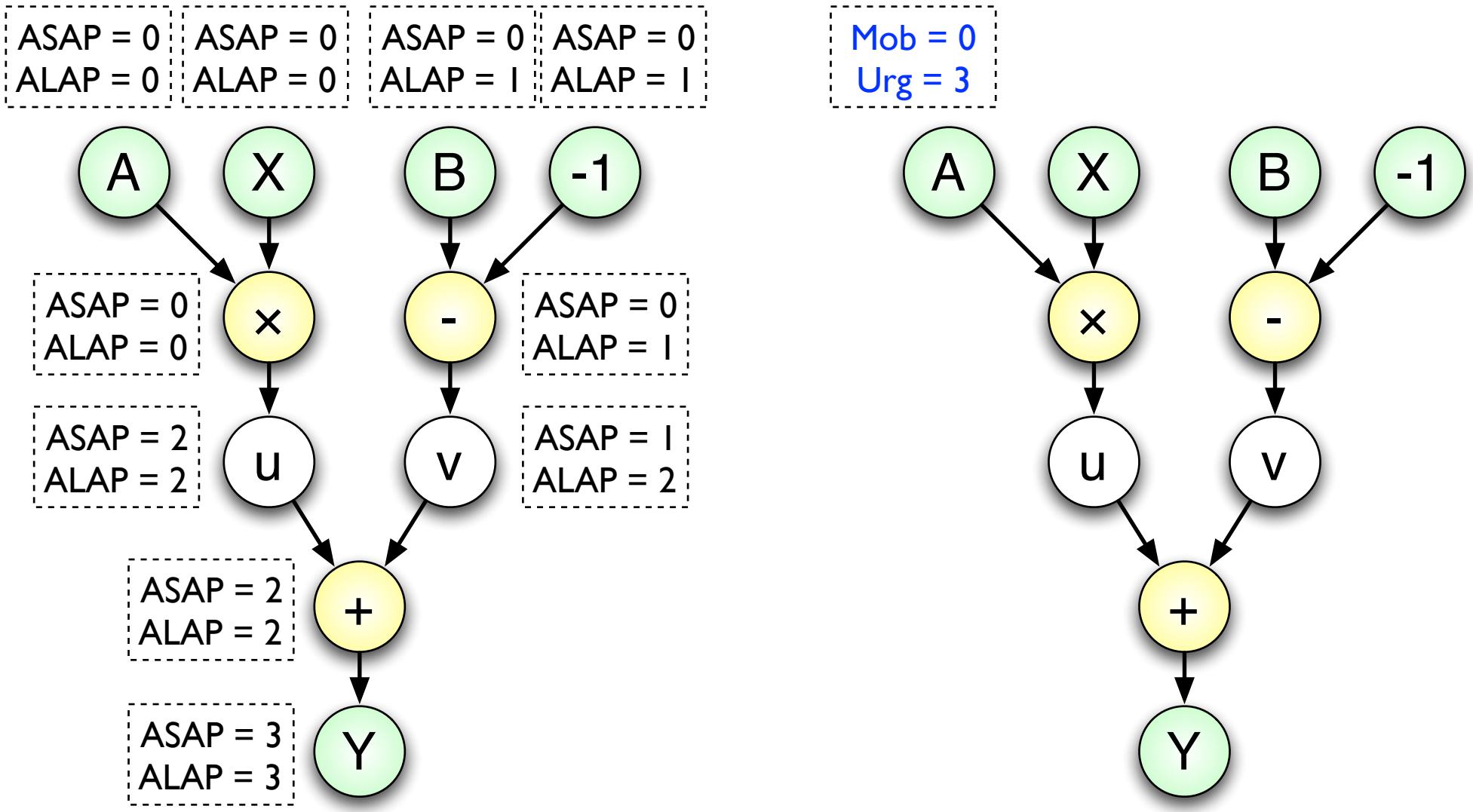


Calcul de l'urgence et de la mobilité des noeuds

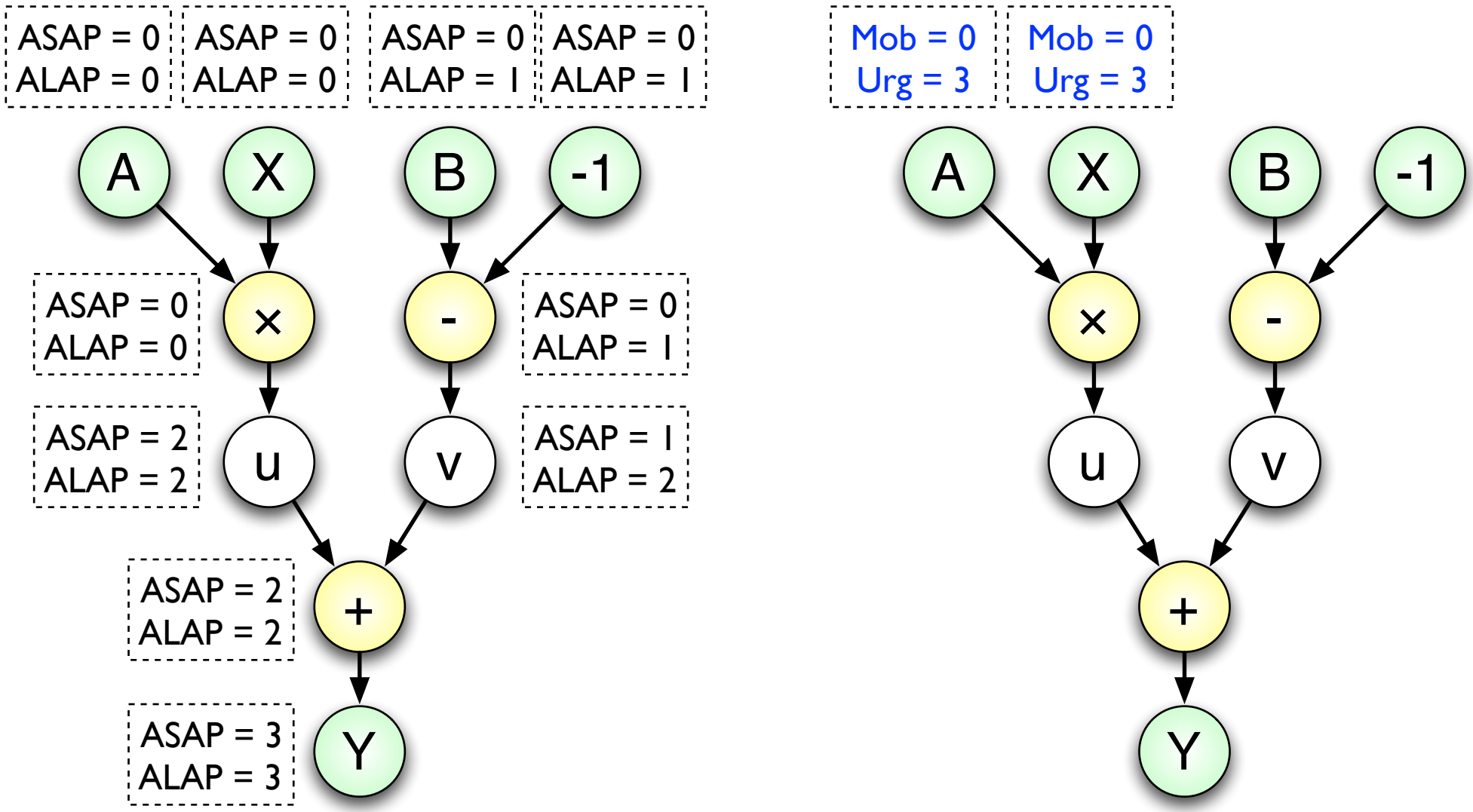




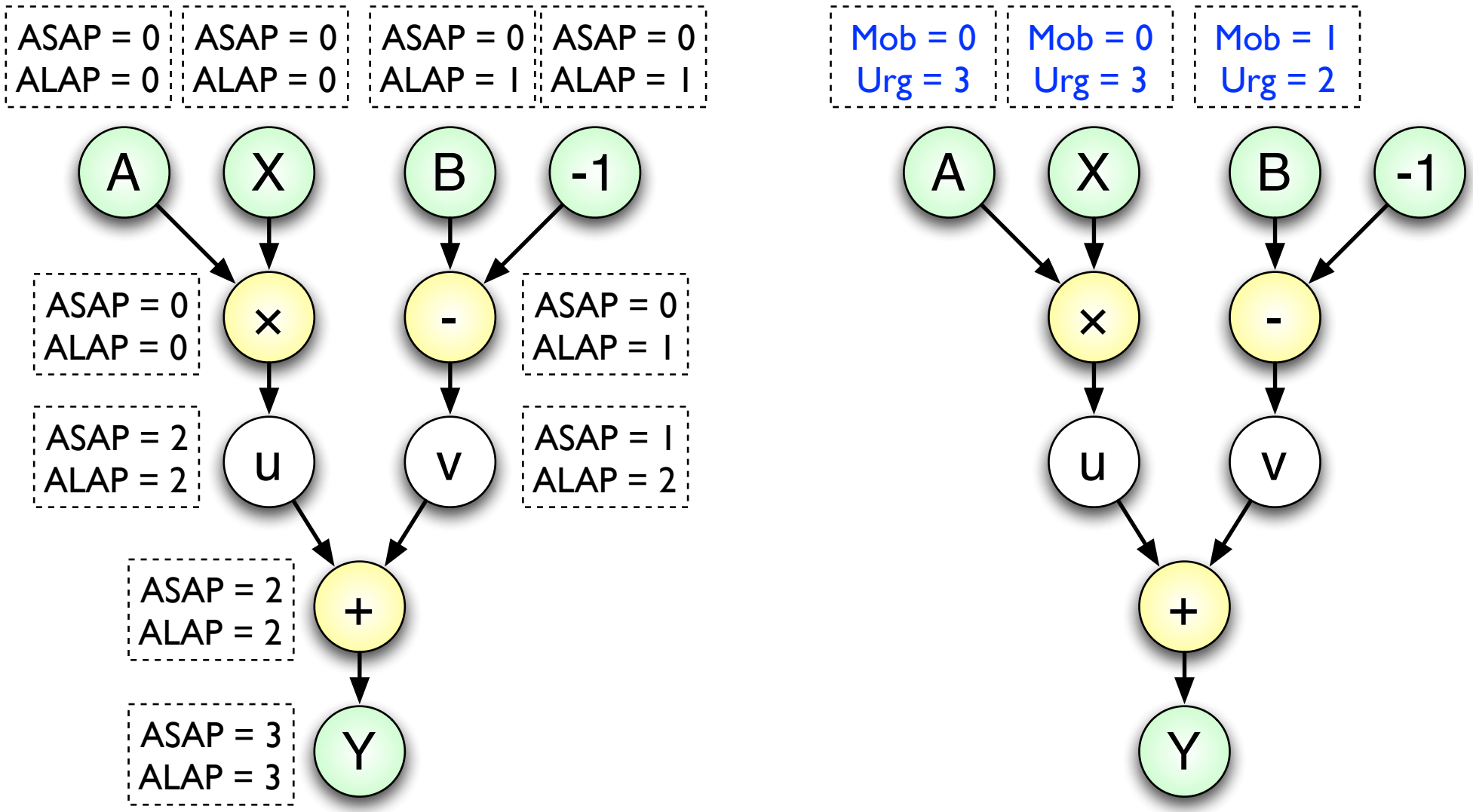
Calcul de l'urgence et de la mobilité des noeuds



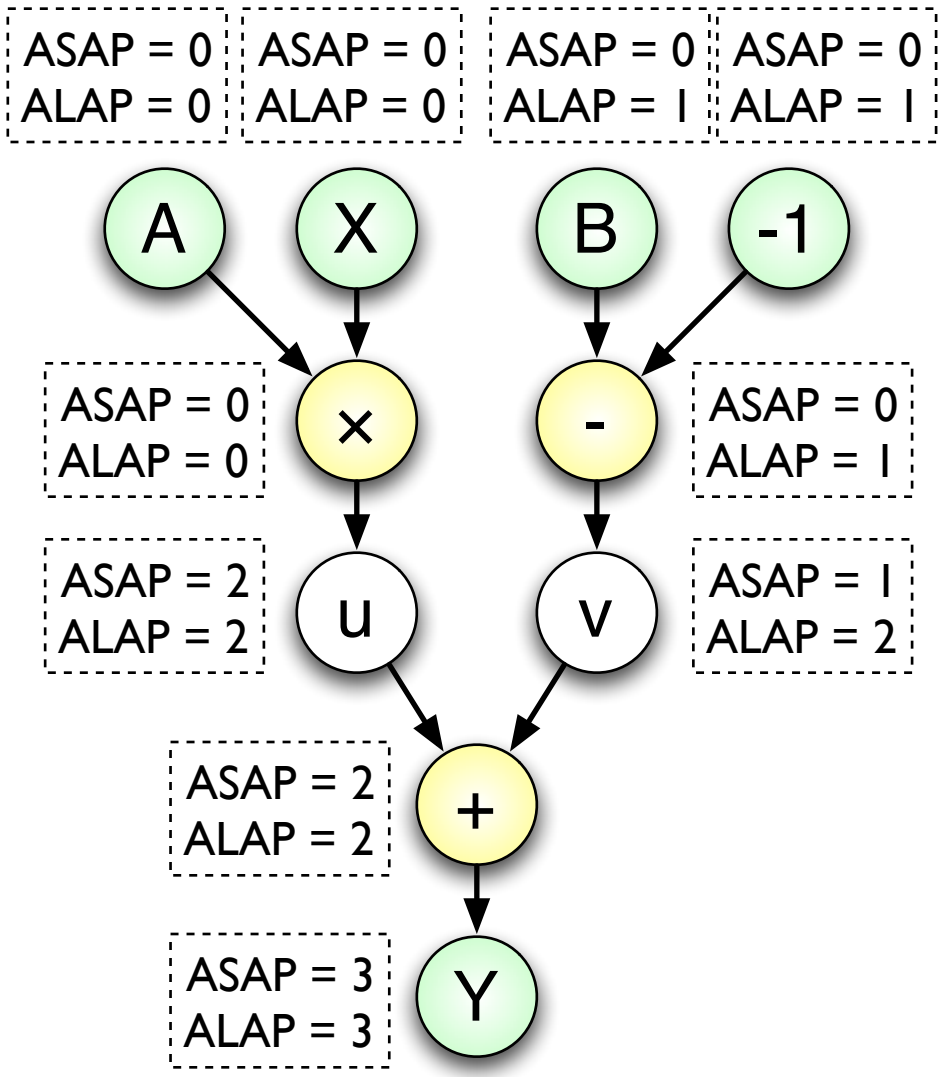
Calcul de l'urgence et de la mobilité des noeuds

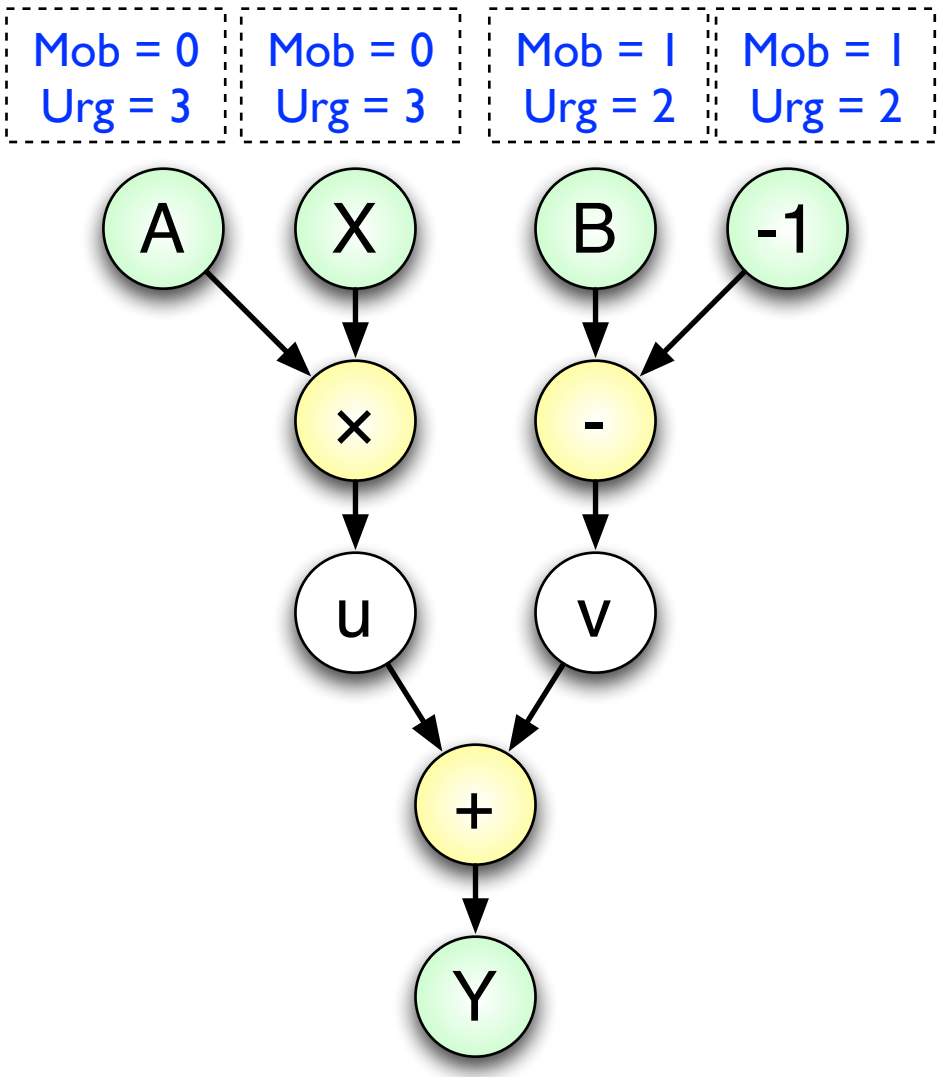


Calcul de l'urgence et de la mobilité des noeuds

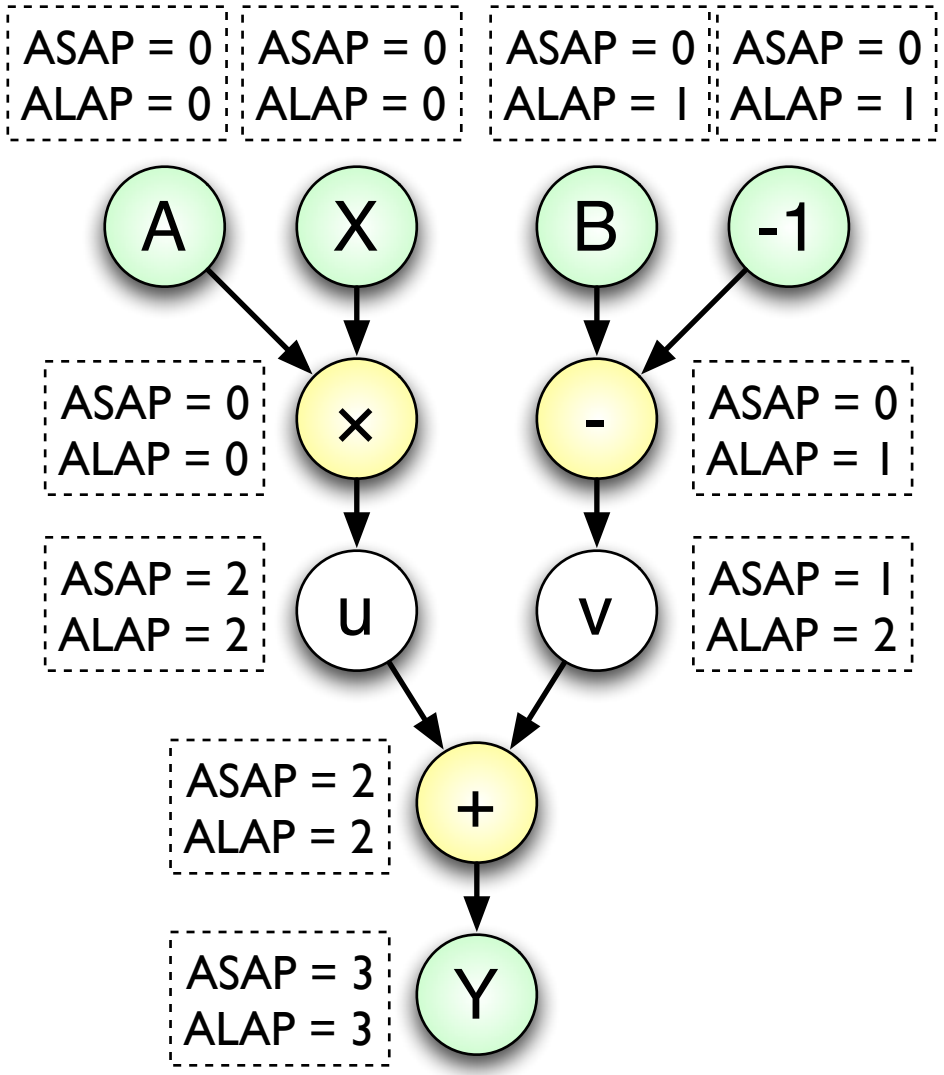


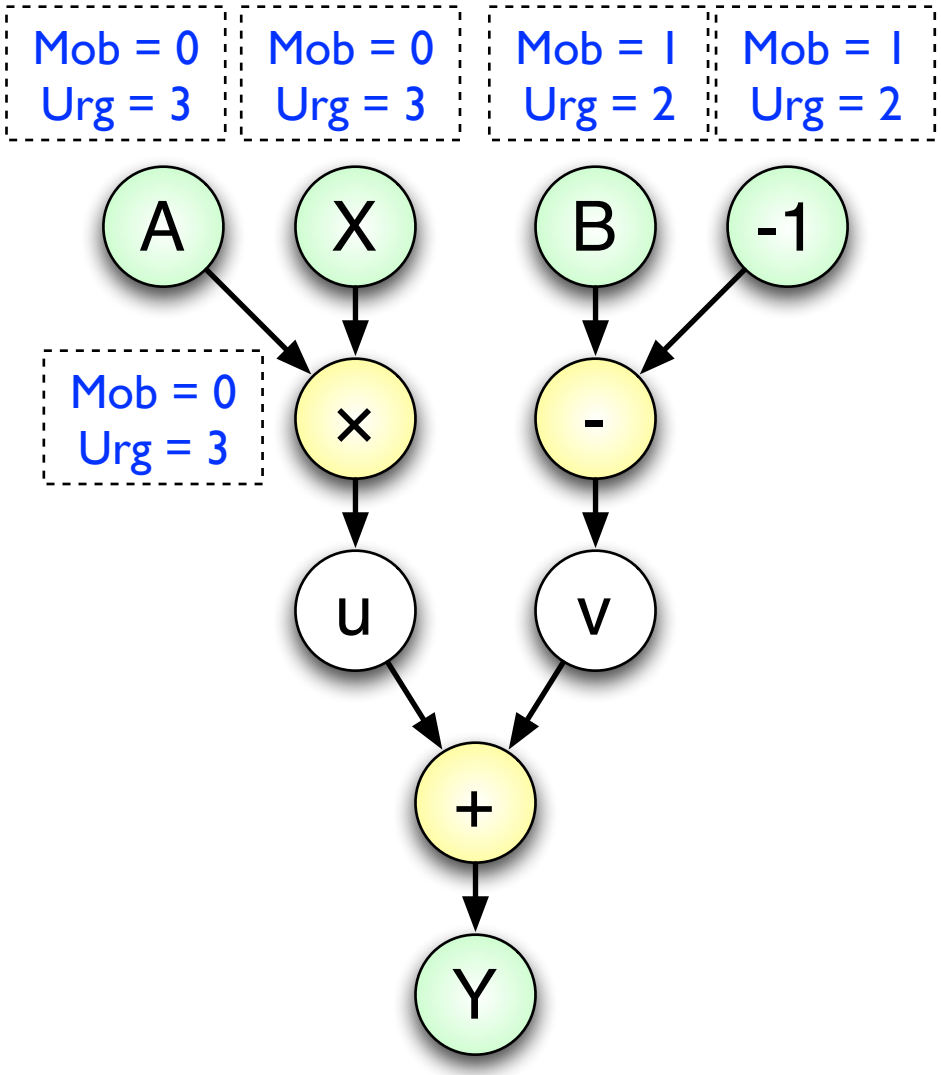
Calcul de l'urgence et de la mobilité des noeuds



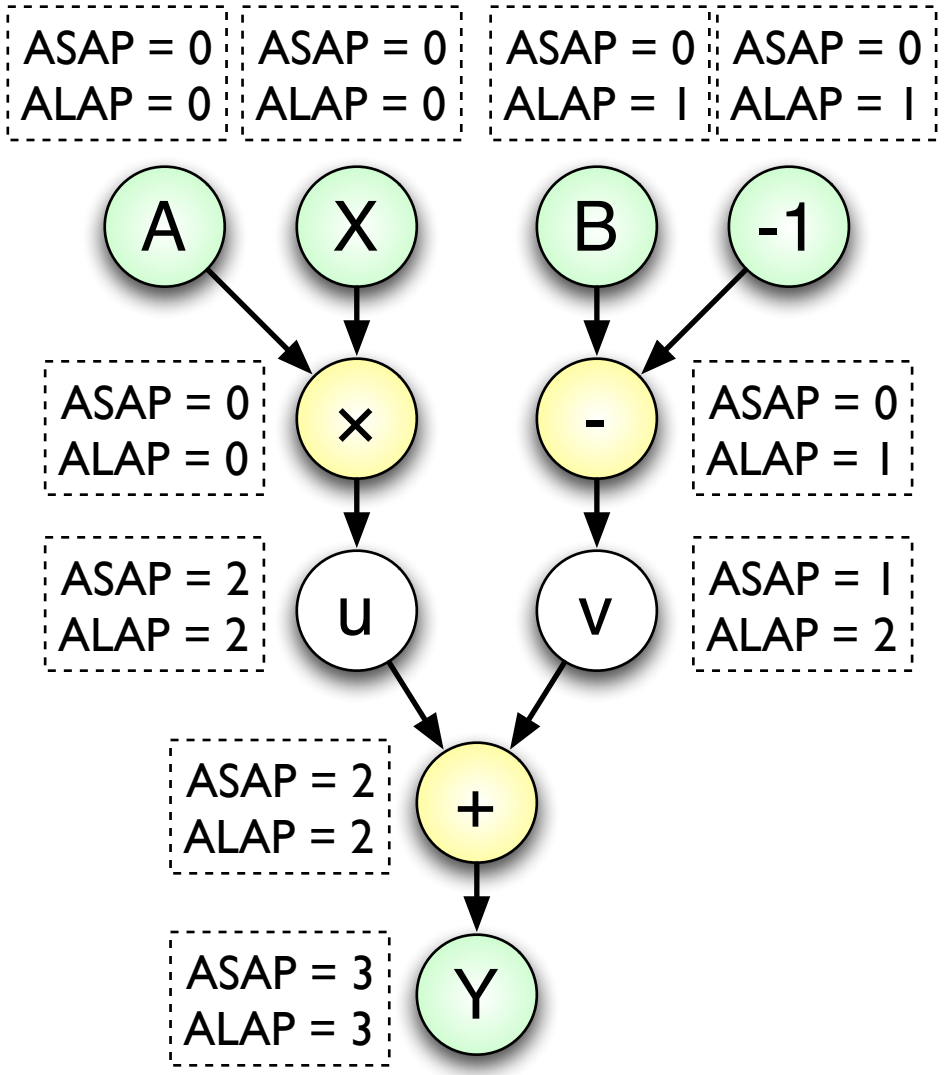


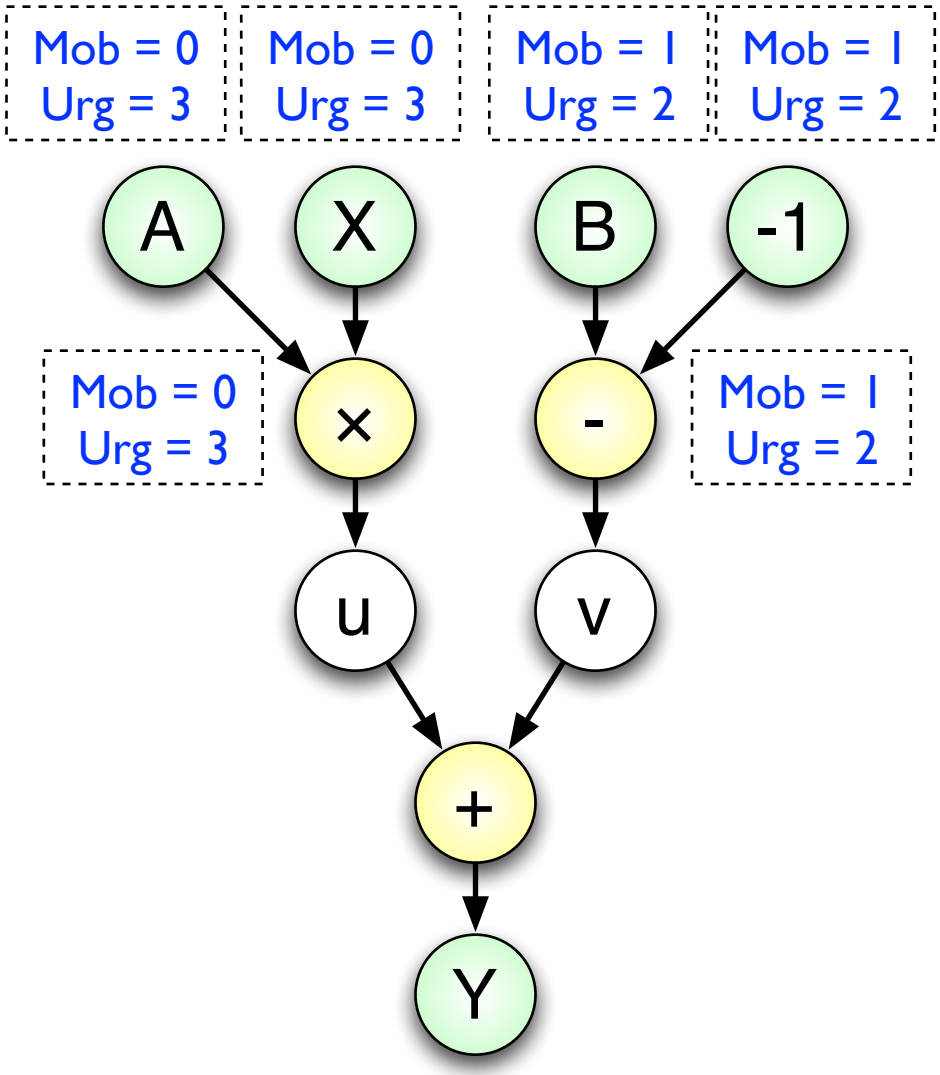
Calcul de l'urgence et de la mobilité des noeuds



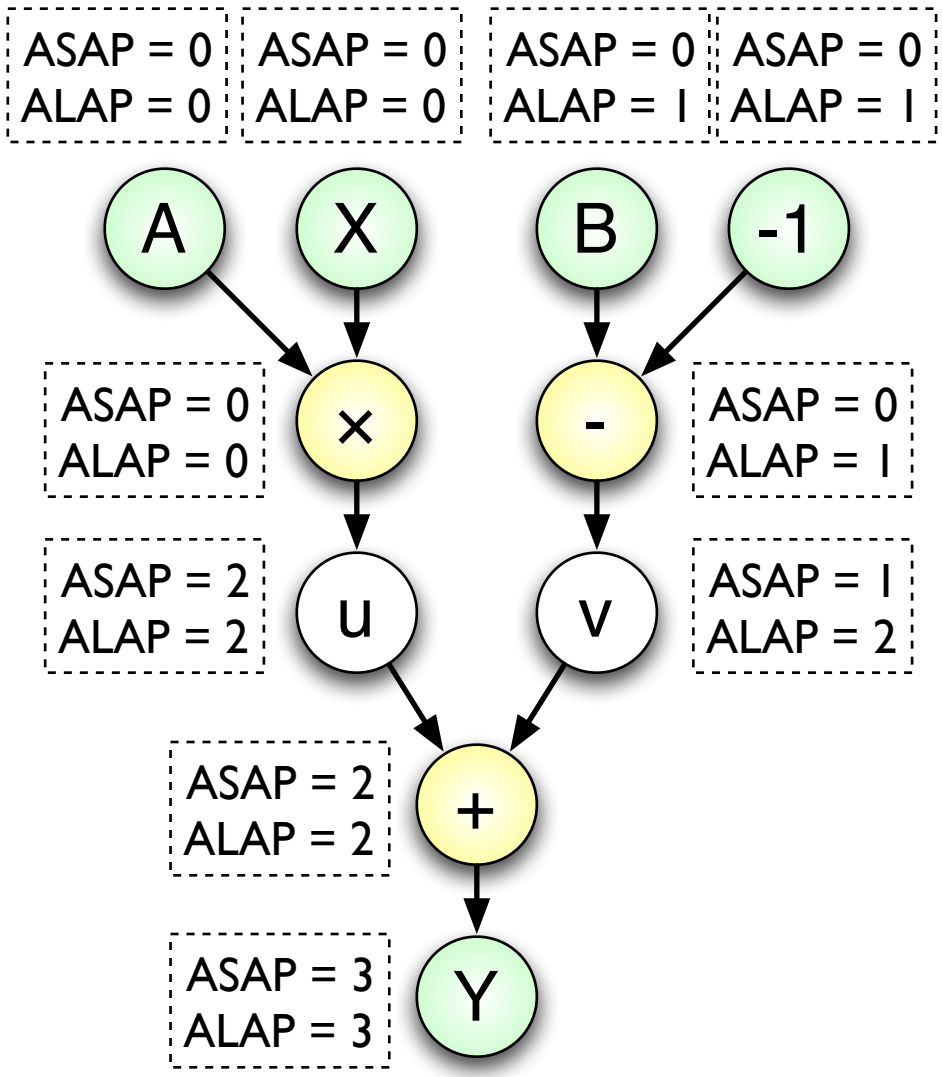


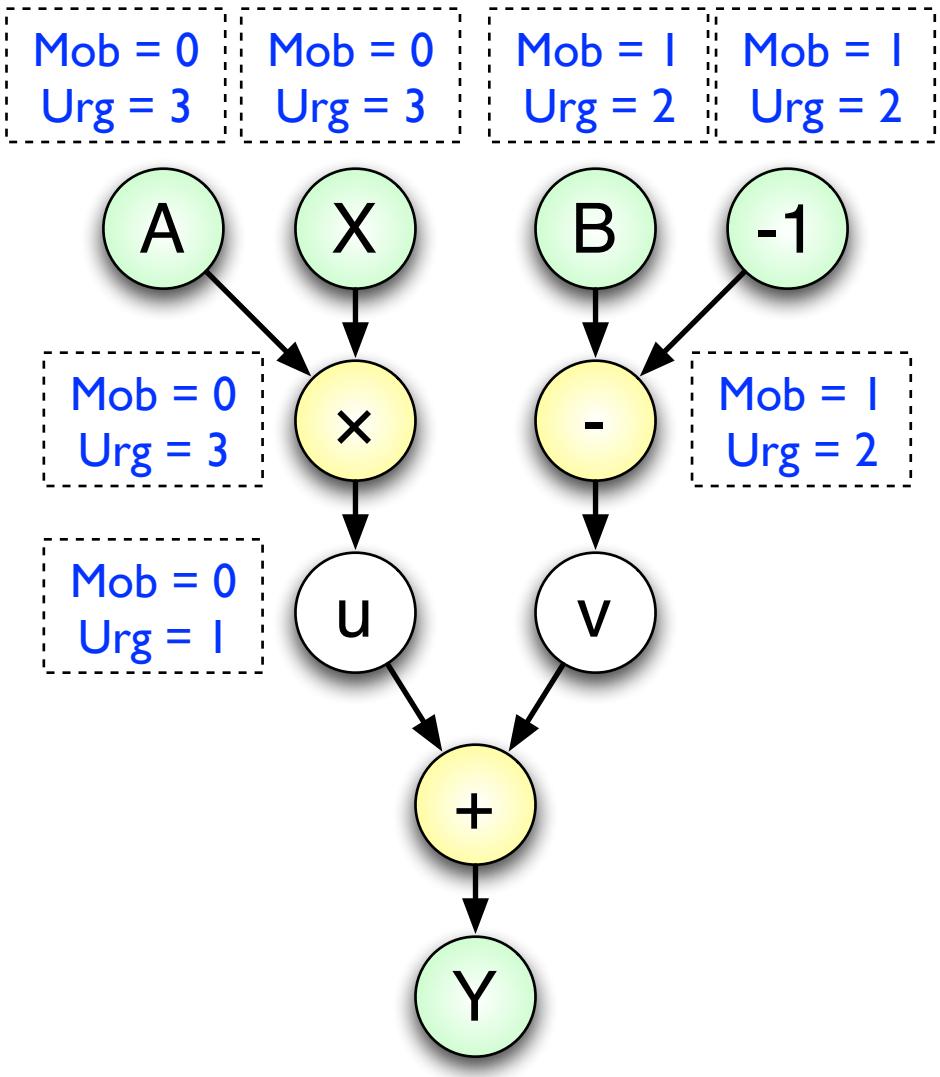
Calcul de l'urgence et de la mobilité des noeuds



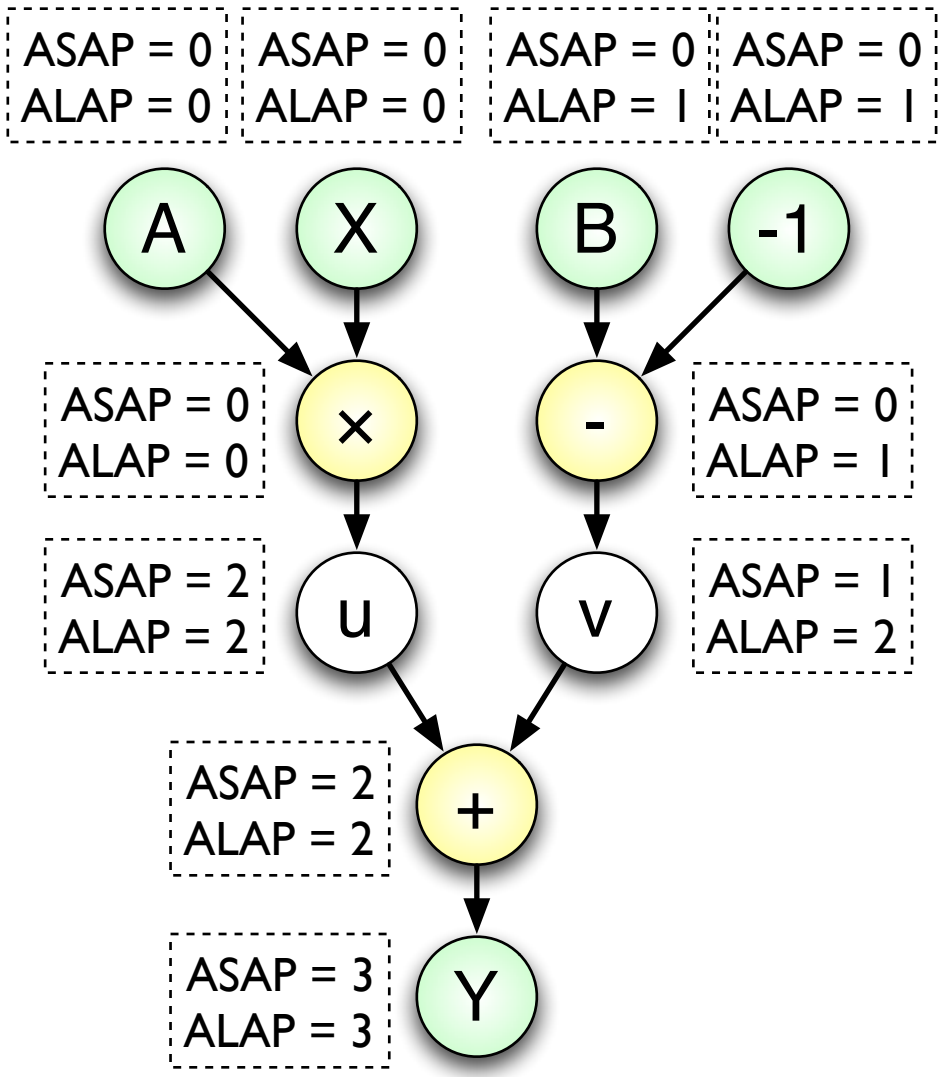


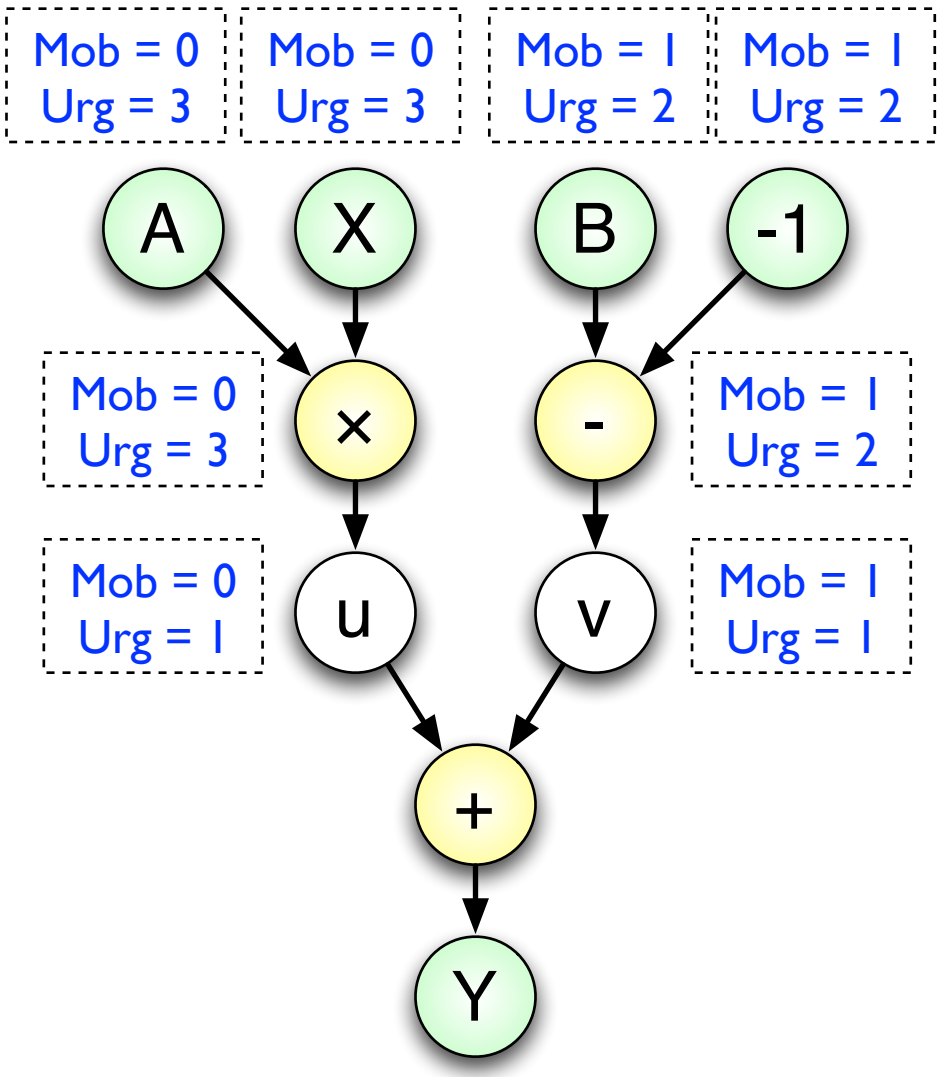
Calcul de l'urgence et de la mobilité des noeuds



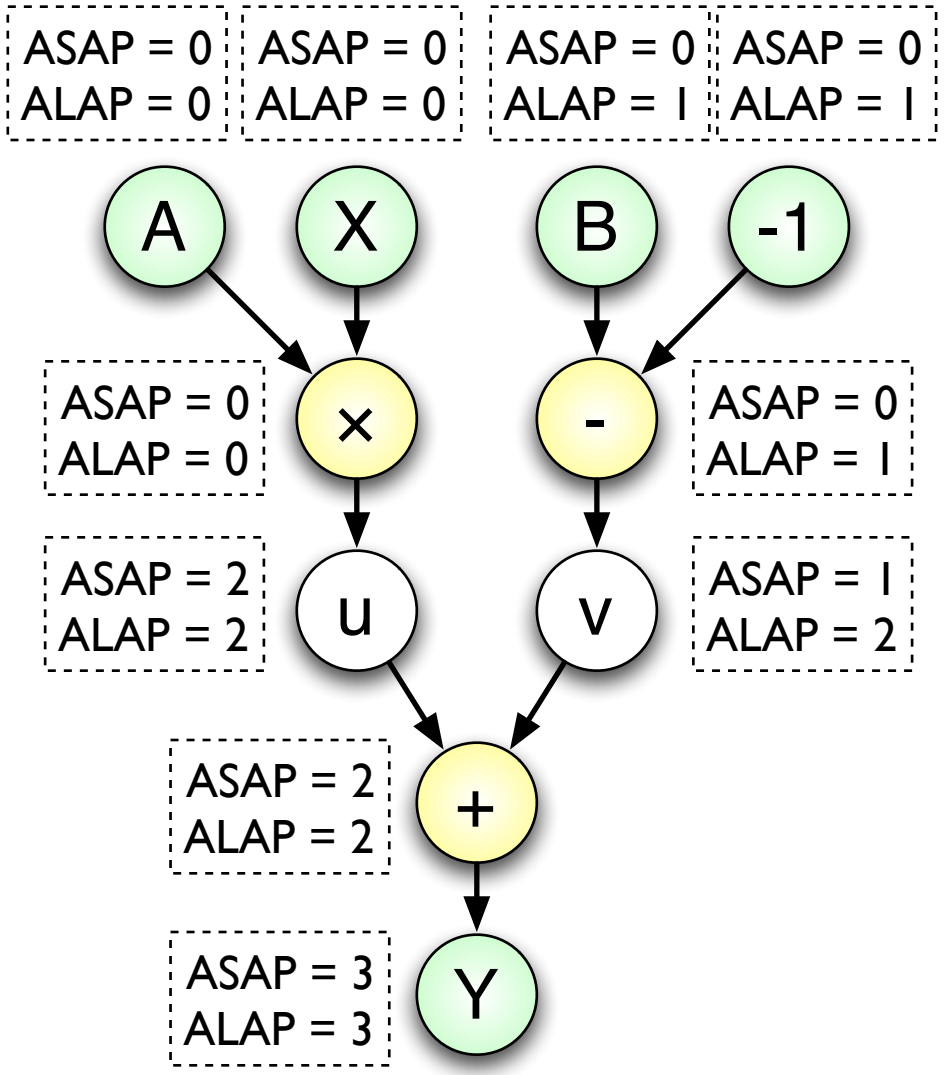


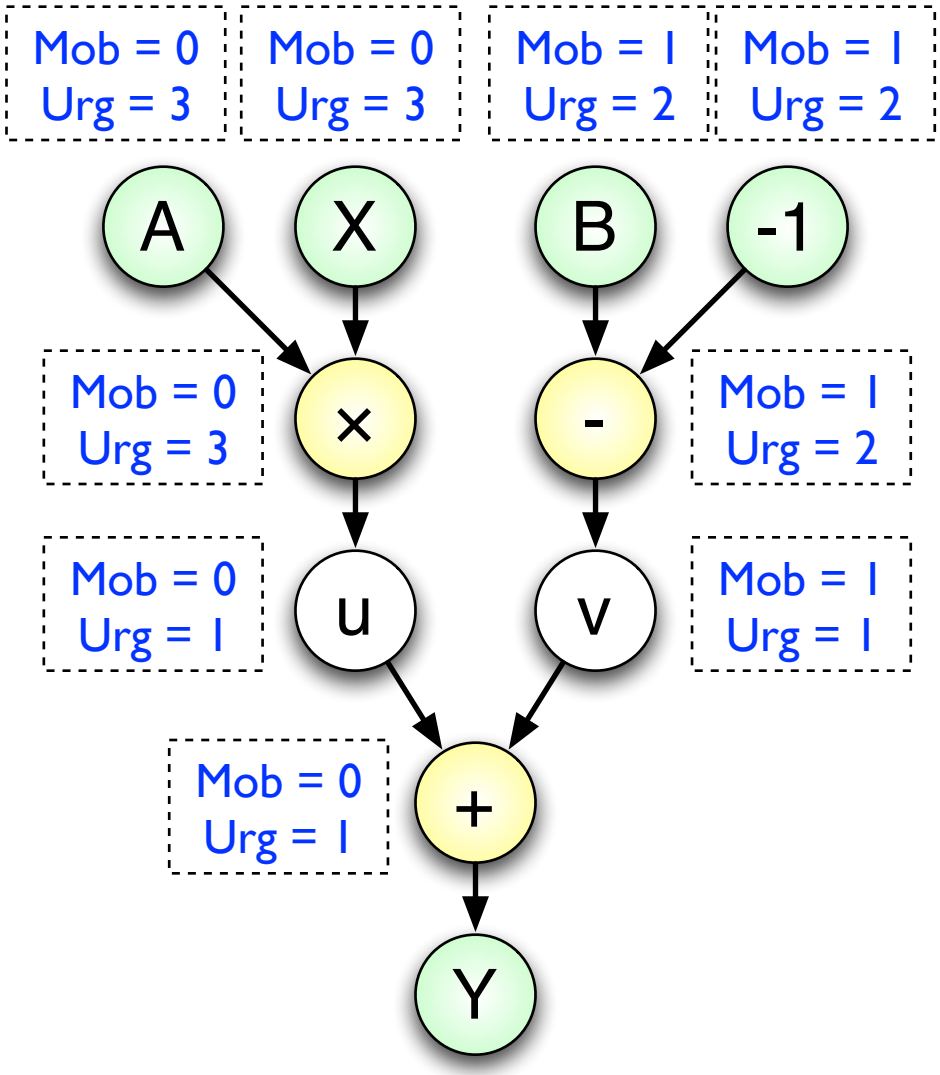
Calcul de l'urgence et de la mobilité des noeuds



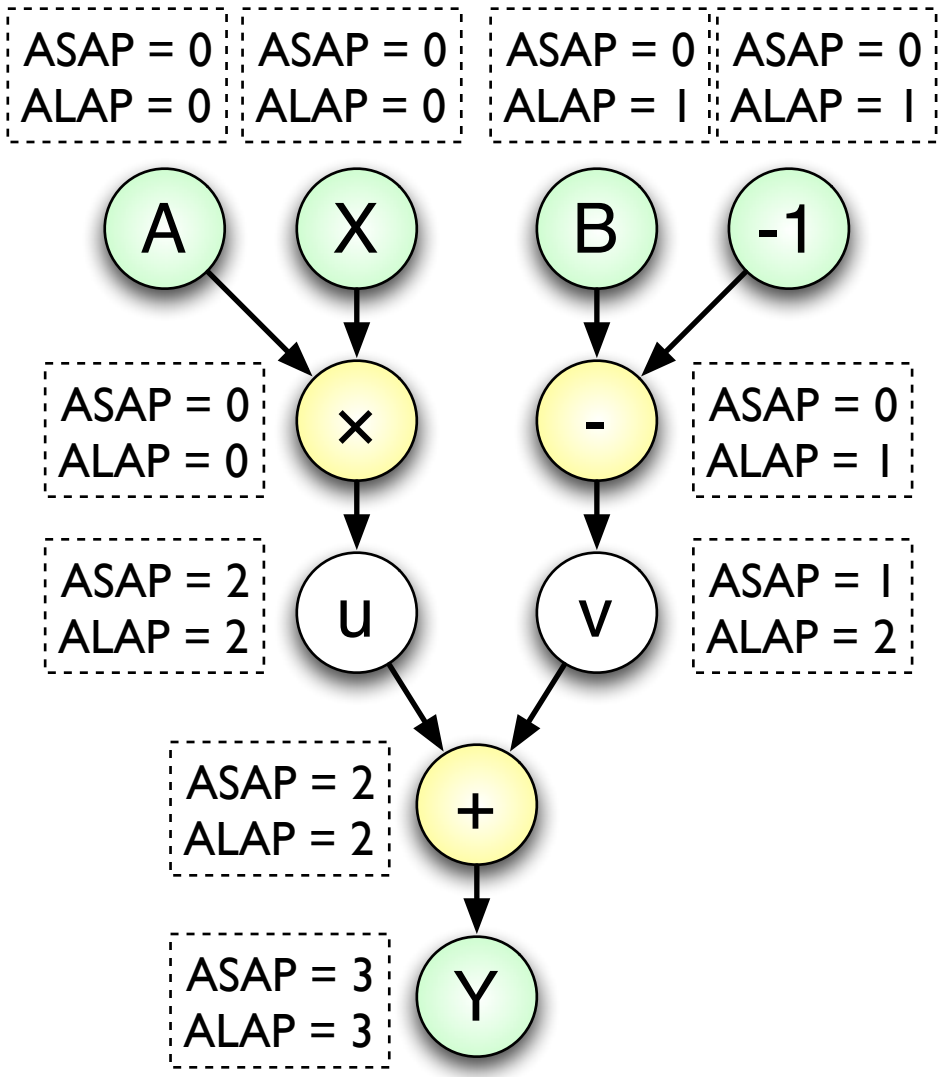


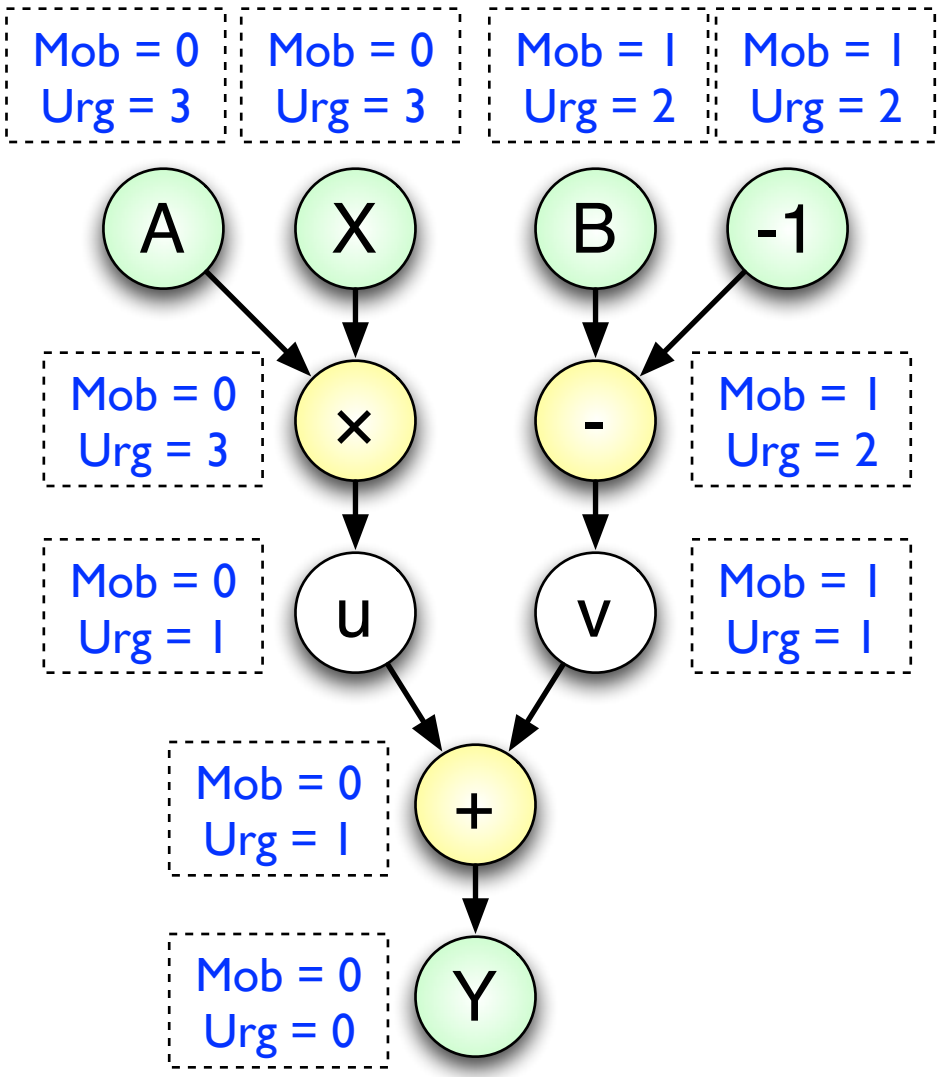
Calcul de l'urgence et de la mobilité des noeuds



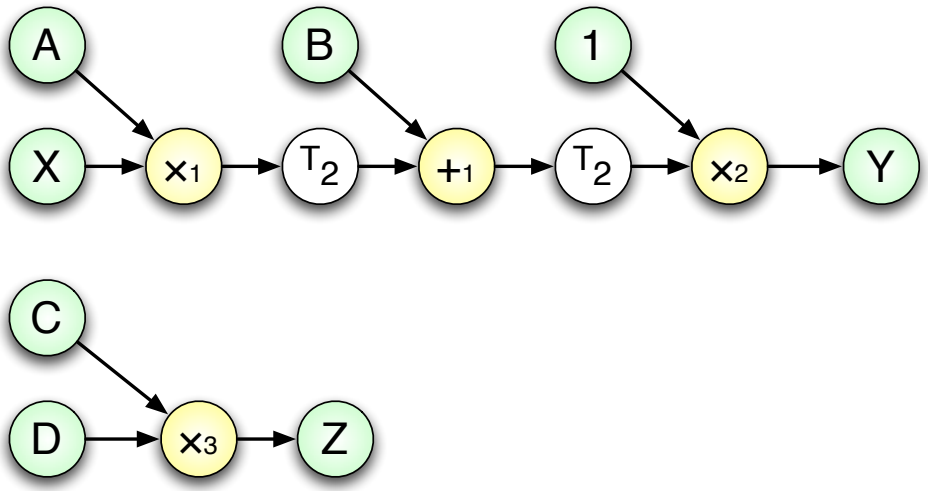


Calcul de l'urgence et de la mobilité des noeuds



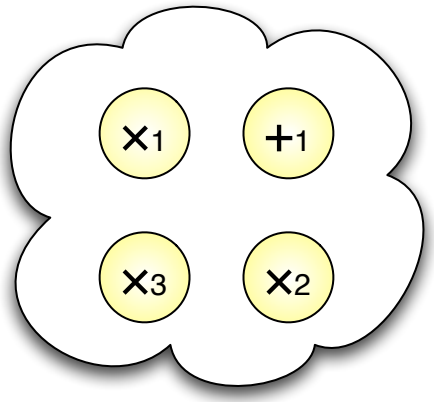


Ordonnancement par liste - Surface contrainte



Opération	ASAP	ALAP	Mobilité
$+_1$	2	2	0
$\times_1$	0	0	0
$\times_2$	3	3	0
$\times_3$	0	3	3

Liste contenant les opérations à exécuter + bornes d'exécution



Opération	ASAP	ALAP	Mobilité	Eligible
$\times_1$	0	0	0	*oui*
$\times_3$	0	3	3	*oui*
$+_1$	2	2	0	*non*
$\times_2$	3	3	0	*non*

Liste triée par Eligibilité / ASAP / mobilité

Ordonnancement par liste - Surface contrainte

Opération	ASAP	ALAP	Mobilité	Eligible
$\times_1$	0	0	0	*oui*
$\times_3$	0	3	3	*oui*
$+_1$	2	2	0	*non*
$\times_2$	3	3	0	*non*

Liste triée par :

Eligibilité / ASAP / mobilité

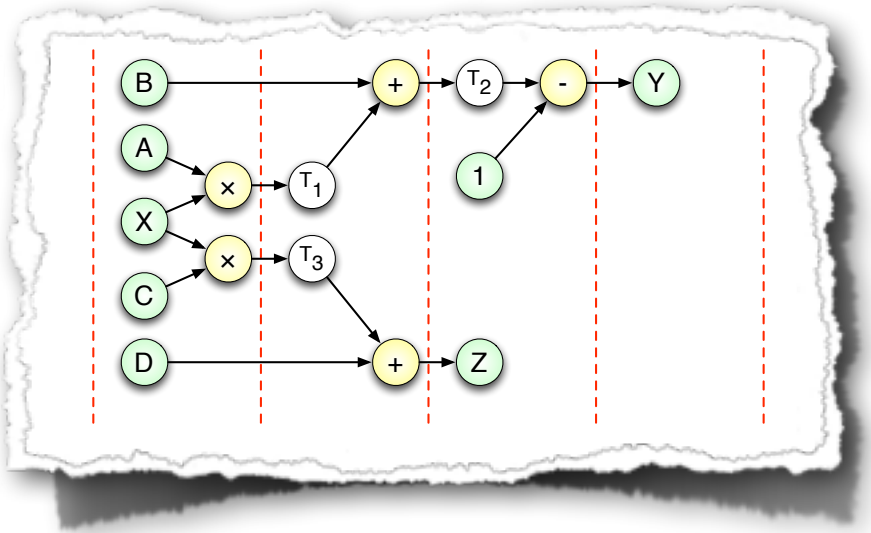
Utiliser comme métrique de décision la plus faible mobilité

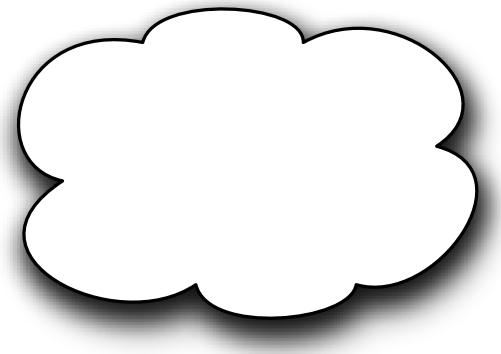
Opérateur	Cycle 0	Cycle 1	Cycle 2	Cycle 3
×				
+				

Utiliser comme métrique de décision la plus forte mobilité

Opérateur	Cycle 0	Cycle 1	Cycle 2	Cycle 3
×				
+				

Ordonnancement (plus faible mobilité)



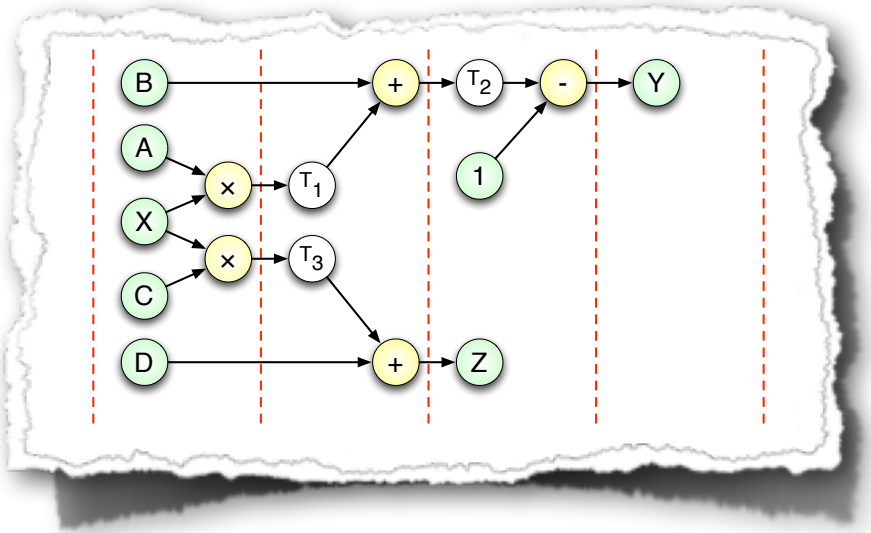


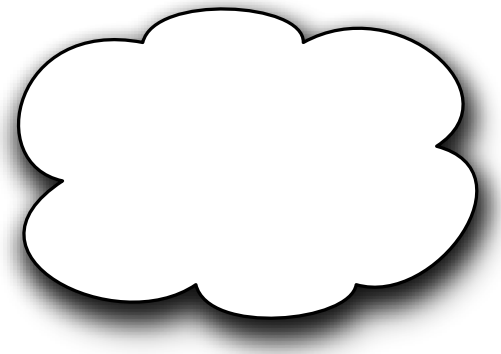
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$				
$\times_2$				
$+_1$				
$+_2$				
$-_1$				

Cycle	Mult (1)	Add (1)	Sub (1)
1			
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



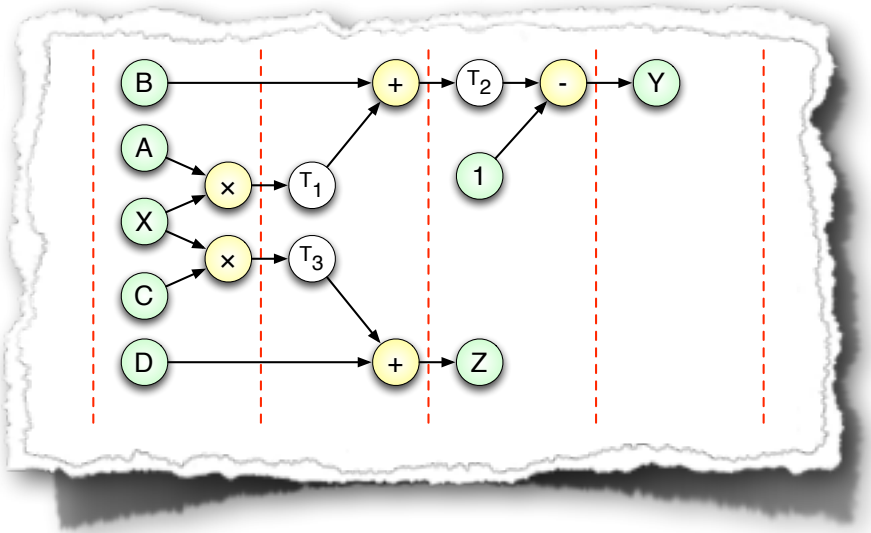


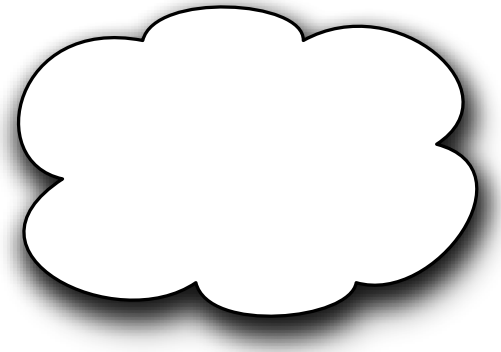
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0			
×2	0			
+1	1			
+2	1			
-1	2			

Cycle	Mult (1)	Add (1)	Sub (1)
1			
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



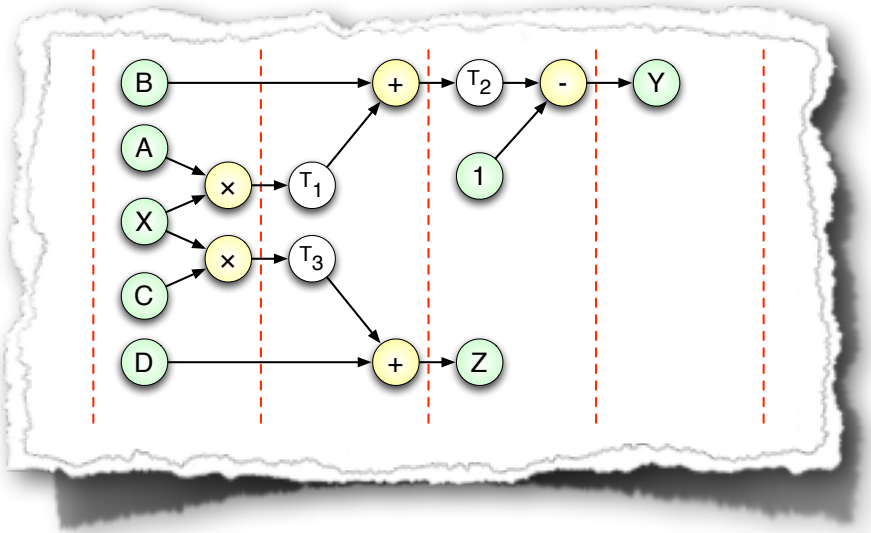


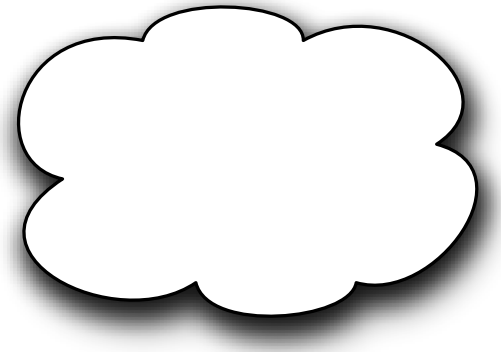
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0		
×2	0	1		
+1	1	2		
+2	1	1		
-1	2	2		

Cycle	Mult (1)	Add (1)	Sub (1)
1			
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



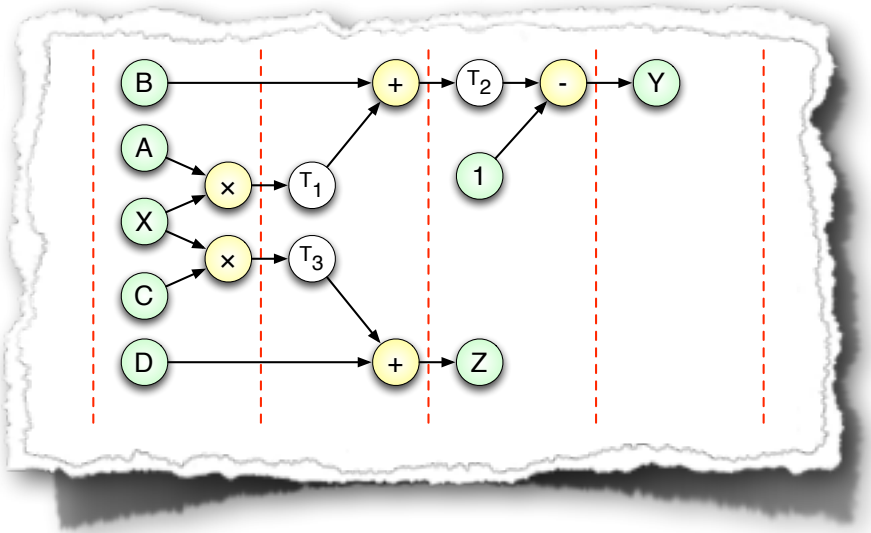


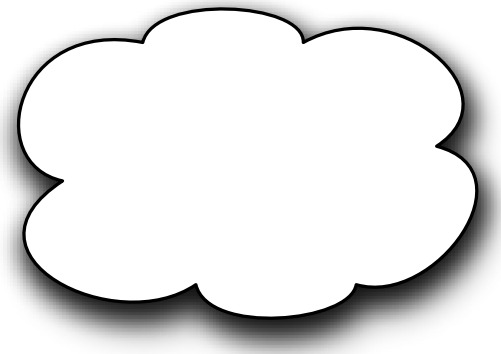
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	
×2	0	1	1	
+1	1	2	1	
+2	1	1	0	
-1	2	2	0	

Cycle	Mult (1)	Add (1)	Sub (1)
1			
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



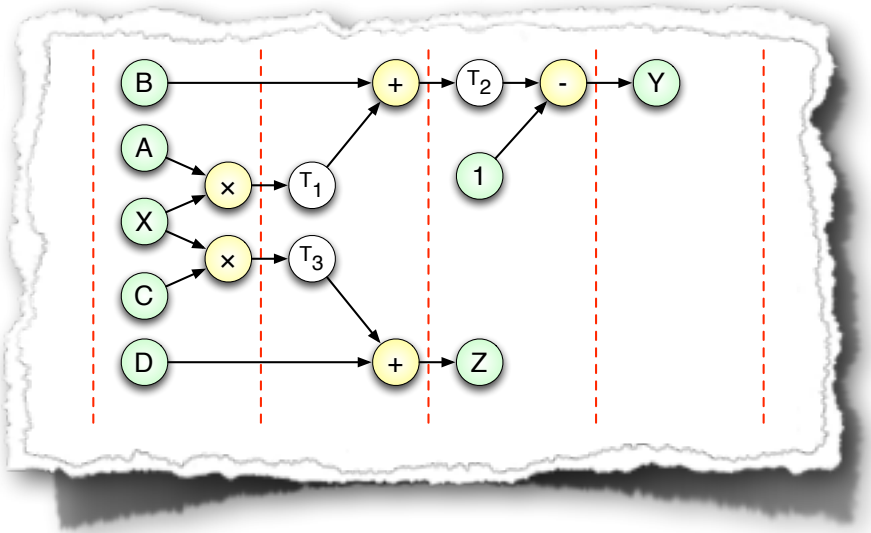


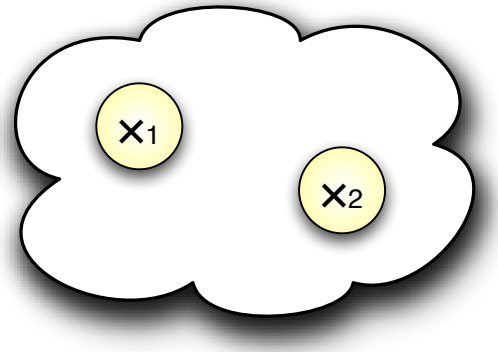
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1			
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



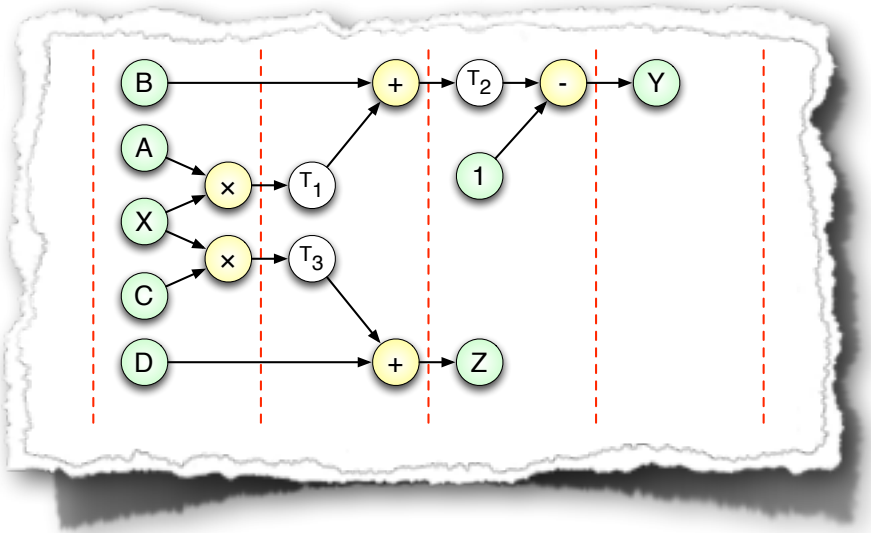


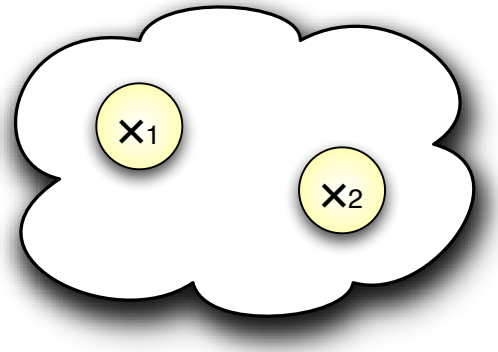
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1			
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



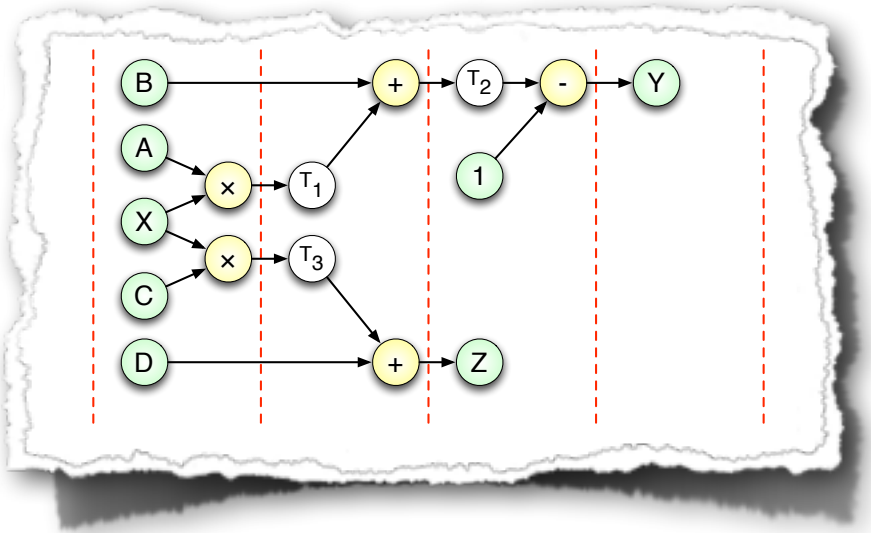


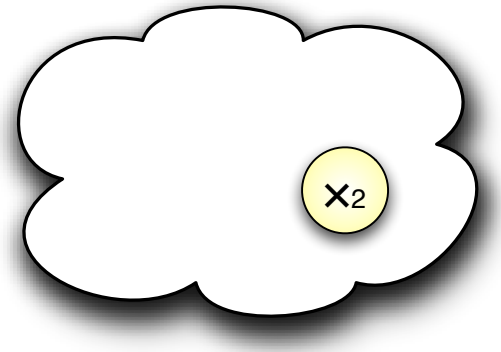
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1			
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



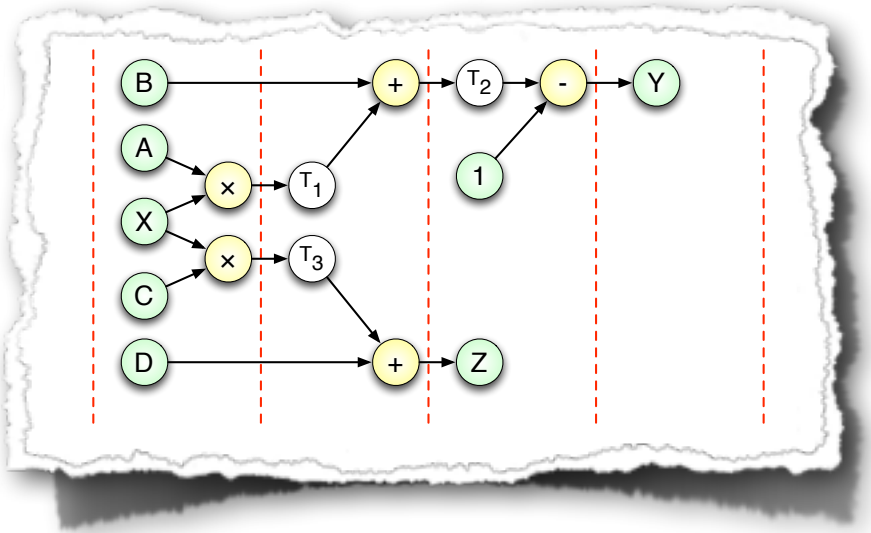


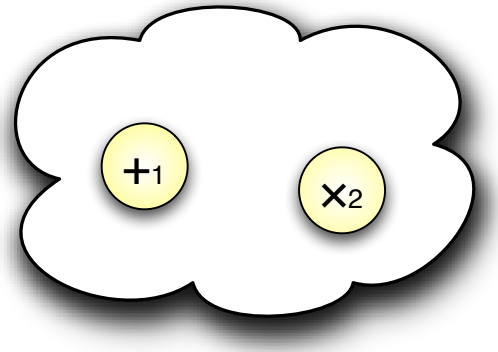
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1	×1		
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



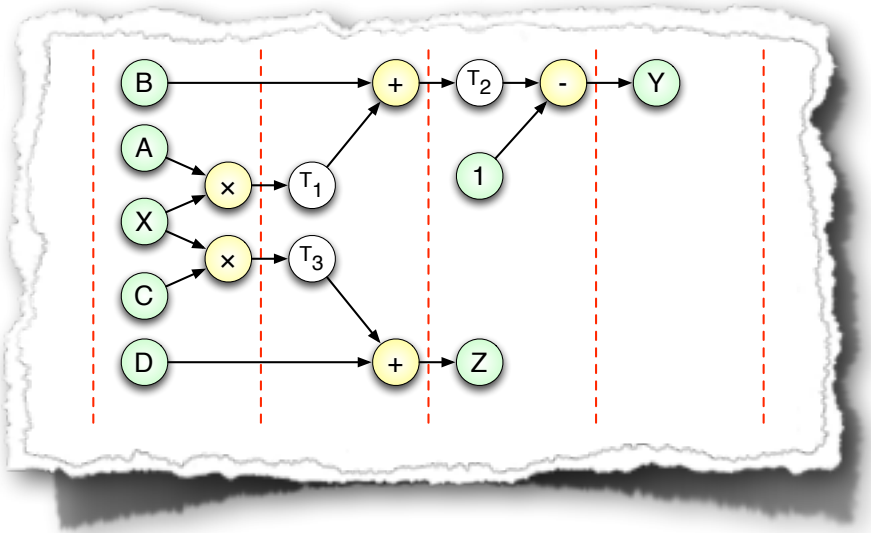


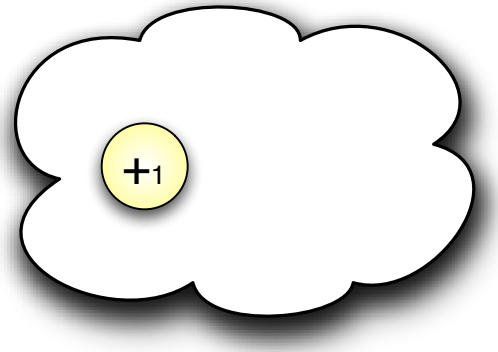
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1	×1		
2			
3			
4			
5			

Ordonnancement (plus faible mobilité)



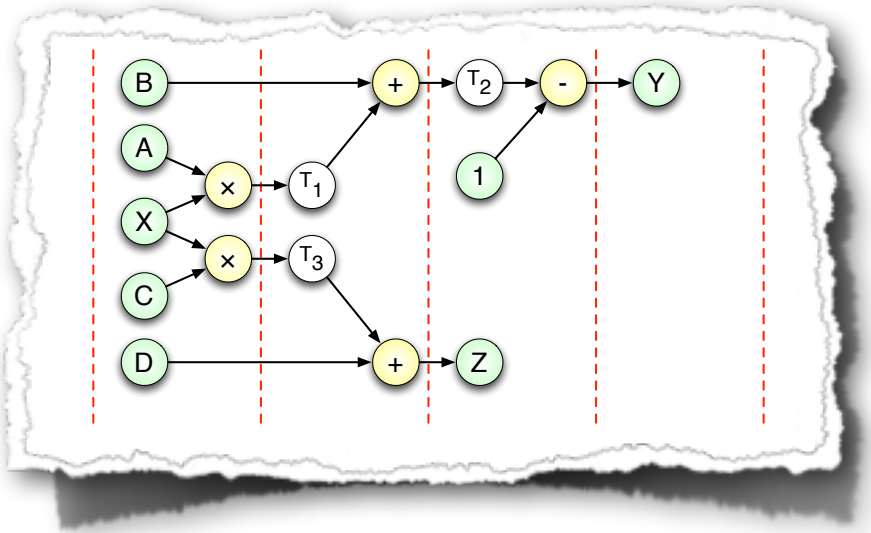


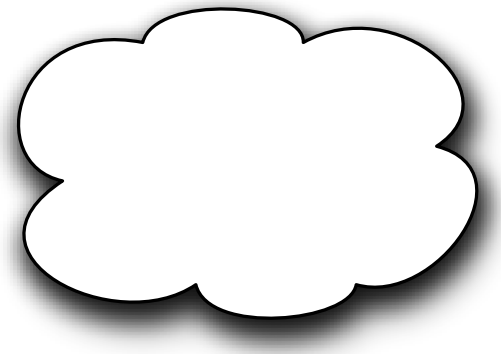
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1	×1		
2	×2		
3			
4			
5			

Ordonnancement (plus faible mobilité)



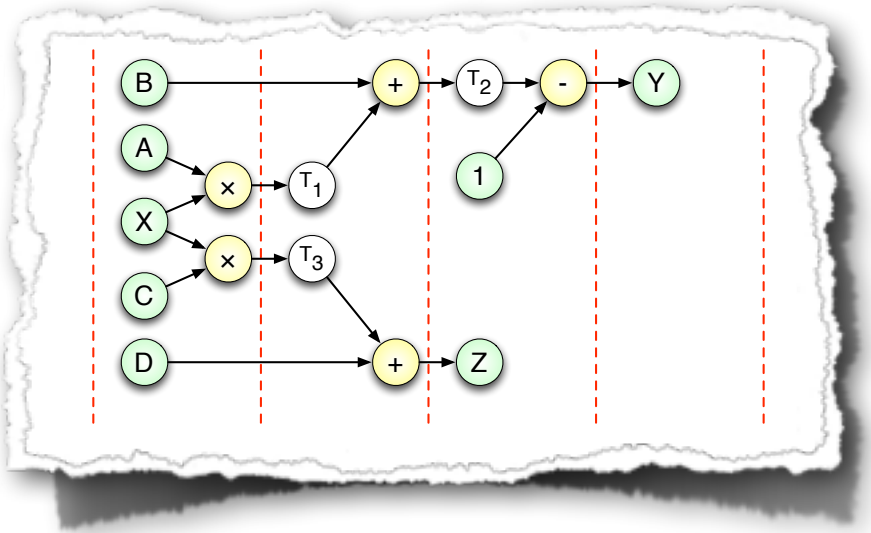


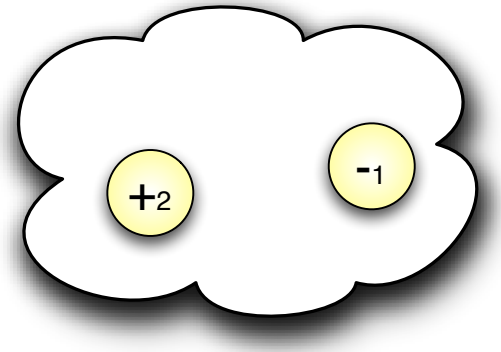
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1	×1		
2	×2	+1	
3			
4			
5			

Ordonnancement (plus faible mobilité)



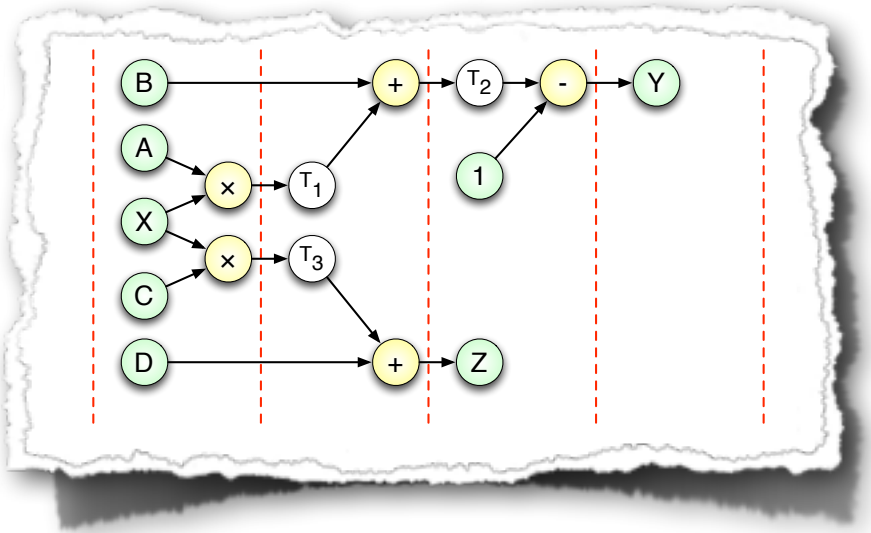


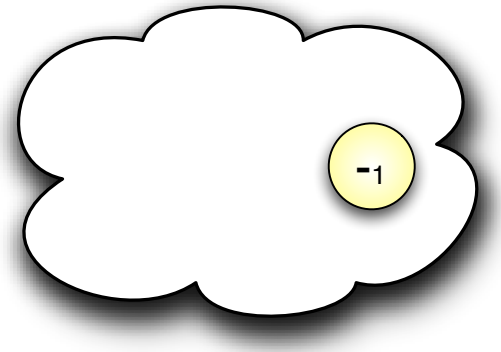
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1	×1		
2	×2	+1	
3			
4			
5			

Ordonnancement (plus faible mobilité)



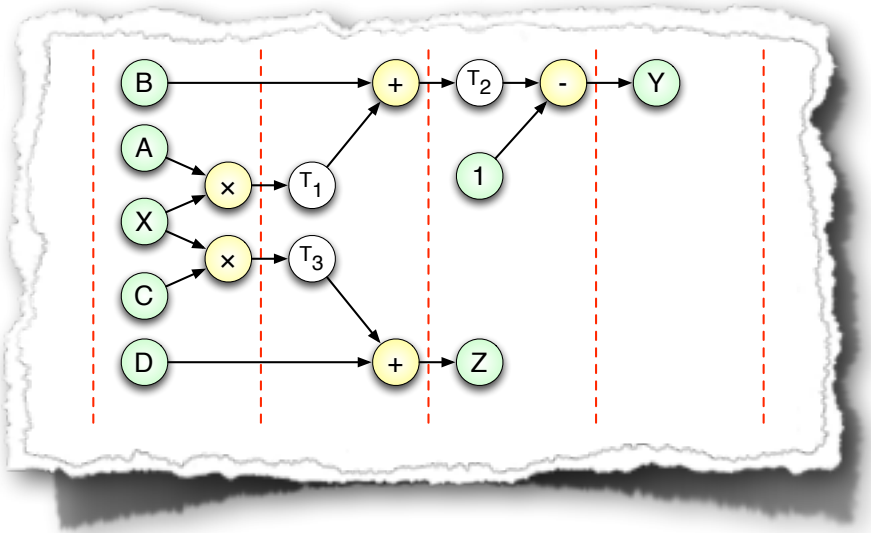


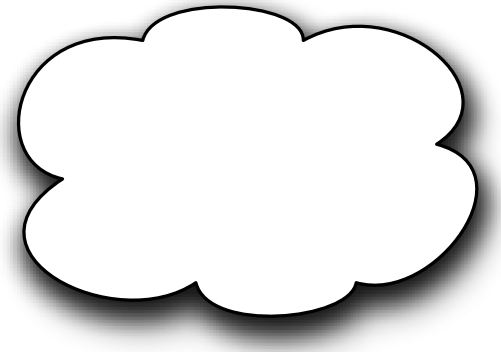
Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1	×1		
2	×2	+1	
3		+2	
4			
5			

Ordonnancement (plus faible mobilité)



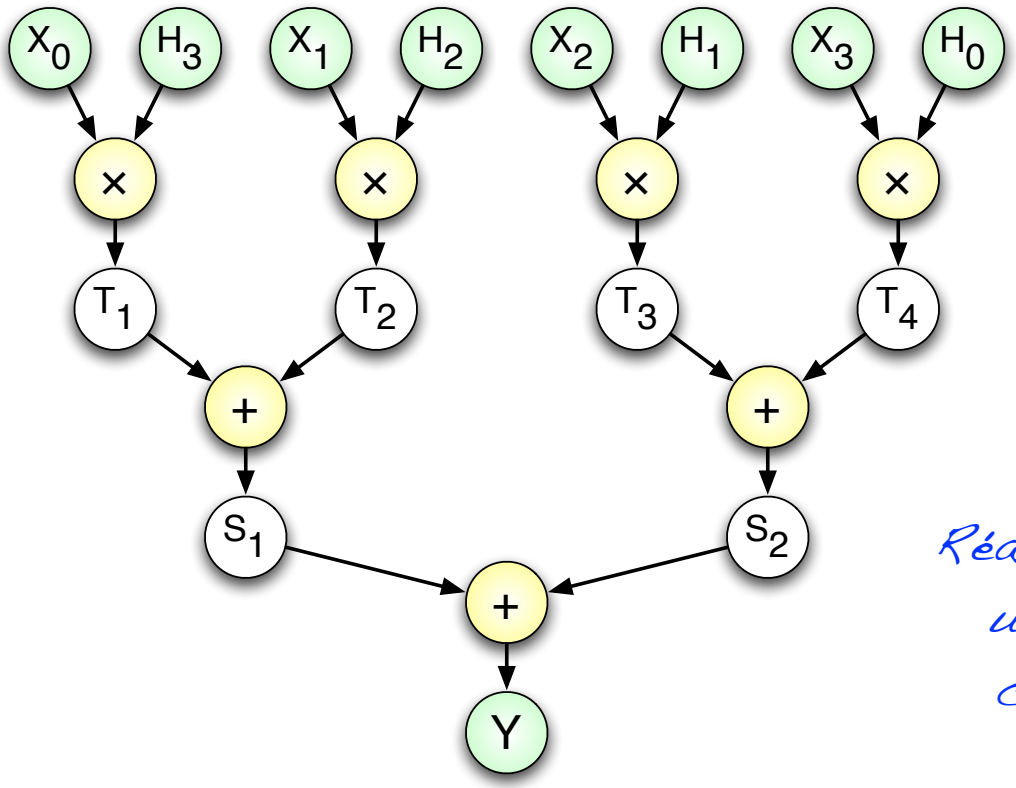


Opérations ordonnançables

Noeud	ASAP	ALAP	Mobilité	Urgence
×1	0	0	0	3
×2	0	1	1	2
+1	1	2	1	2
+2	1	1	0	1
-1	2	2	0	1

Cycle	Mult (1)	Add (1)	Sub (1)
1	×1		
2	×2	+1	
3		+2	-1
4			
5			

Ordonnancement sous contrainte matérielle, à vous de jouer...

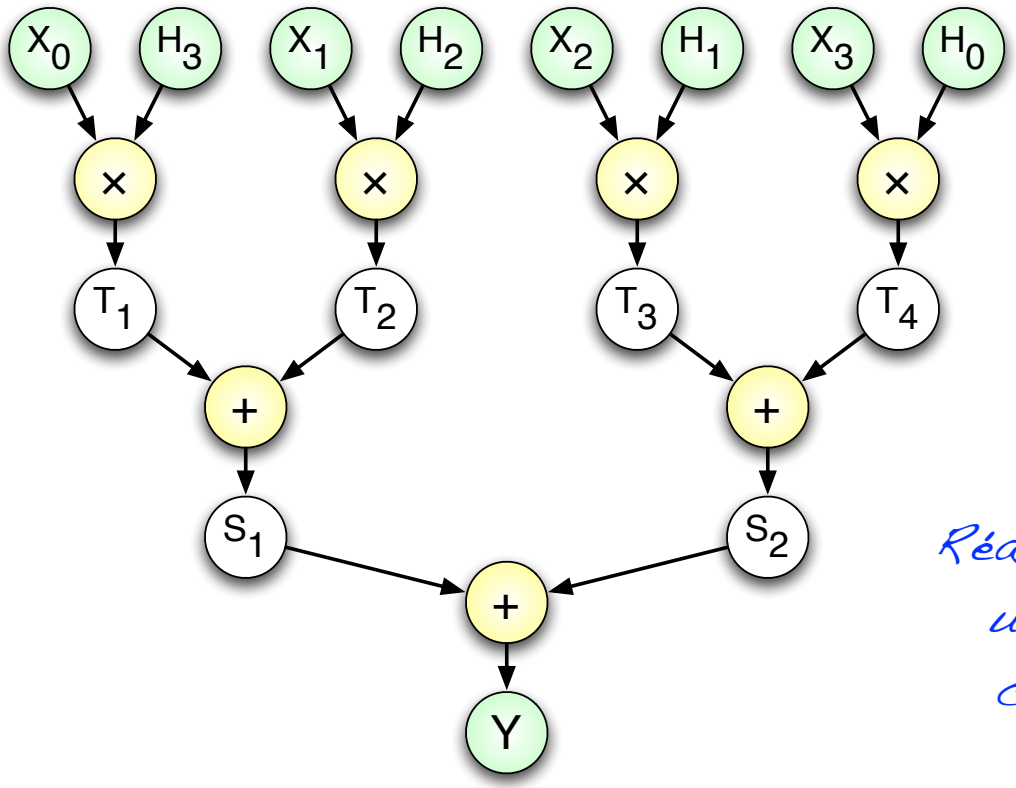


La contrainte matérielle appliquée à la synthèse est que vous disposez uniquement d'un additionneur et un multiplieur

Réaliser l'ordonnancement de ce graphe avec un algorithme de type liste. Combien de cycles d'horloge sont nécessaire pour cette application ?

Opérateur	Cycle 0	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7
×								
+								

Ordonnancement sous contrainte matérielle, à vous de jouer...



La contrainte matérielle appliquée à la synthèse est que vous disposez uniquement d'un additionneur et de 2 multiplieurs

Réaliser l'ordonnancement de ce graphe avec un algorithme de type liste. Combien de cycles d'horloge sont nécessaire pour cette application ?

Opérateur	Cycle 0	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7
×								
×								
+								

Les techniques complémentaires

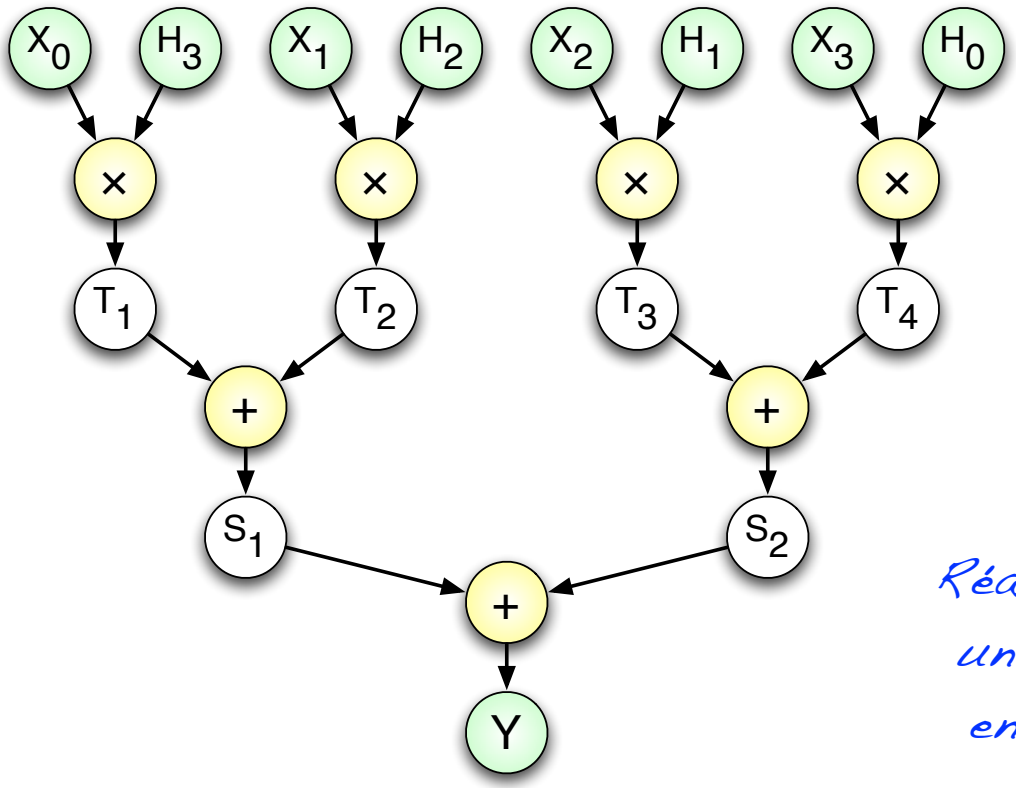
- La fonction de coût (choix entre 2 noeuds éligibles) à un impact certain sur les performances de l'étape d'ordonnancement,
- Différents métriques ont été proposés dans la littérature afin d'optimiser au mieux l'ordonnancement,
  - Tri des opérations par mobilité,
  - Tri des opérations par appartenance au chemin critique,
  - Tri des opérations par nombre de successeurs presque éligibles,
  - Tri des opérations par nombre de successeurs,
  - Composition des métriques ci-dessus,
- Malheureusement l'efficacité des différents métriques de choix est assez souvent liée à l'application (modèle formel) traité.

Ordonnancement contrainte temporelle

- Nous allons faire évoluer l'algorithme basé sur les listes afin de gérer une contrainte forte (latence maximum)
  - On va placer toutes les opérations à réaliser dans une liste,
  - On va trier les opérations dans la liste en fonction d'un critère,
  - On va sélectionne les opérations prêtes à être ordonnancés dans la liste sous réserve d'avoir assez de ressources matérielles disponibles,
  - On regarde si des opérations sont en attente avec (mobilité = 0)
  - Si oui, on alloue une nouvelle ressource matérielle et on ordonnance l'opération,
  - On réitère sur l'étape (4)
  - On libère l'ensemble des ressources et on incrémente le compteur de cycles d'horloge écoulés,
  - On réitère (3) tant que la liste n'est pas vide.

Hypothèse : la datation ALAP a été effectuée à l'aide de la contrainte temporelle max.

Ordonnancement sous contrainte temporelle, à vous de jouer...

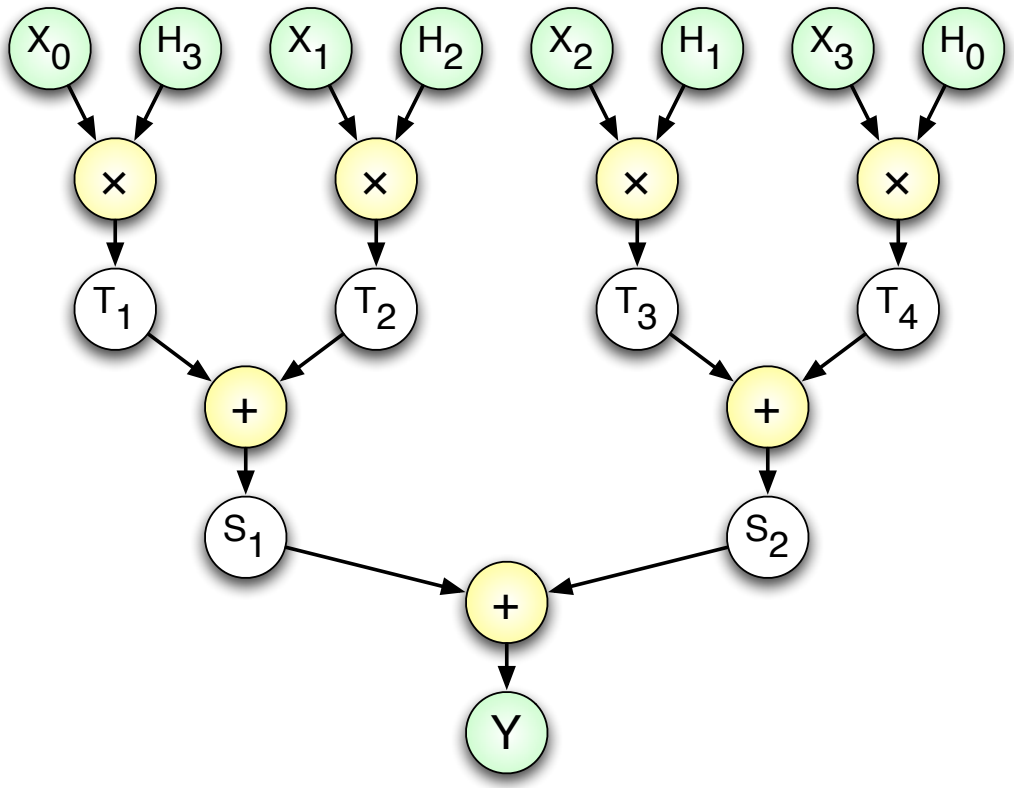


La contrainte temporelle appliquée à la synthèse est de 4 cycles en considérant $T(+) = 1$ et $T(\times) = 1$

Réaliser l'ordonnancement de ce graphe avec un algorithme de type liste. Sans itération en cas d'ajout d'une nouvelle ressource.

Opérateur	Cycle 0	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
×						
+						

Ordonnancement sous contrainte temporelle, à vous de jouer...



Recommencez la même opération en considérant maintenant que l'on relance l'ordonnancement de 0 à chaque ajout d'une nouvelle ressource matérielle

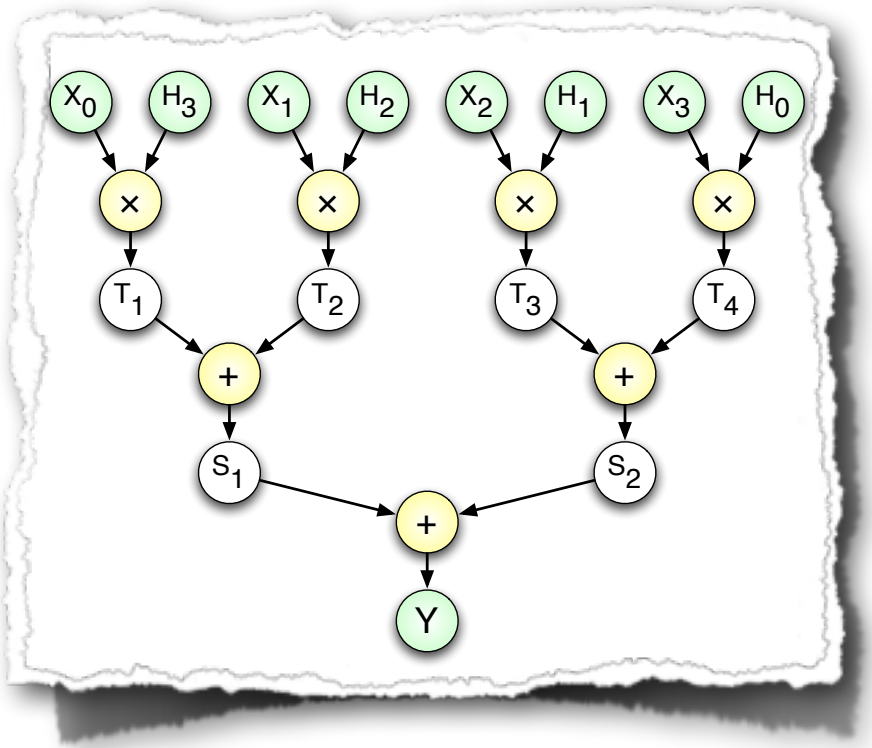
Opérateur	Cycle 0	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
×						
+						

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

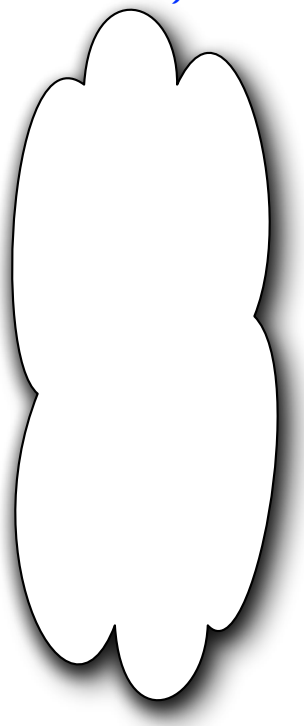
Allocation d'origine => $\pi(+) = 1$ et $\pi(\times) = 1$

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



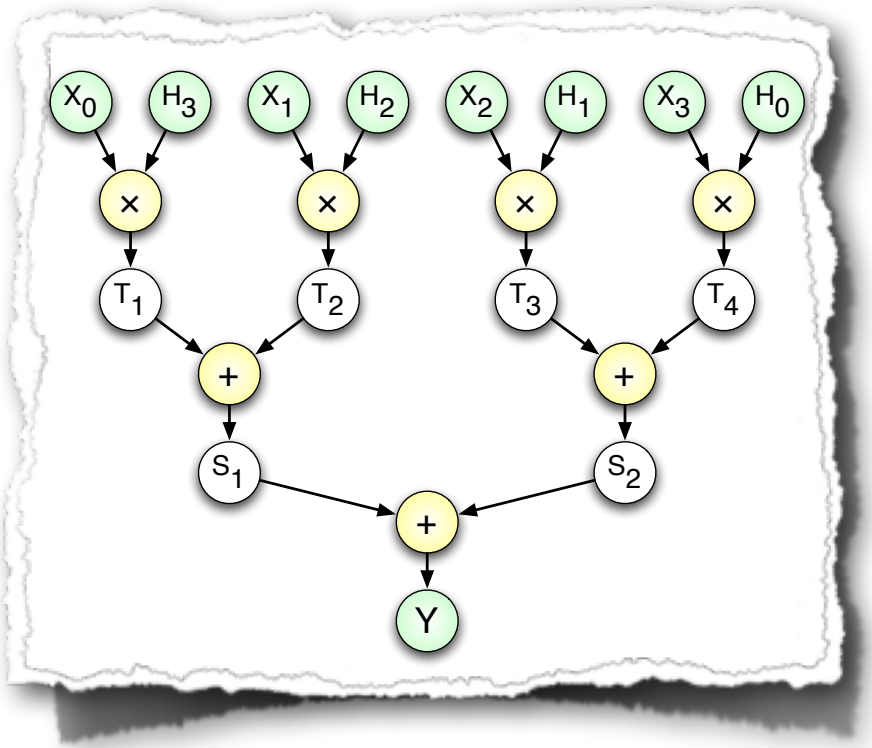
Cycle	Mult (1)	Add (1)			
1					
2					
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

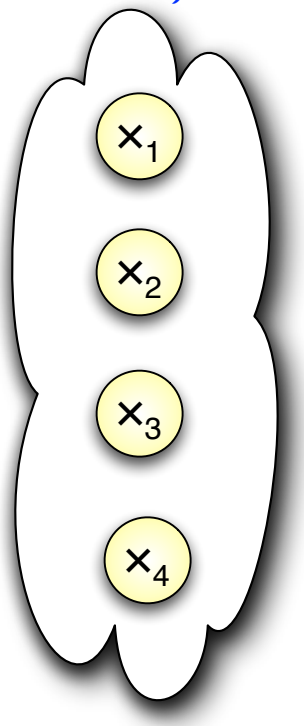
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



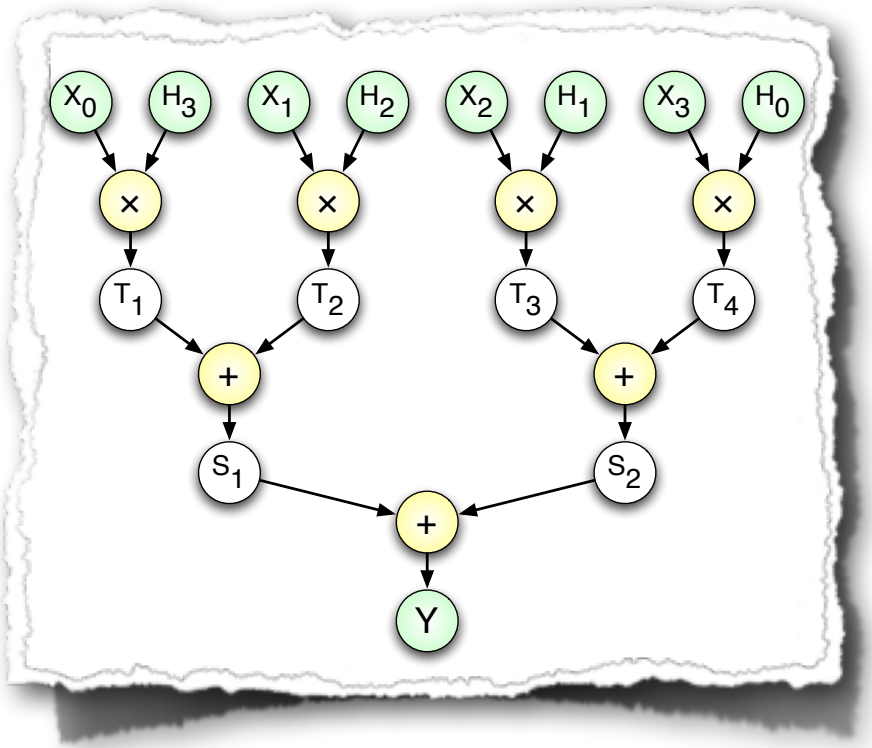
Cycle	Mult (1)	Add (1)			
1					
2					
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

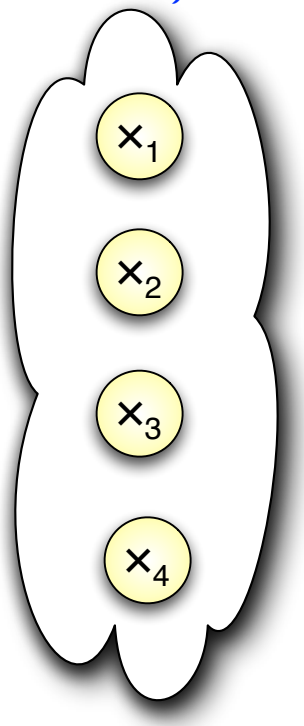
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



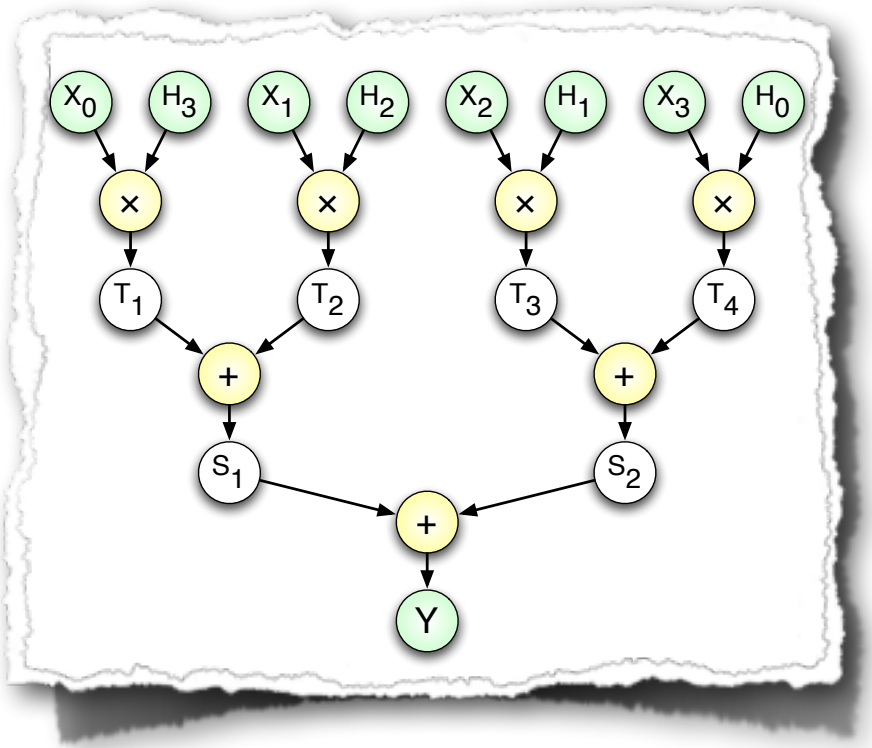
Cycle	Mult (1)	Add (1)			
1					
2					
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

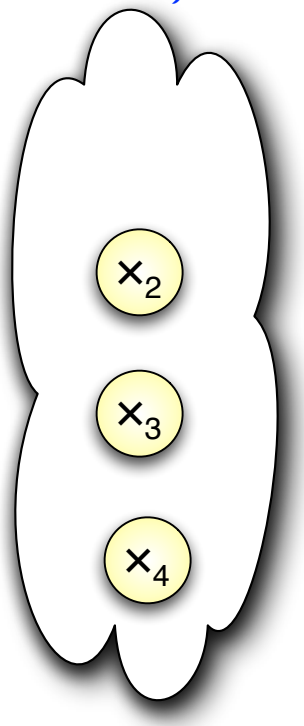
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



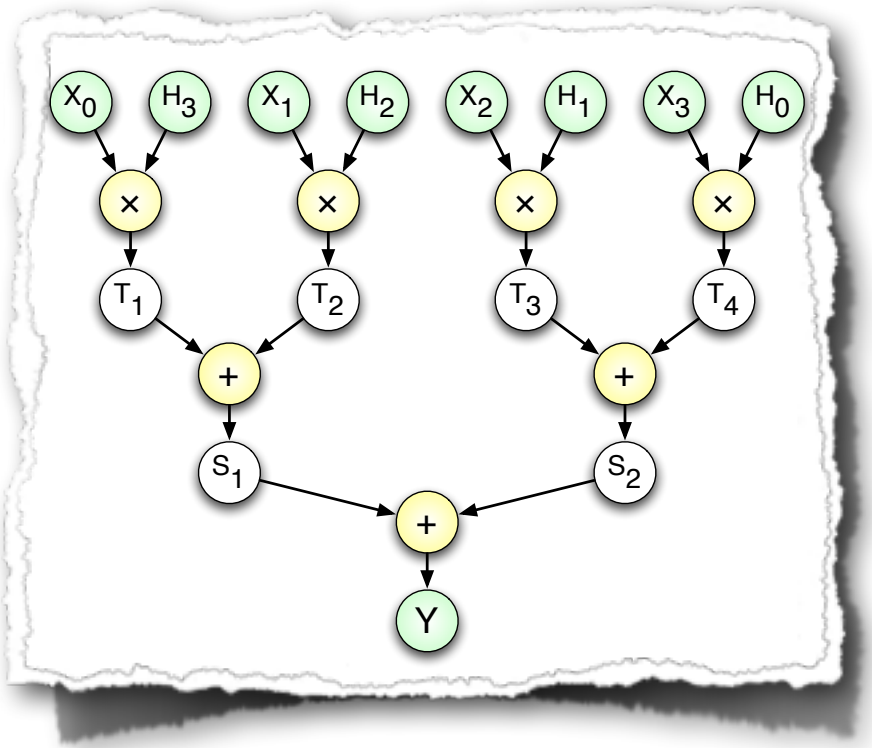
Cycle	Mult (1)	Add (1)			
1	$\times_1$				
2					
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

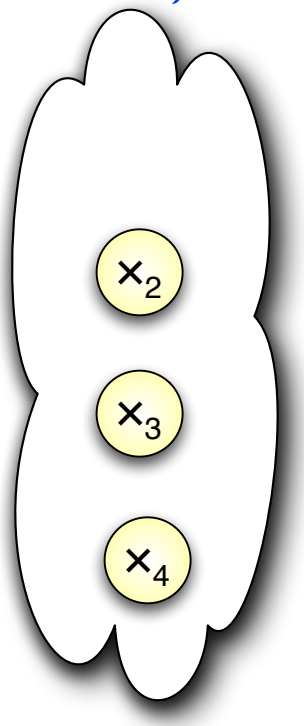
Allocation d'origine => $\mathcal{T}(+) = 1$ et $\mathcal{T}(\times) = 1$

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



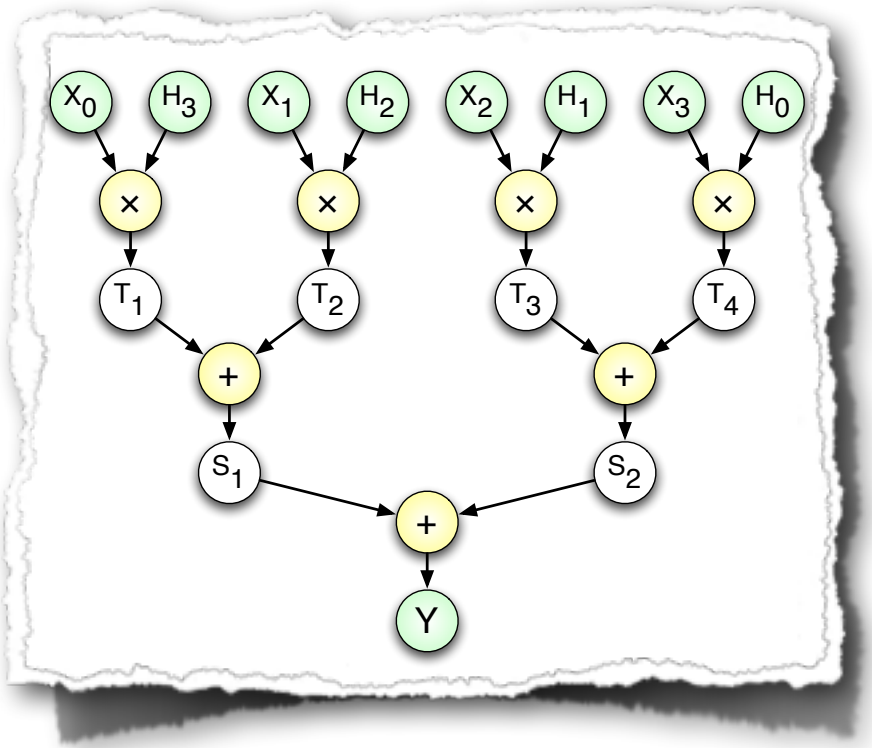
Cycle	Mult (1)	Add (1)			
1	$\times_1$				
2					
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

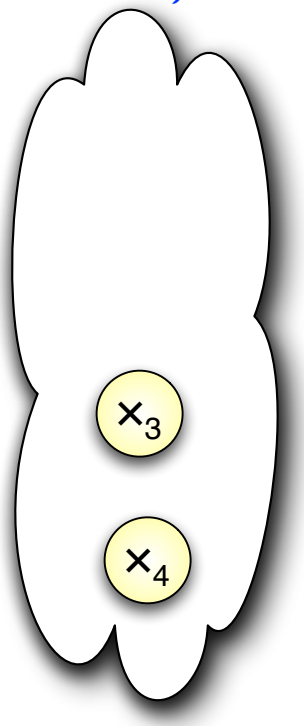
Allocation d'origine => $\mathcal{T}(+) = 1$ et $\mathcal{T}(\times) = 1$

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



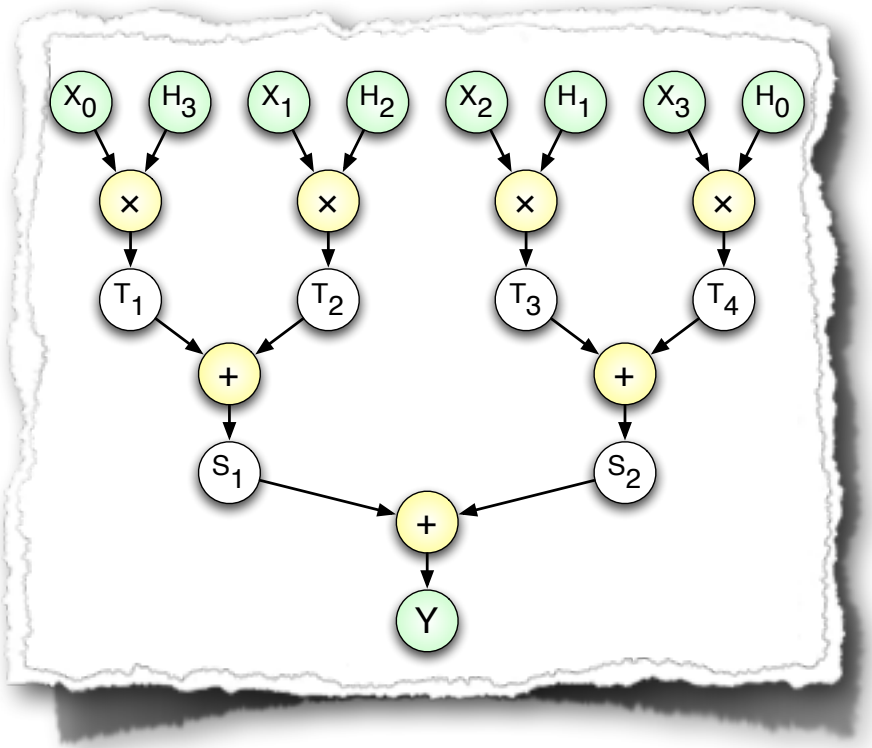
Cycle	Mult (1)	Add (1)			
1	$\times_1$				
2	$\times_2$				
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

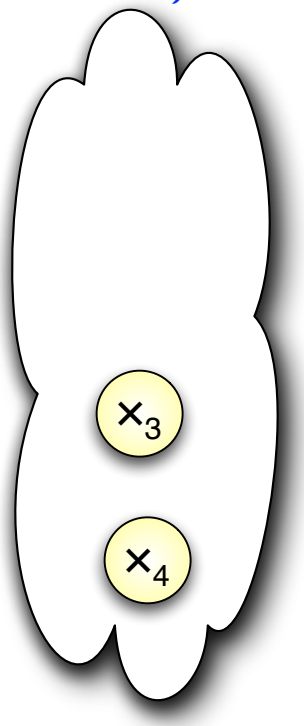
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



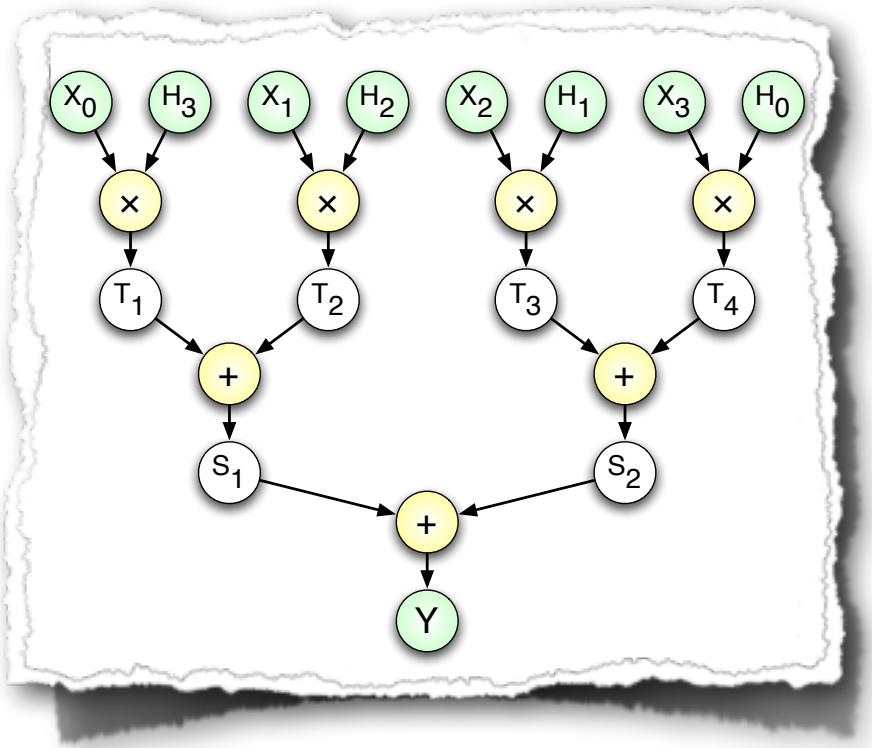
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$				
2	$\times_2$				
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

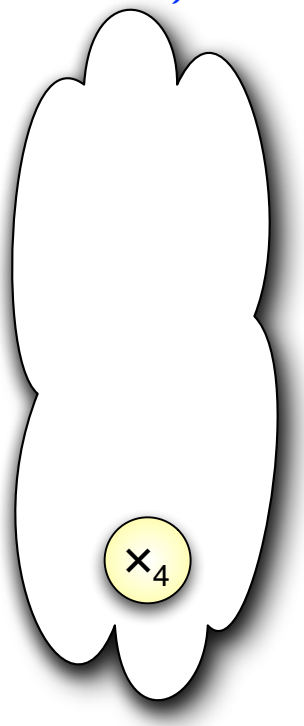
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



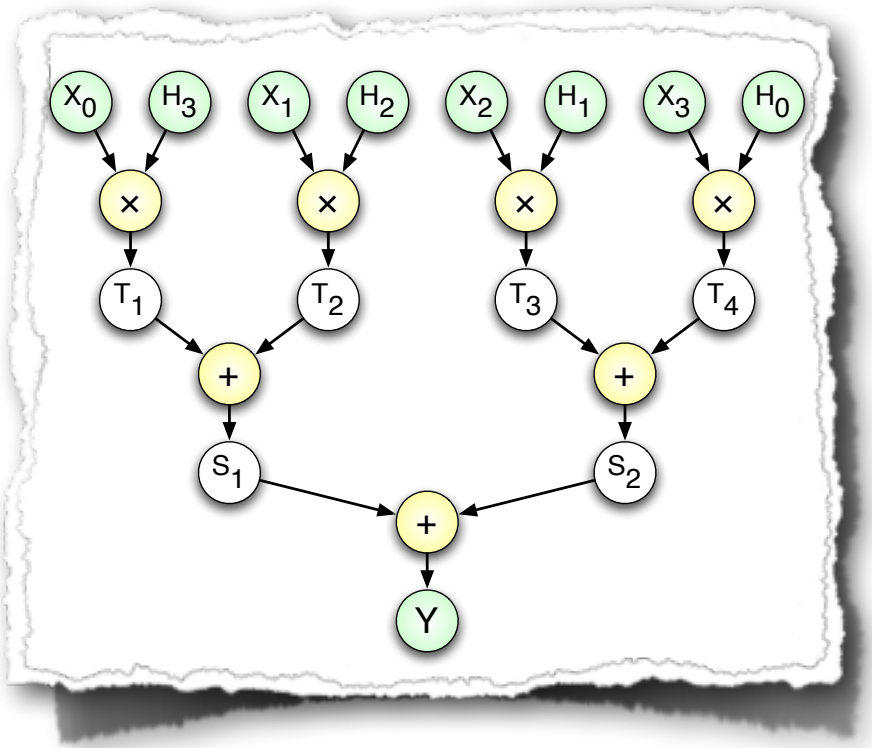
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$				
2	$\times_2$		$\times_3$		
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

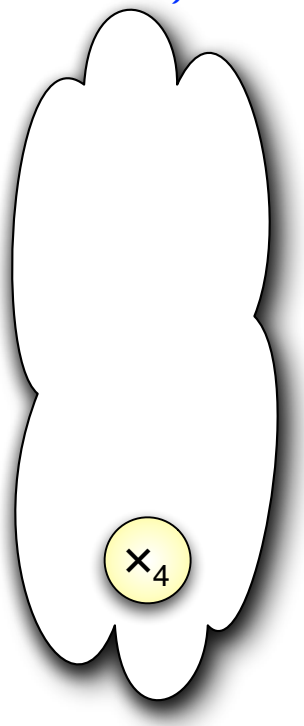
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



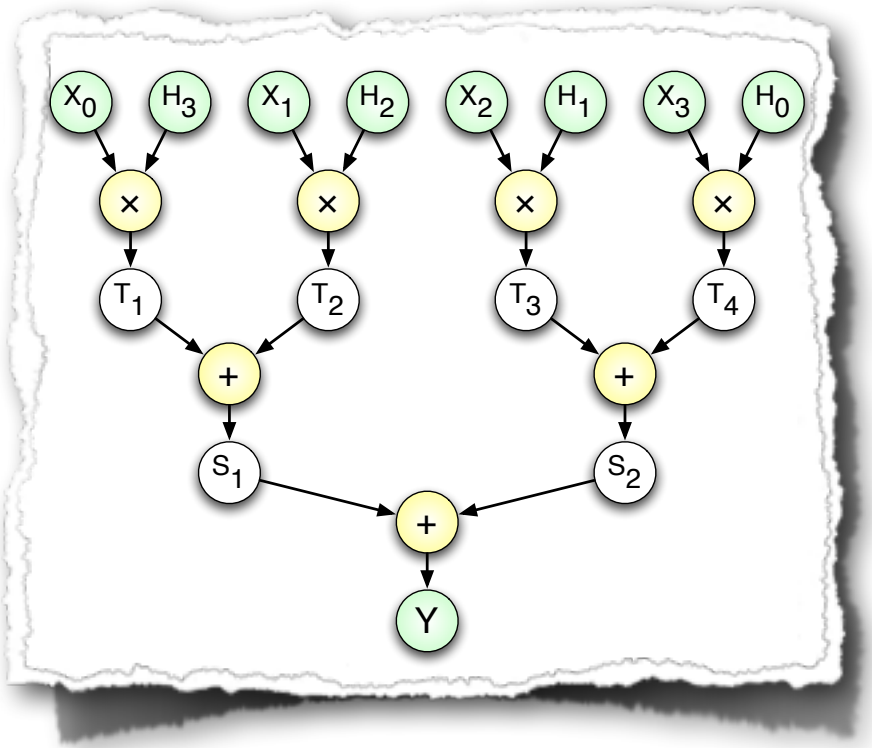
Cycle	Mult (1)	Add (1)	Mult (2)	Mult (3)	
1	$\times_1$				
2	$\times_2$		$\times_3$		
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

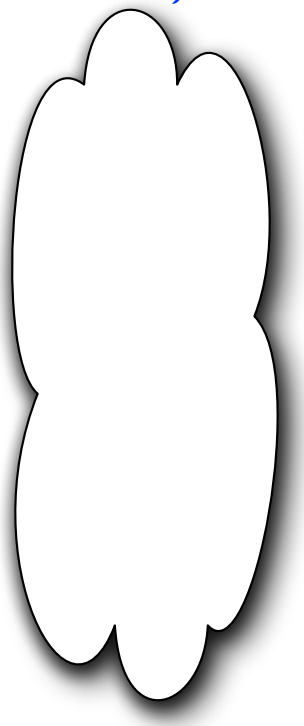
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



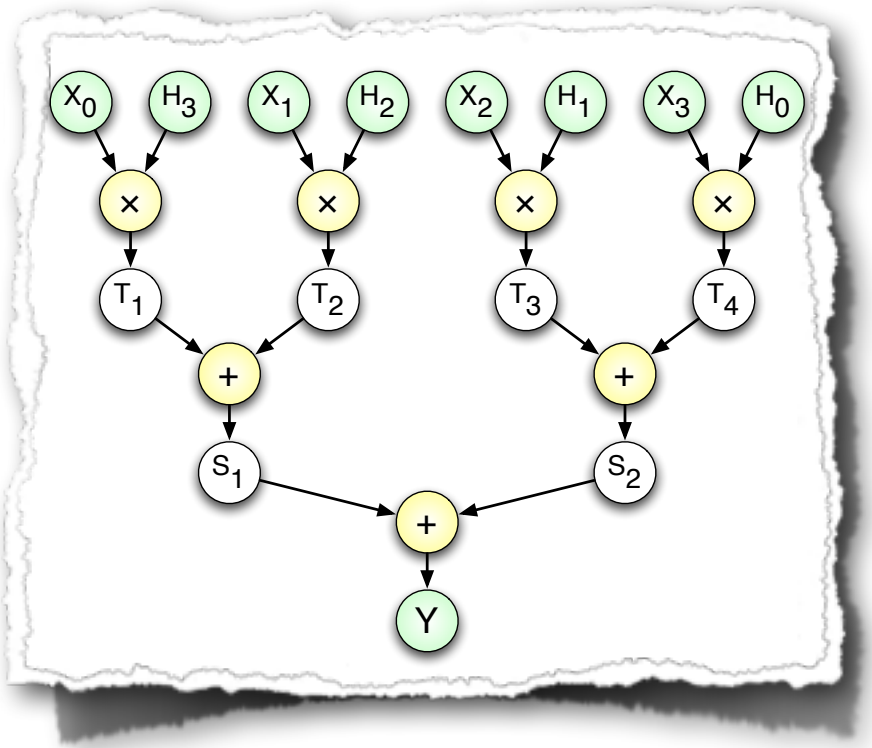
Cycle	Mult (1)	Add (1)	Mult (2)	Mult (3)	
1	$\times_1$				
2	$\times_2$		$\times_3$	$\times_4$	
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

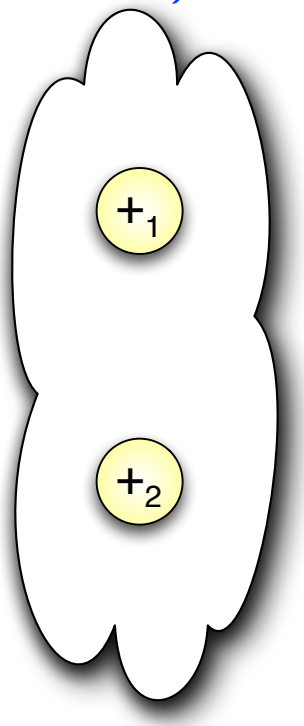
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



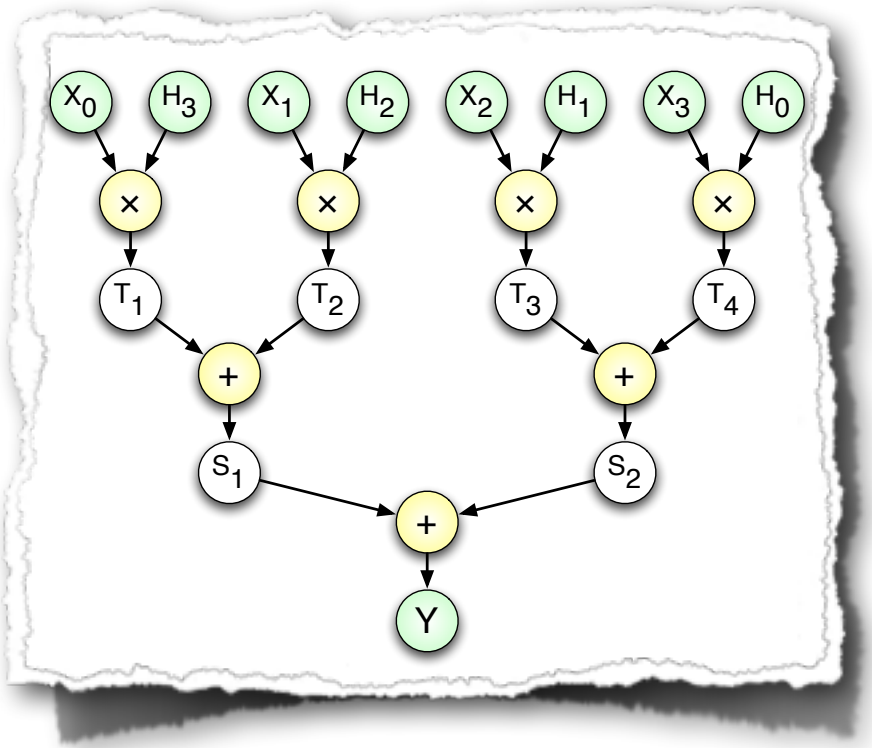
Cycle	Mult (1)	Add (1)	Mult (2)	Mult (3)	
1	$\times_1$				
2	$\times_2$		$\times_3$	$\times_4$	
3					
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

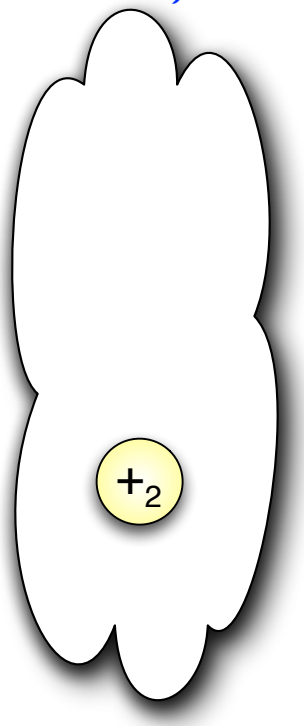
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



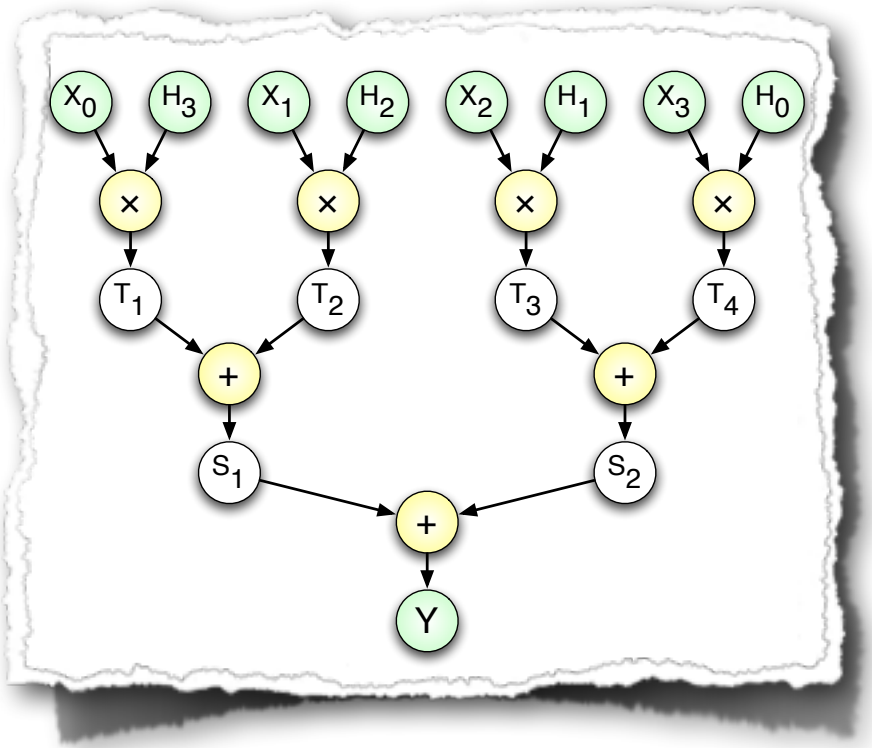
Cycle	Mult (1)	Add (1)	Mult (2)	Mult (3)	
1	$\times_1$				
2	$\times_2$		$\times_3$	$\times_4$	
3		$+_1$			
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

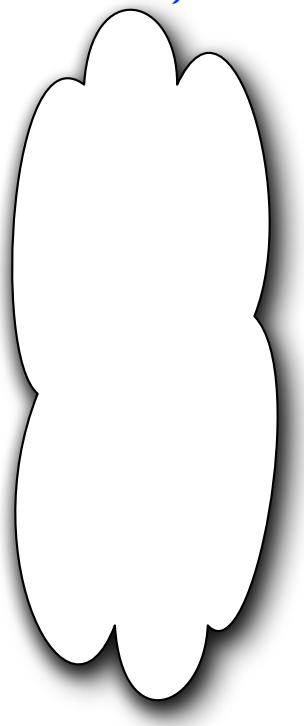
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



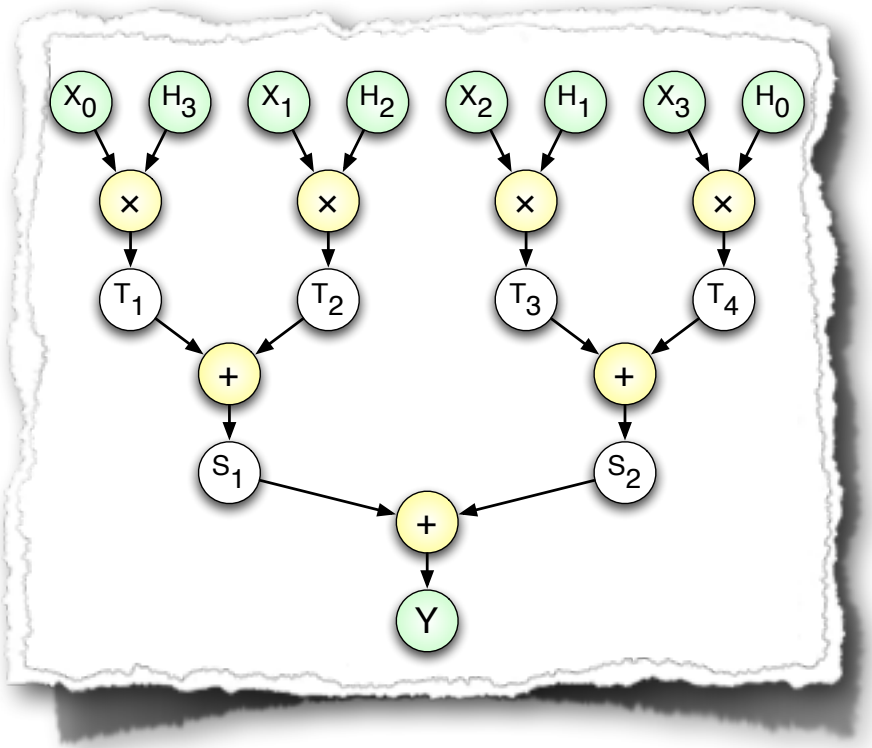
Cycle	Mult (1)	Add (1)	Mult (2)	Mult (3)	
1	$\times_1$				
2	$\times_2$		$\times_3$	$\times_4$	
3		$+_1$			$+_2$
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

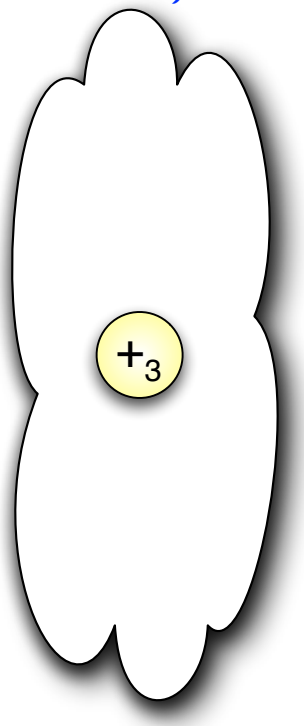
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



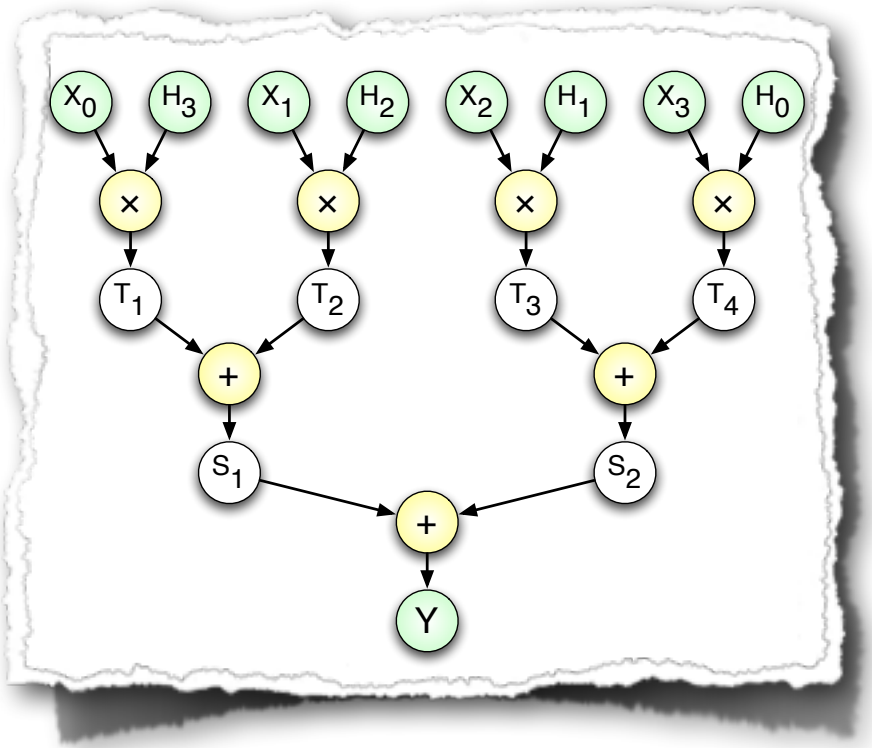
Cycle	Mult (1)	Add (1)	Mult (2)	Mult (3)	
1	$\times_1$				
2	$\times_2$		$\times_3$	$\times_4$	
3		$+_1$			$+_2$
4					

Ordonnancement (temporel) sans itération

La contrainte temporelle => 4 cycles maxi

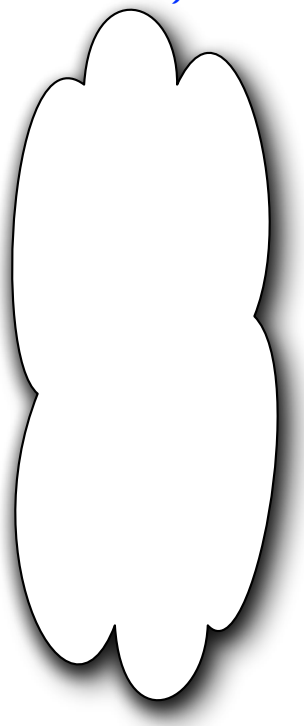
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement sans itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



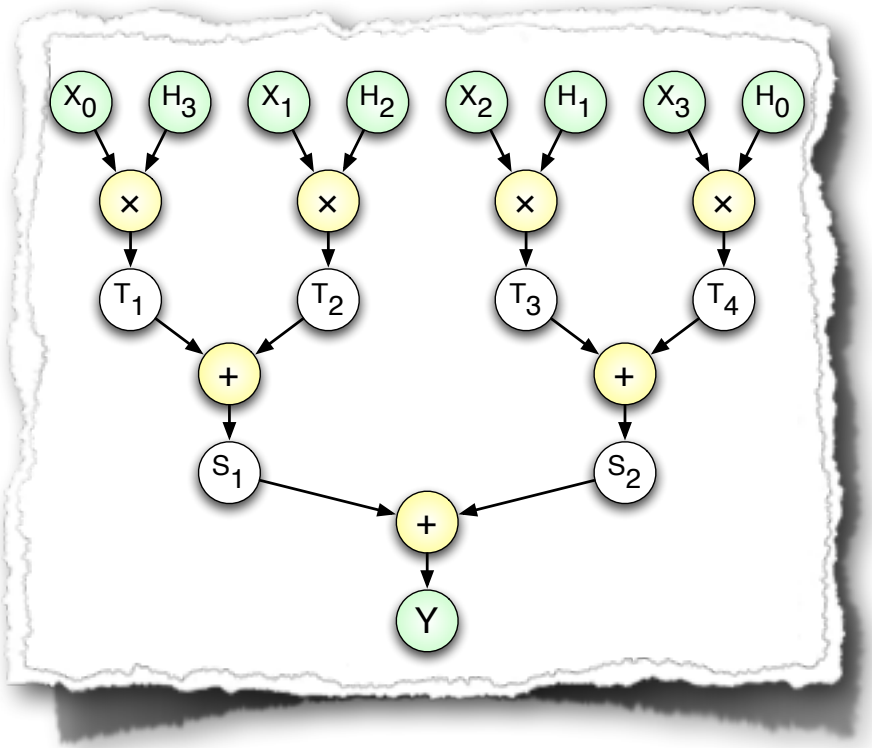
Cycle	Mult (1)	Add (1)	Mult (2)	Mult (3)	
1	$\times_1$				
2	$\times_2$		$\times_3$	$\times_4$	
3		$+_1$			$+_2$
4		$+_3$			

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

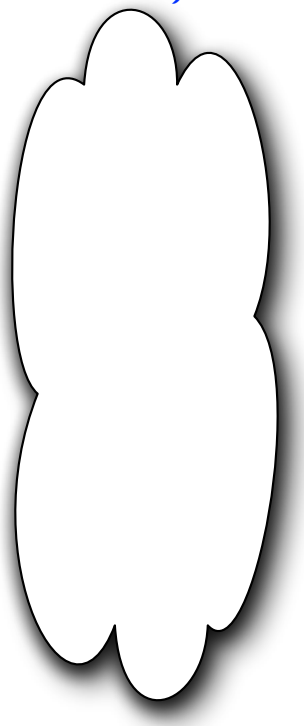
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



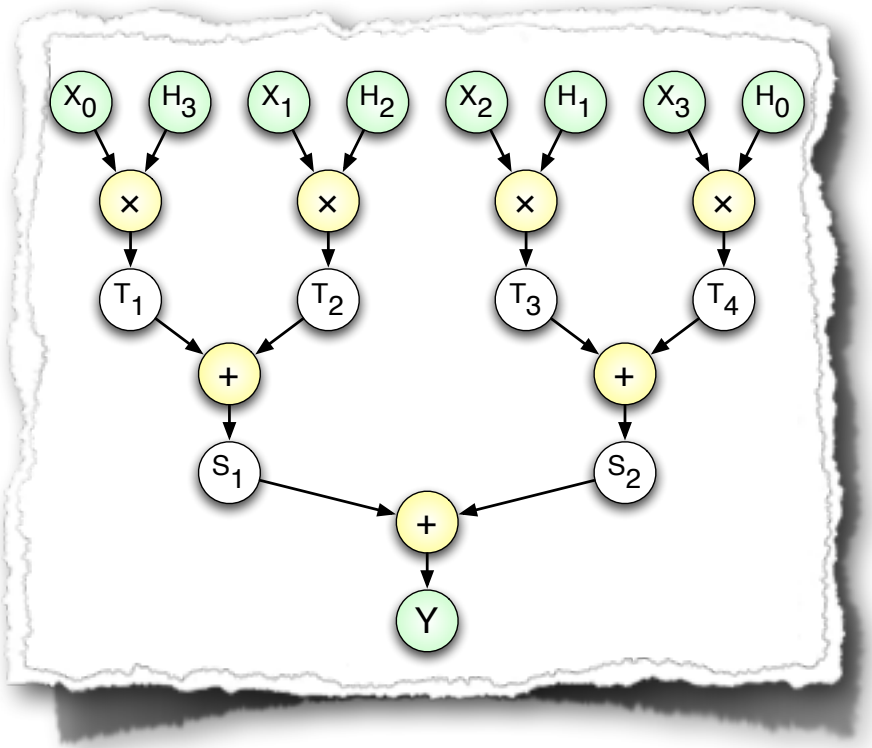
Cycle	Mult (1)	Add (1)			
1					
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

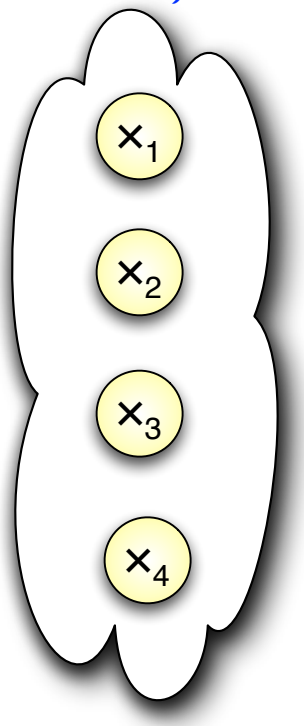
Allocation d'origine => $\mathcal{T}(+) = 1$ et $\mathcal{T}(\times) = 1$

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



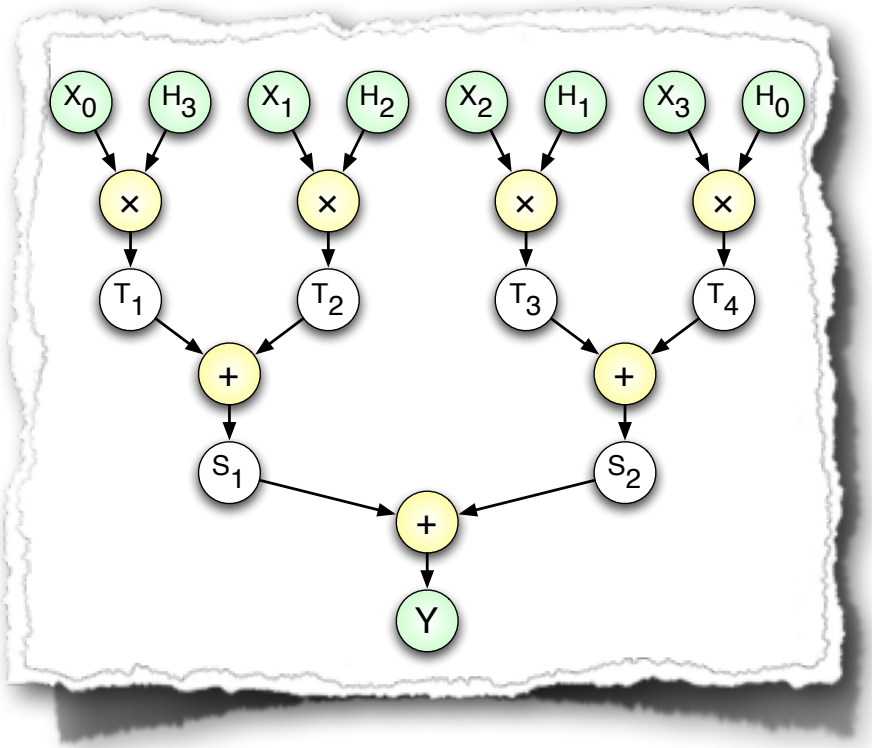
Cycle	Mult (1)	Add (1)			
1					
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

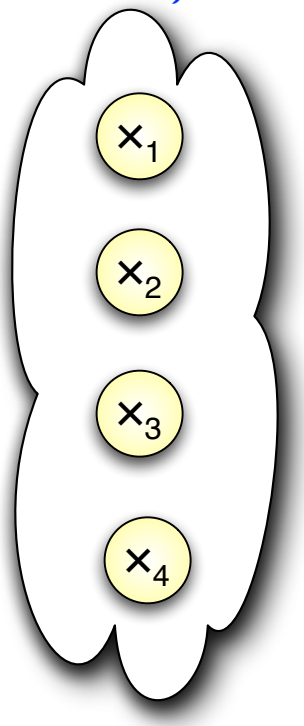
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



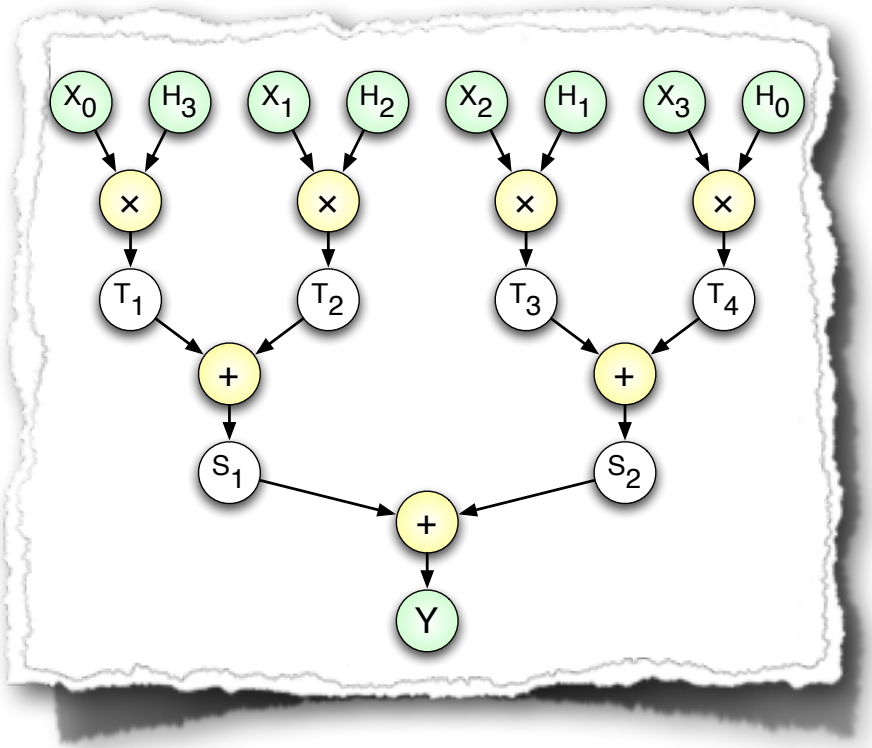
Cycle	Mult (1)	Add (1)			
1					
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

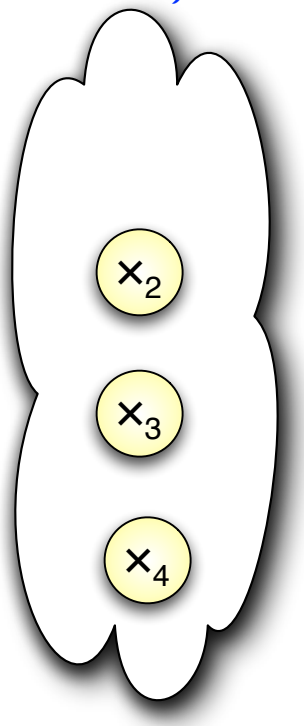
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



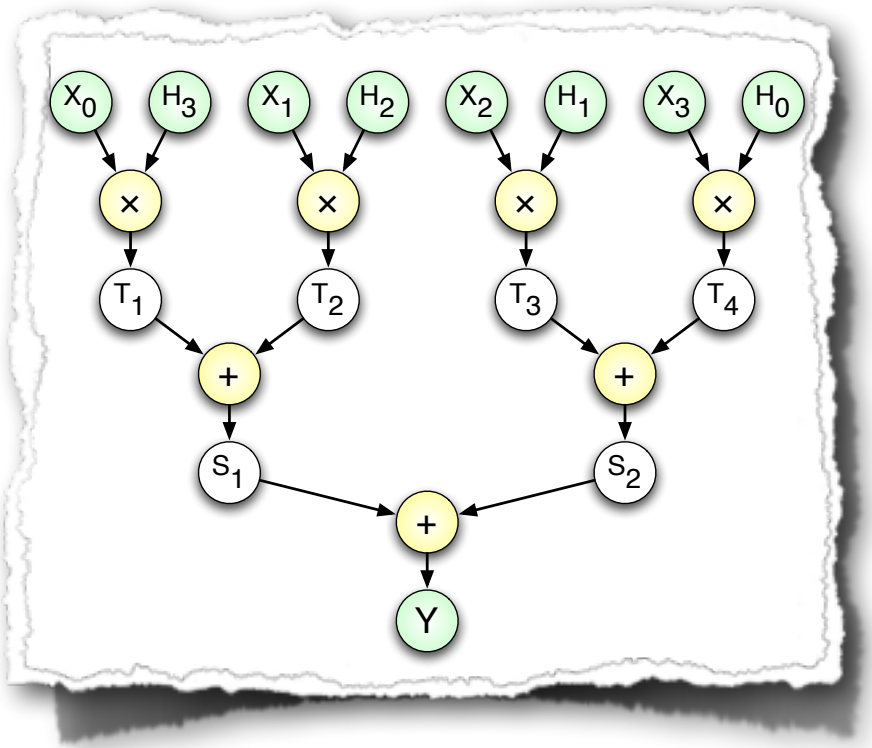
Cycle	Mult (1)	Add (1)			
1	$\times_1$				
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

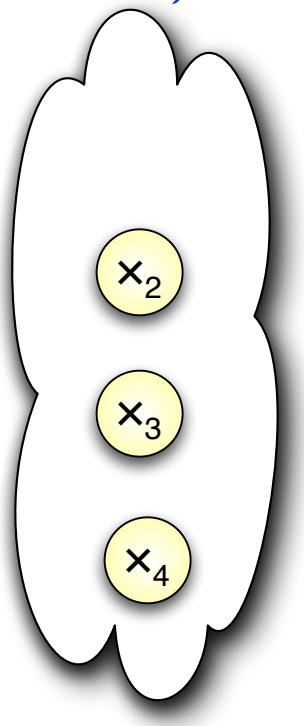
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



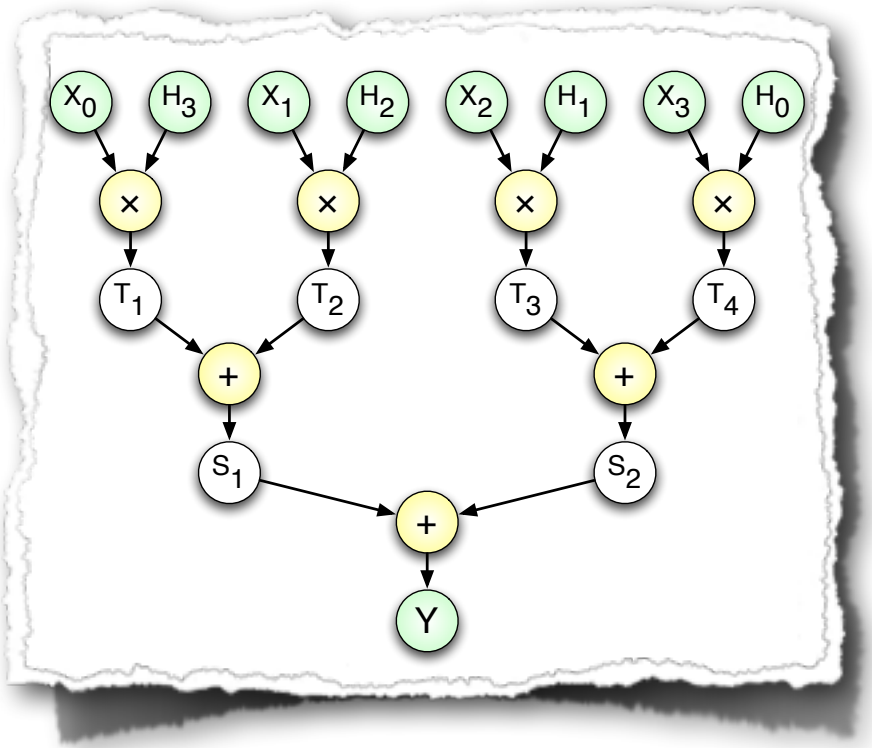
Cycle	Mult (1)	Add (1)			
1	$\times_1$				
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

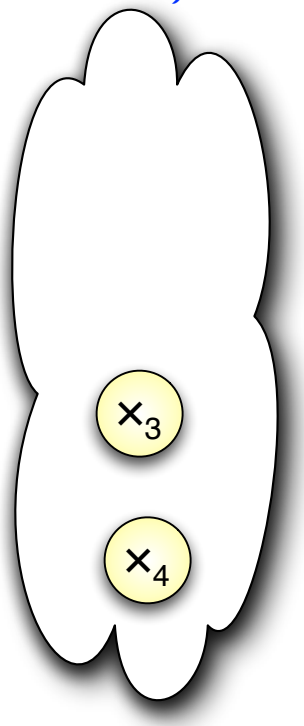
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



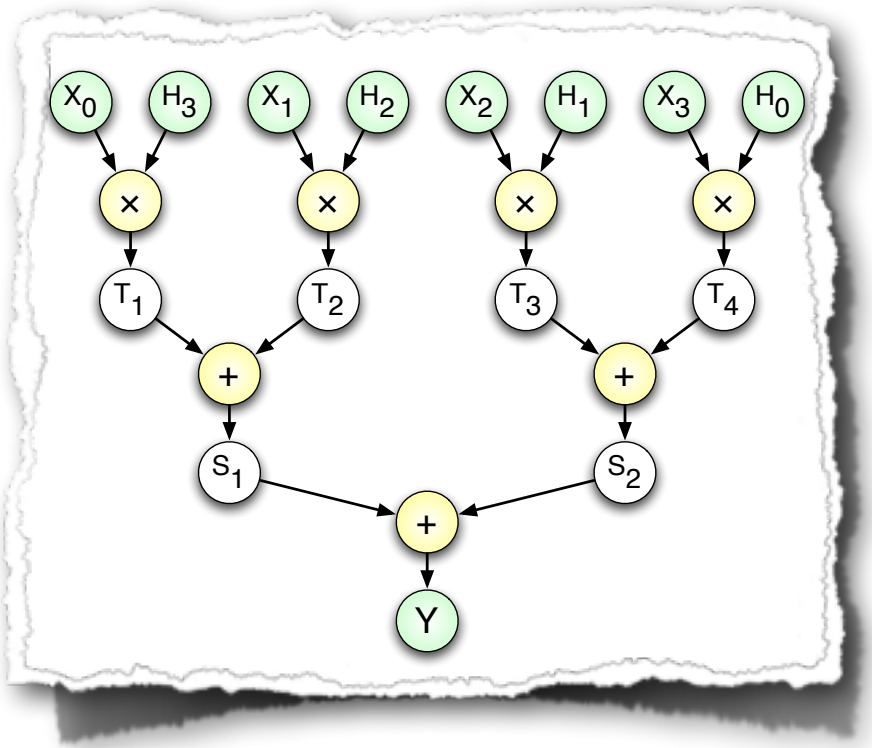
Cycle	Mult (1)	Add (1)			
1	$\times_1$				
2	$\times_2$				
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

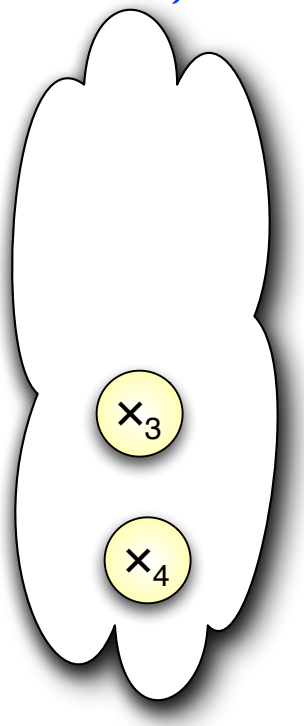
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



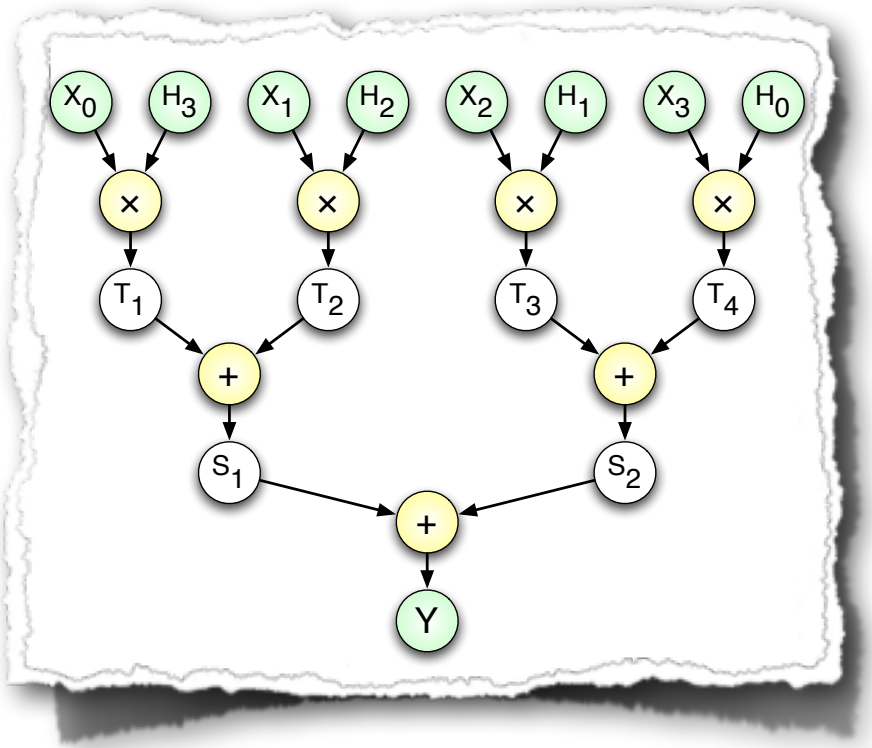
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$				
2	$\times_2$				
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

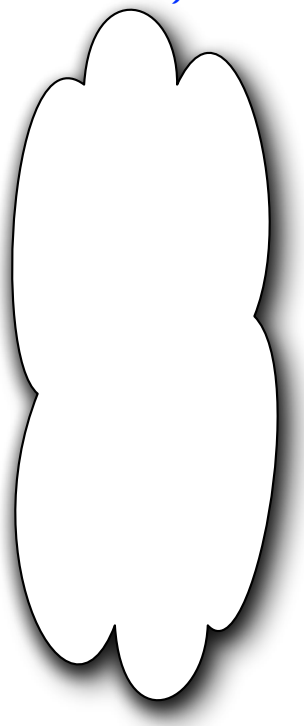
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
x_1	0	1	1	3
x_2	0	1	1	3
x_3	0	1	1	3
x_4	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



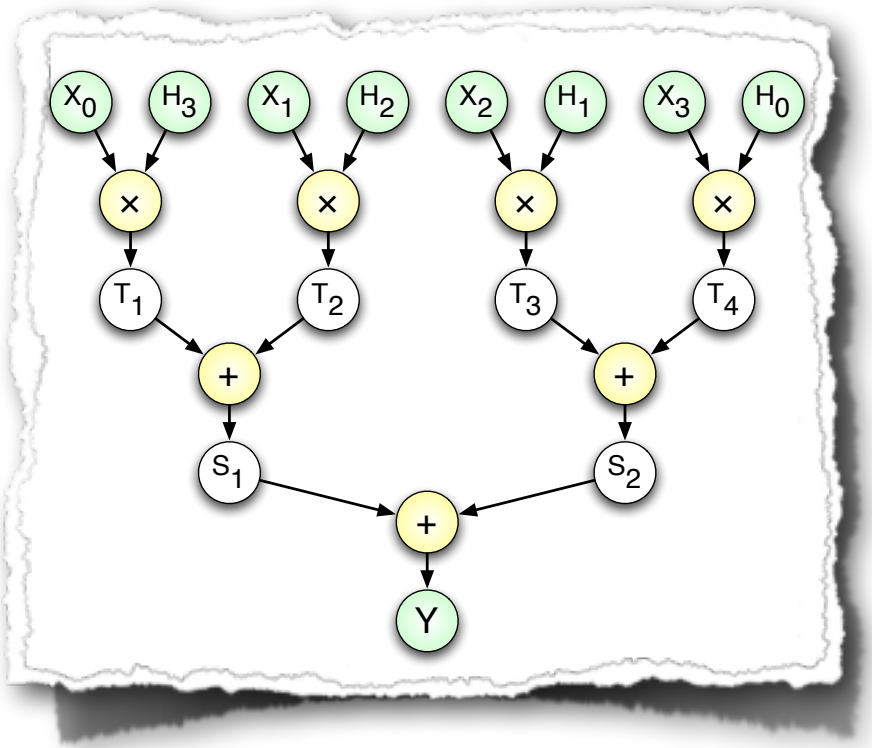
Cycle	Mult (1)	Add (1)	Mult (2)		
1					
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

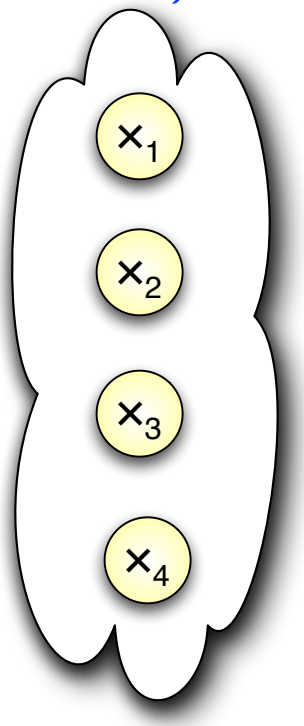
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
x_1	0	1	1	3
x_2	0	1	1	3
x_3	0	1	1	3
x_4	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



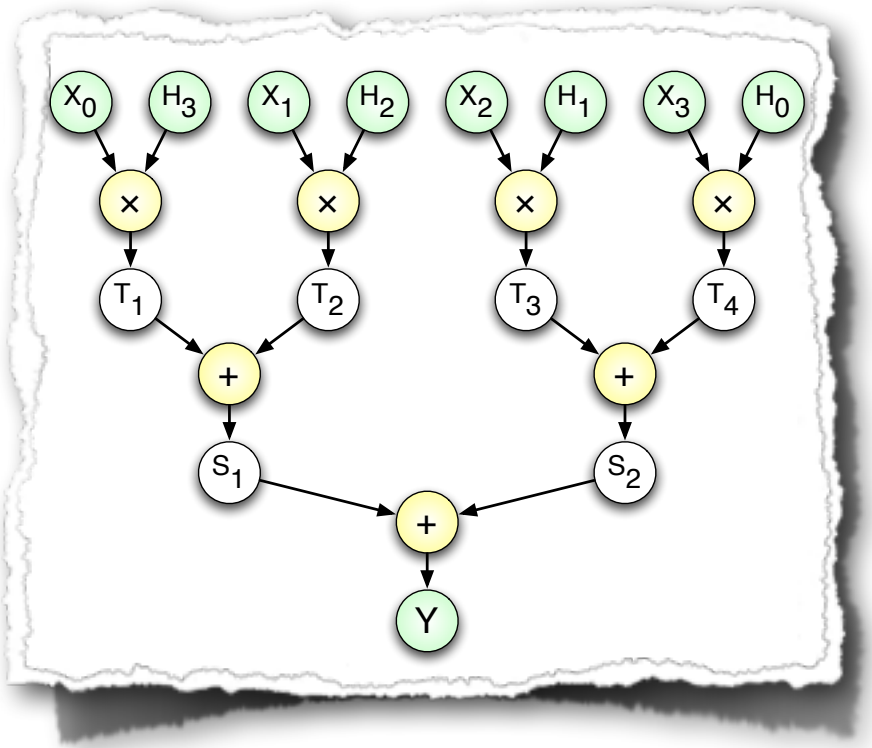
Cycle	Mult (1)	Add (1)	Mult (2)		
1					
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

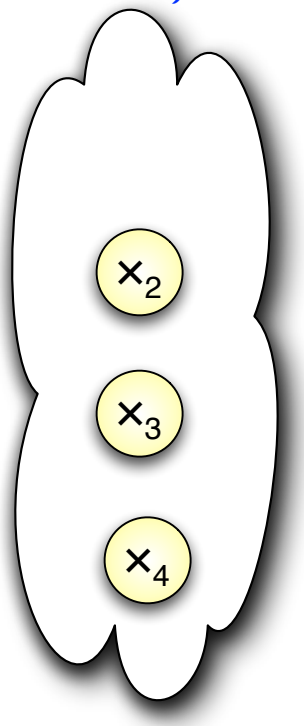
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



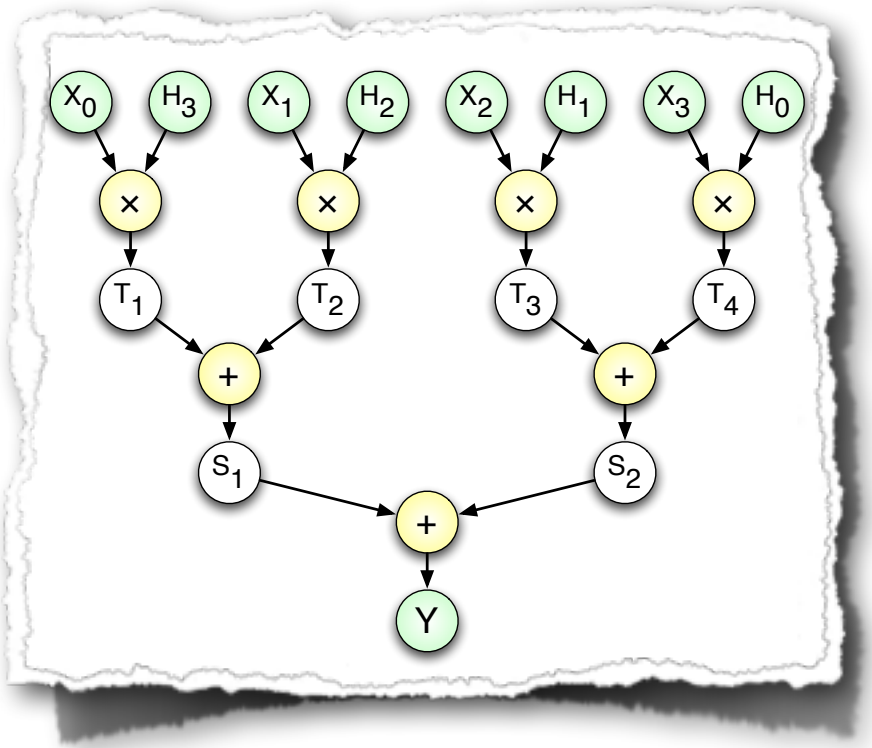
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$				
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

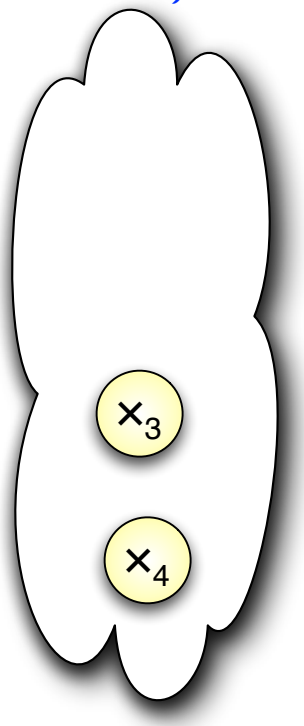
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



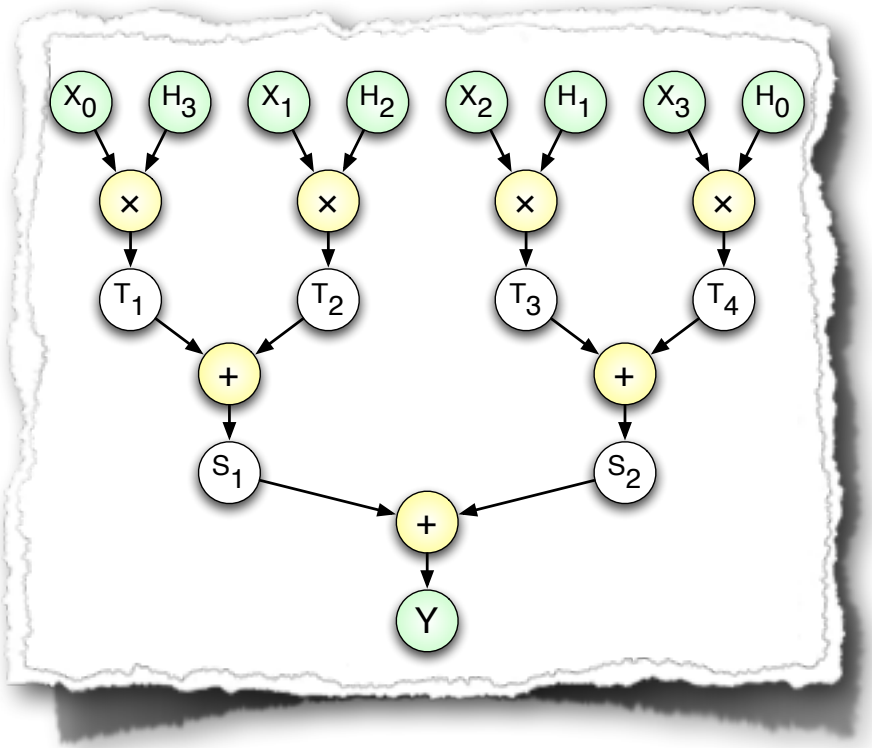
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

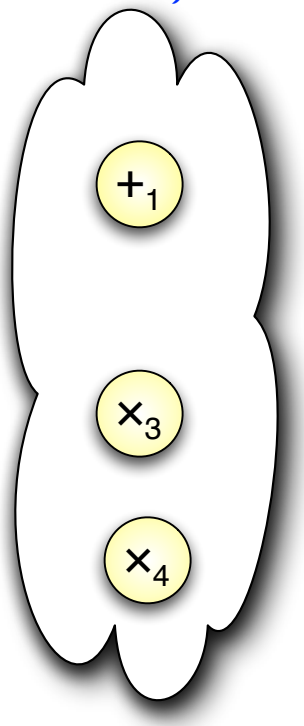
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



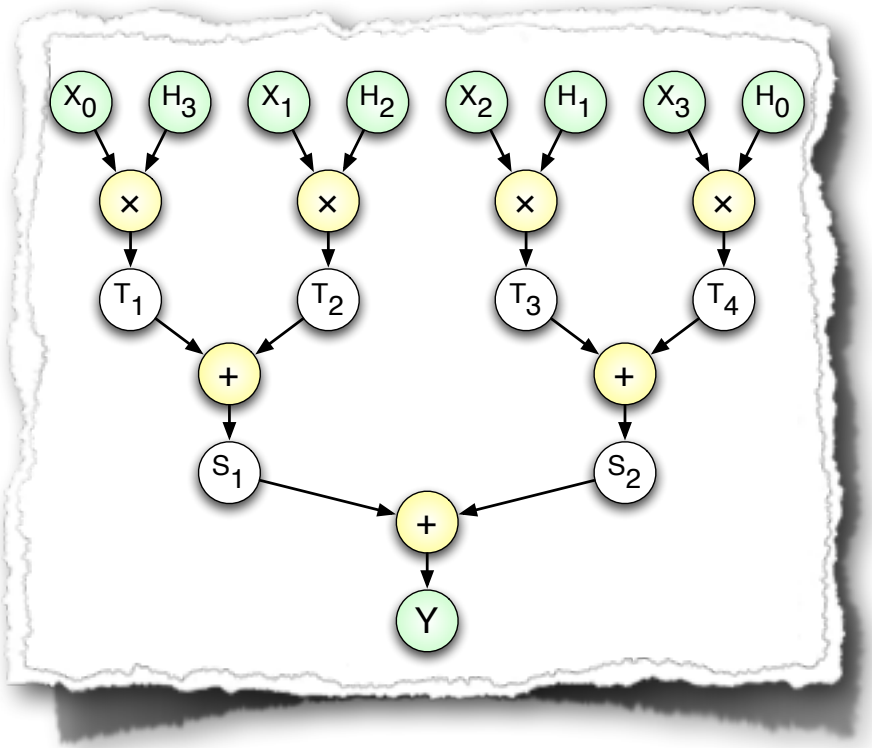
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2					
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

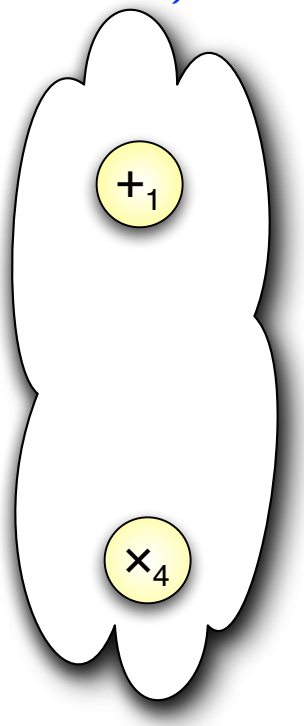
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



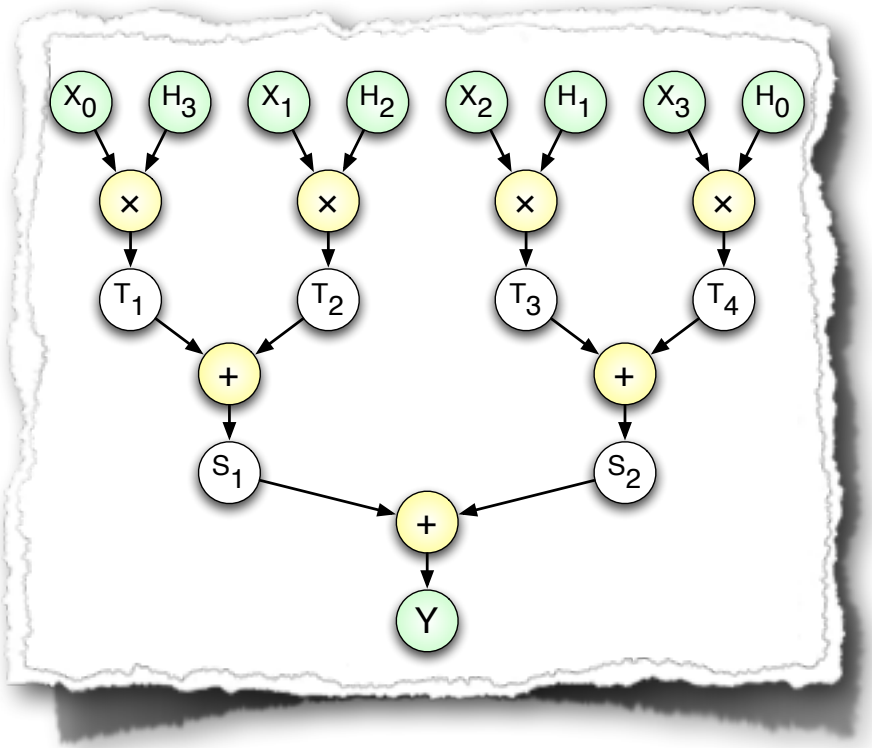
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2	$\times_3$				
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

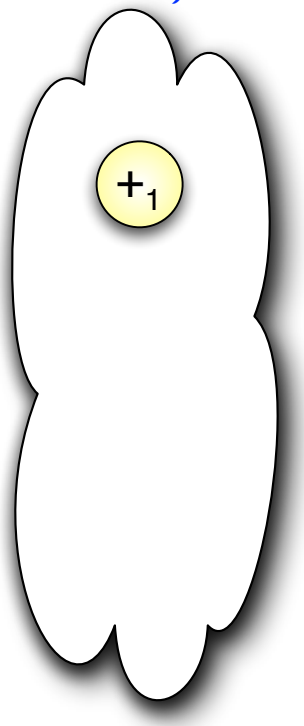
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



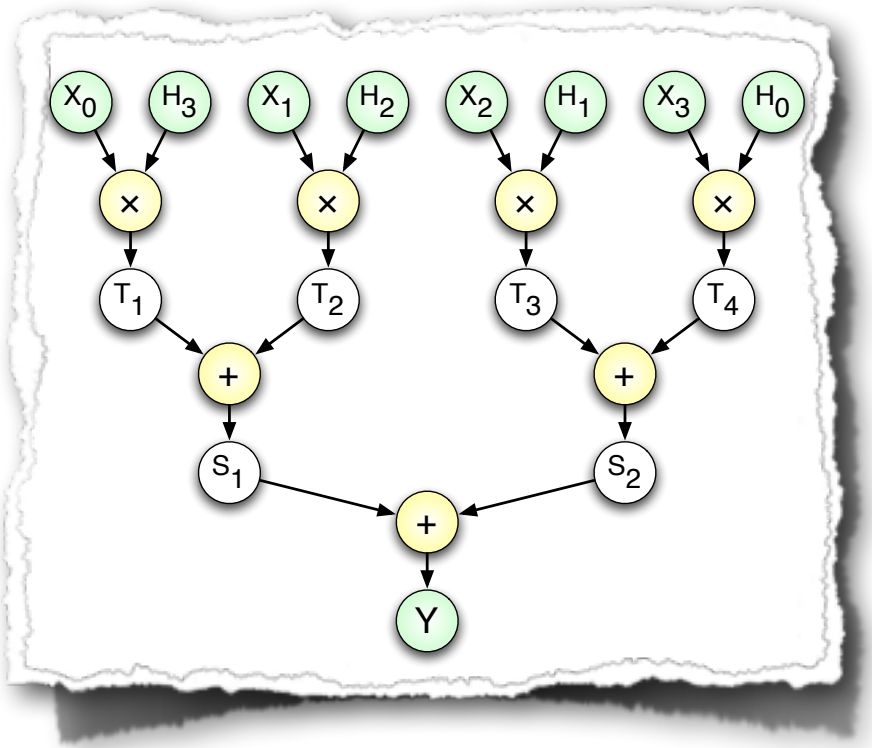
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2	$\times_3$		$\times_4$		
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables

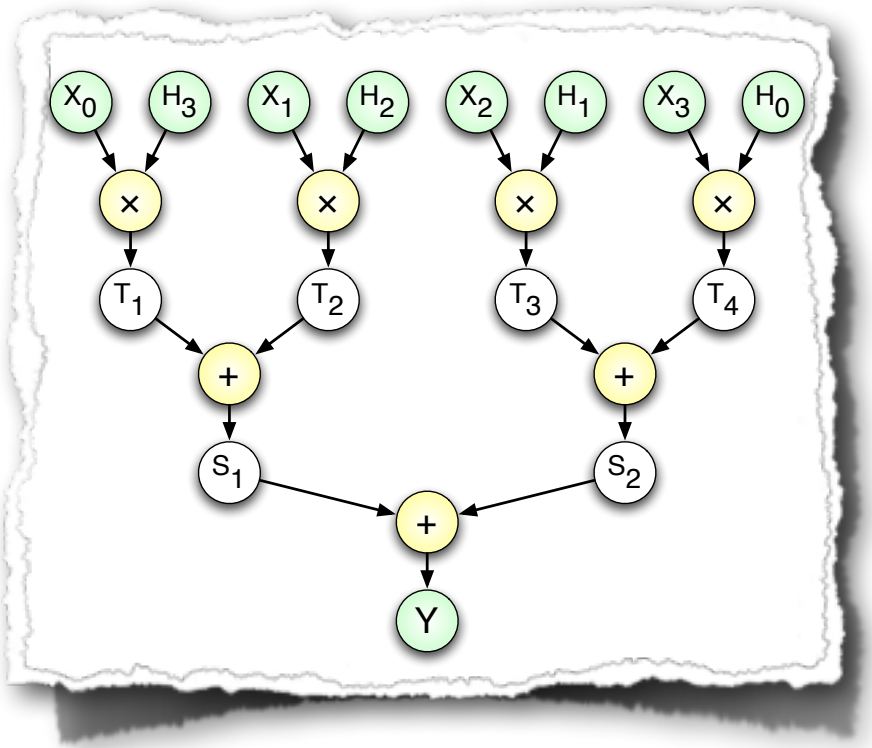
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2	$\times_3$	$+_1$	$\times_4$		
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

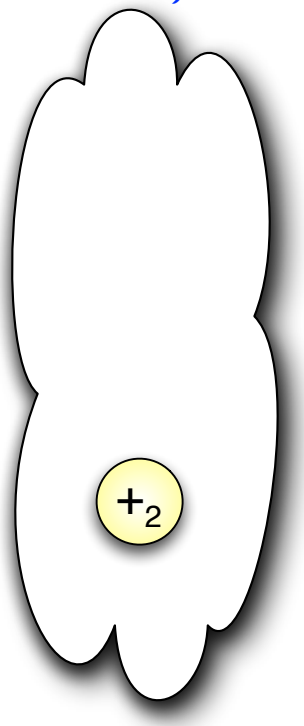
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



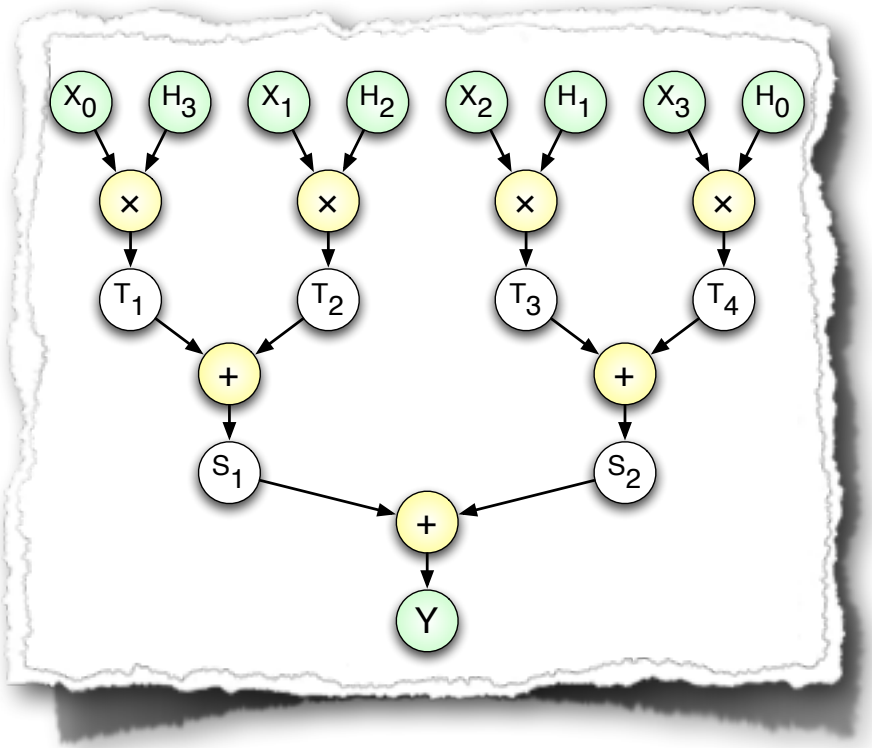
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2	$\times_3$	$+_1$	$\times_4$		
3					
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

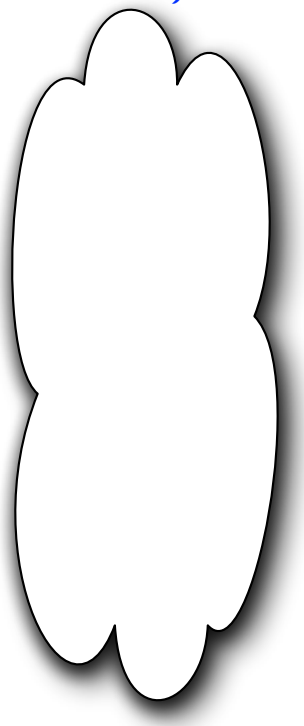
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



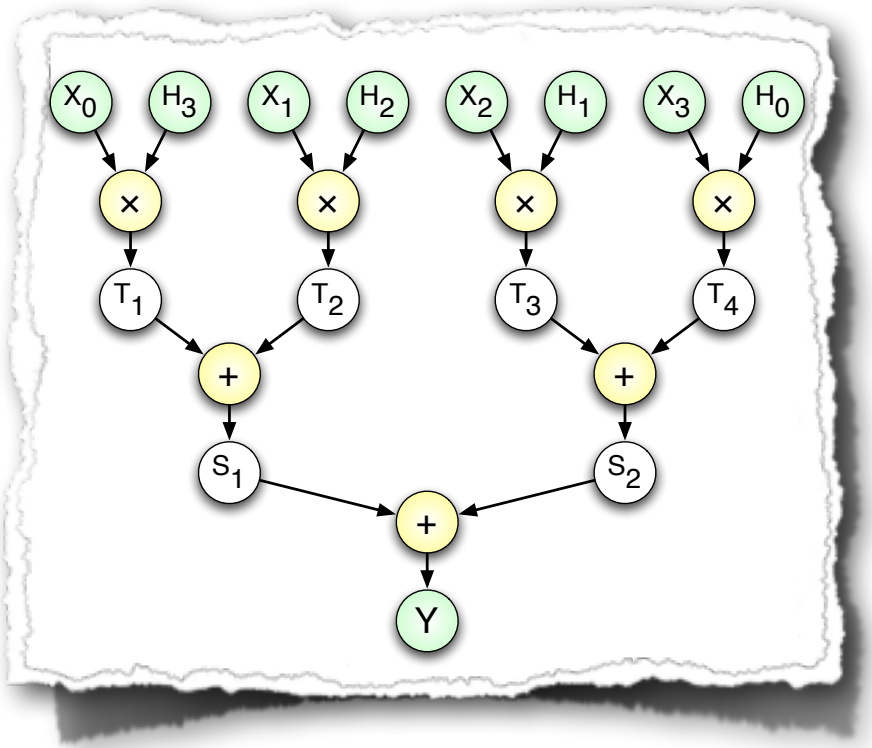
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2	$\times_3$	$+_1$	$\times_4$		
3		$+_2$			
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

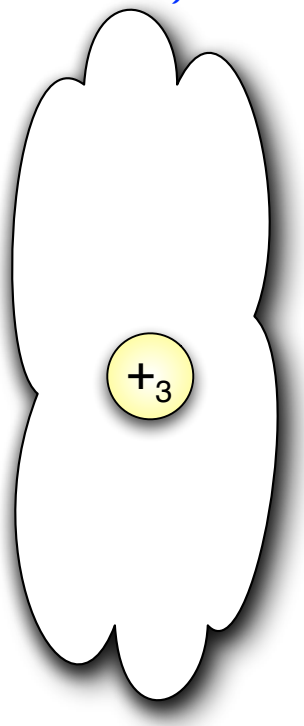
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables



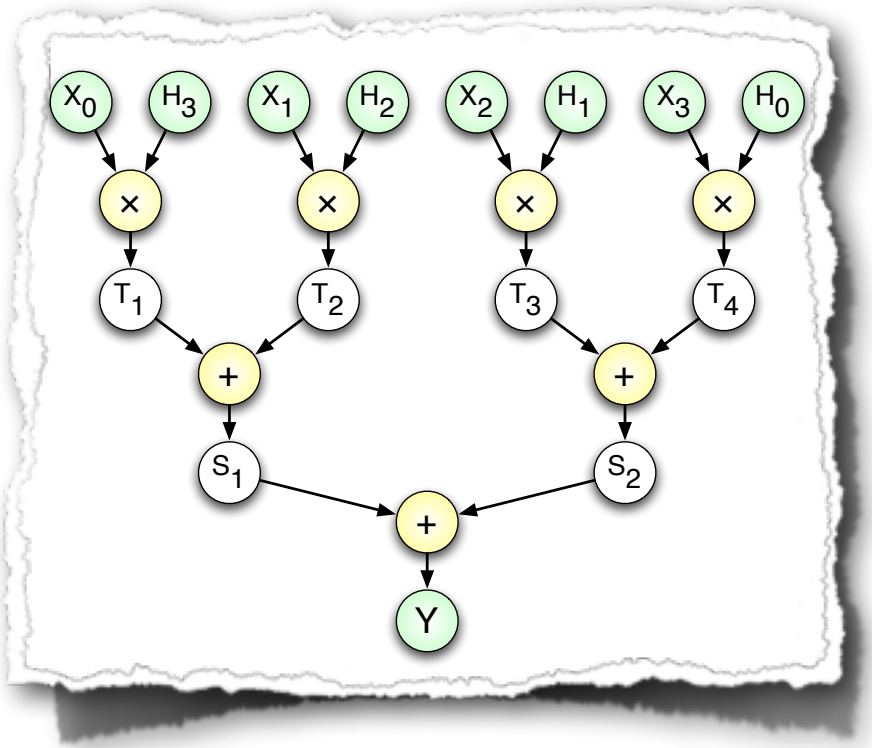
Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2	$\times_3$	$+_1$	$\times_4$		
3		$+_2$			
4					

Ordonnancement (temporel) avec itérations

La contrainte temporelle => 4 cycles maxi

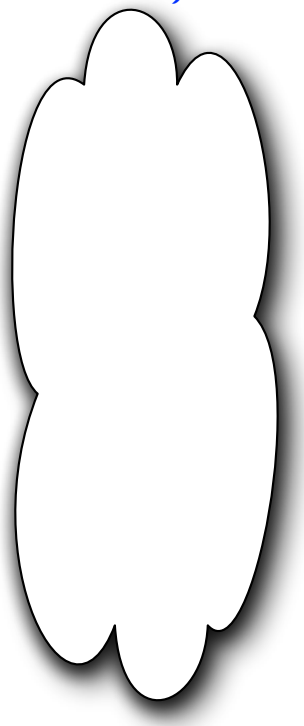
Allocation d'origine => T(+) = 1 et T(x) = 1

Ordonnancement avec itération en cas d'ajout d'une nouvelle ressource.



Noeud	ASAP	ALAP	Mobilité	Urgence
$\times_1$	0	1	1	3
$\times_2$	0	1	1	3
$\times_3$	0	1	1	3
$\times_4$	0	1	1	3
$+_1$	1	2	1	2
$+_2$	1	2	1	2
$+_3$	2	3	1	1
Y	3	4	1	0

Opérations ordonnançables

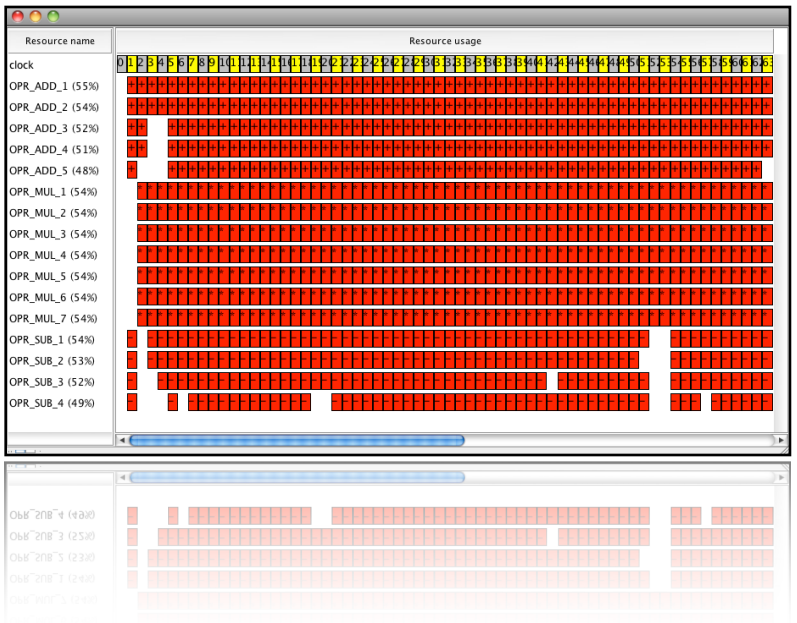


Cycle	Mult (1)	Add (1)	Mult (2)		
1	$\times_1$		$\times_2$		
2	$\times_3$	$+_1$	$\times_4$		
3		$+_2$			
4		$+_3$			

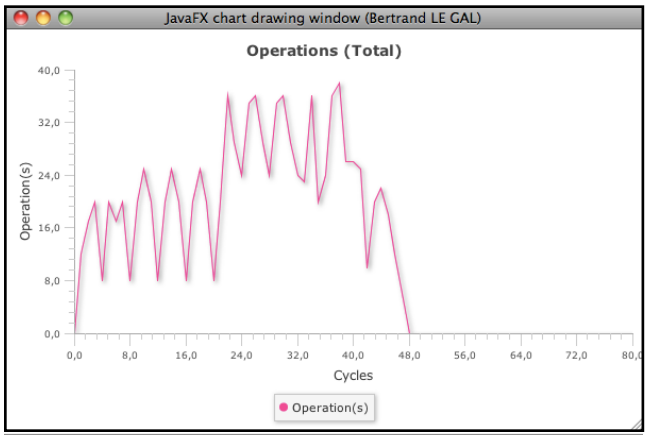
Exemple de la 2d-DCT 8x8 (Ordonnancement)

GANTT opérations (80 cycles)

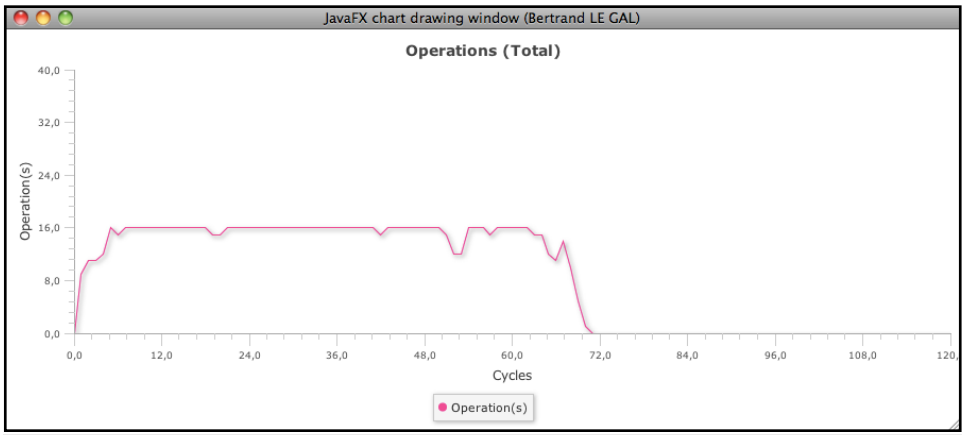
GANTT opérations (120 cycles)



Exemple de la 2d-DCT 8x8 (80 cycles versus 120 cycles)

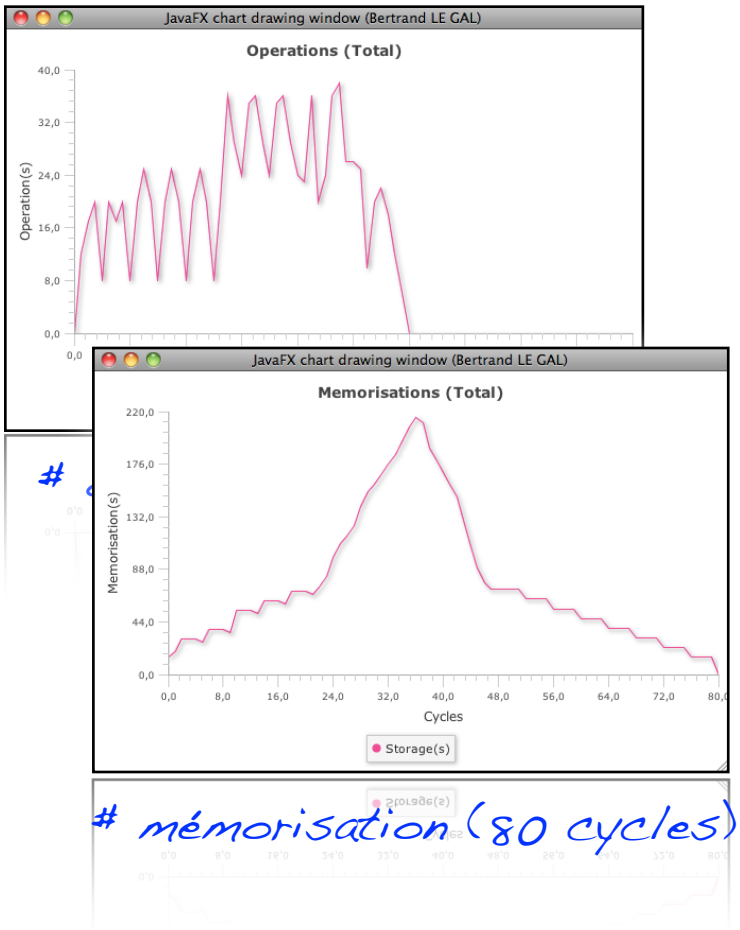


opérations (80 cycles)

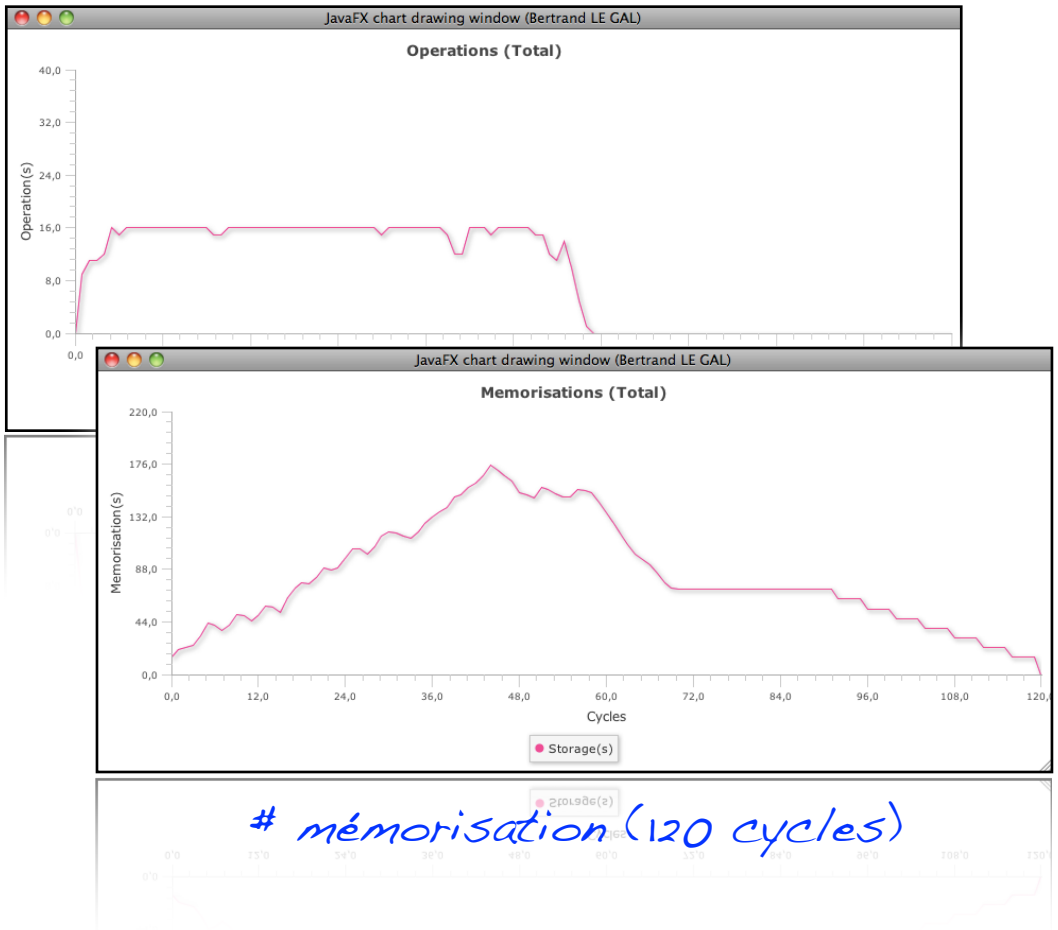


opérations (120 cycles)

Exemple de la 2d-DCT 8x8 (80 cycles versus 120 cycles)

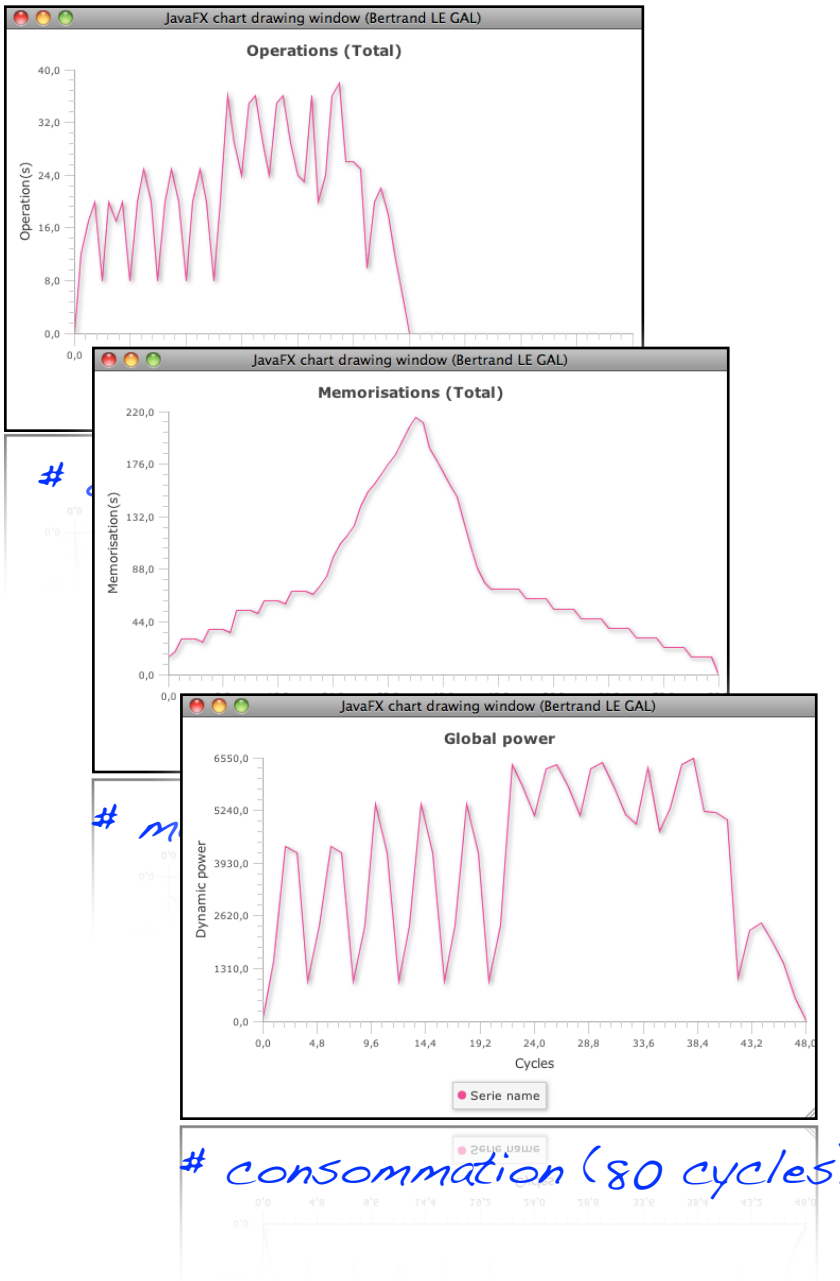


mémorisation (80 cycles)

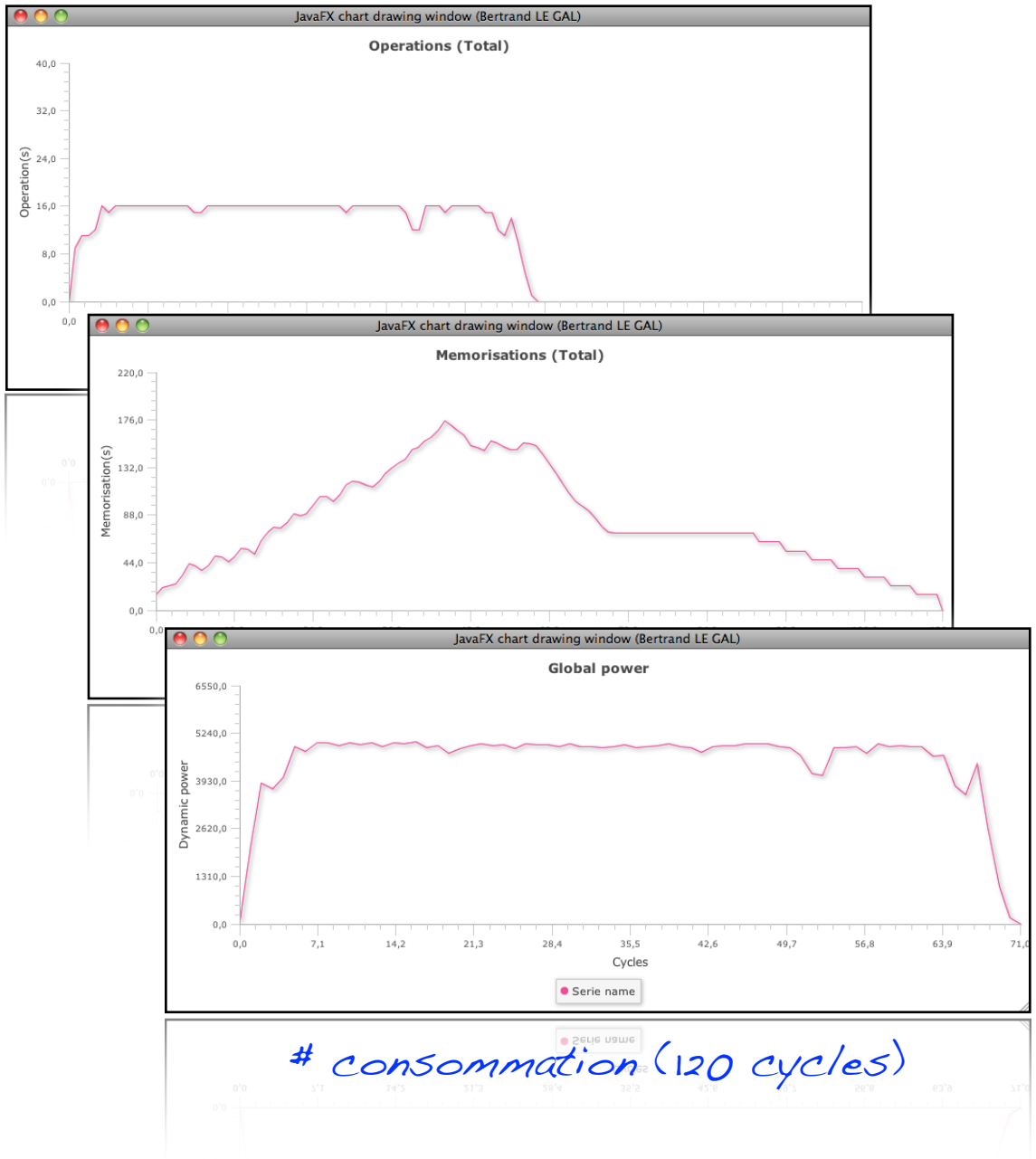


mémorisation (120 cycles)

Exemple de la 2d-DCT 8x8 (80 cycles versus 120 cycles)

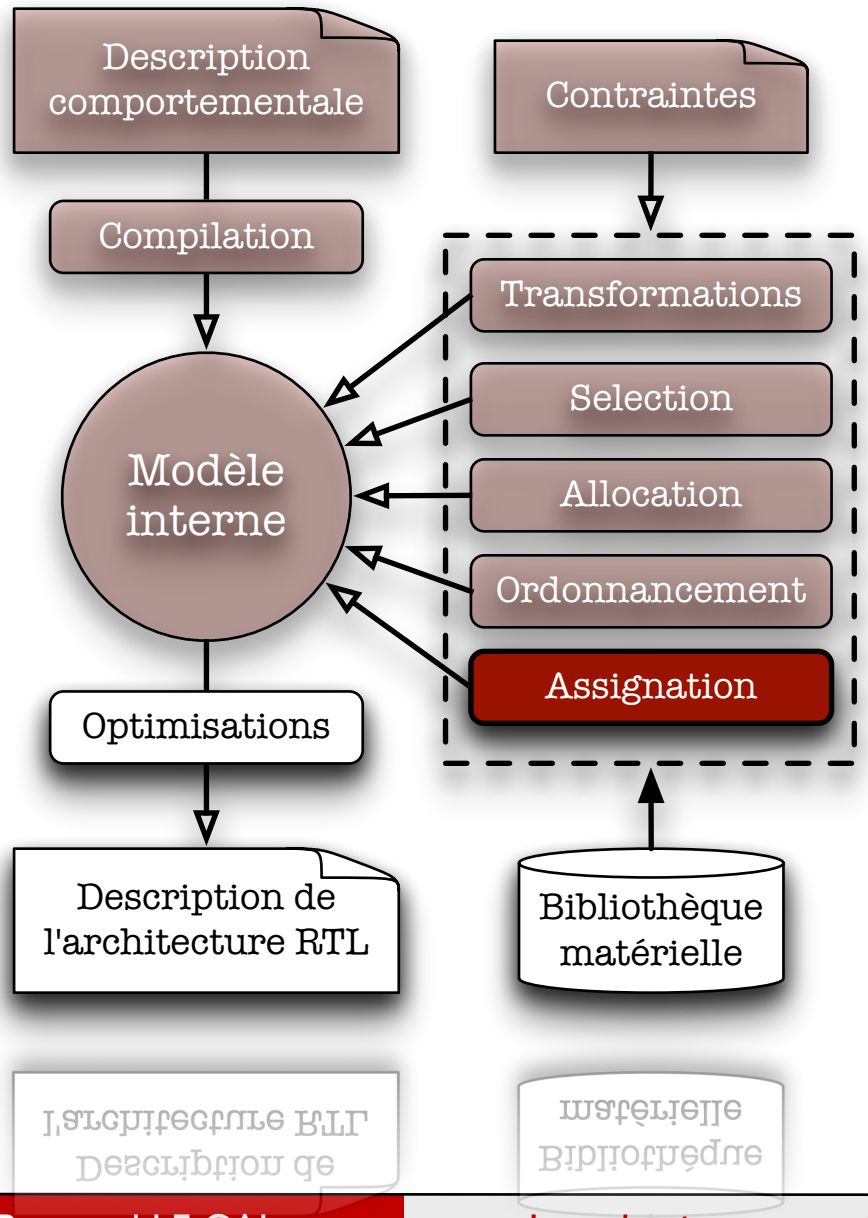


consommation (80 cycles)



consommation (120 cycles)

Assignation sur les ressources matérielles



Les dates d'exécution des différentes opérations à réaliser ont été déterminées vis-a-vis du nombre d'opérateurs disponibles

Sur quelles ressources matérielles, les opérations sont elles réalisées ?

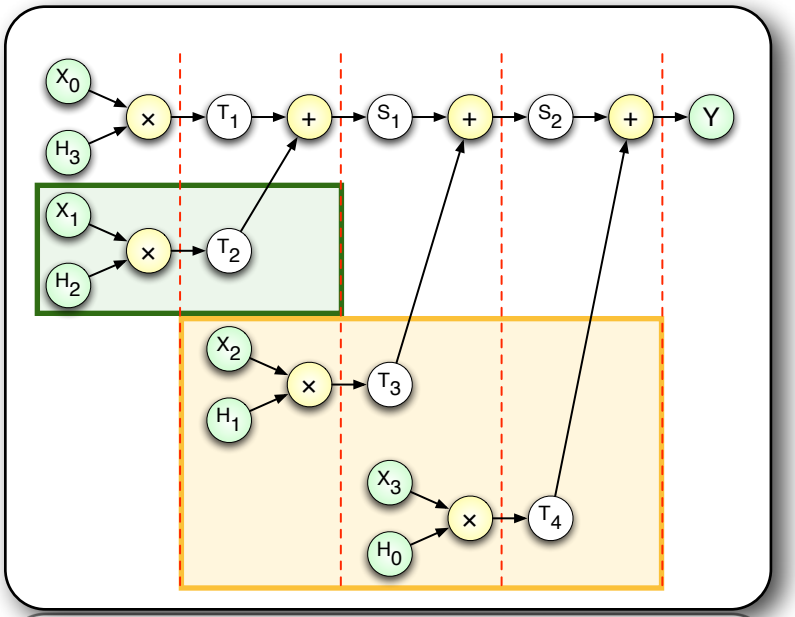
=> réduire la glue logique nécessaire au routage des informations,

=> limiter la complexité du contrôleur,

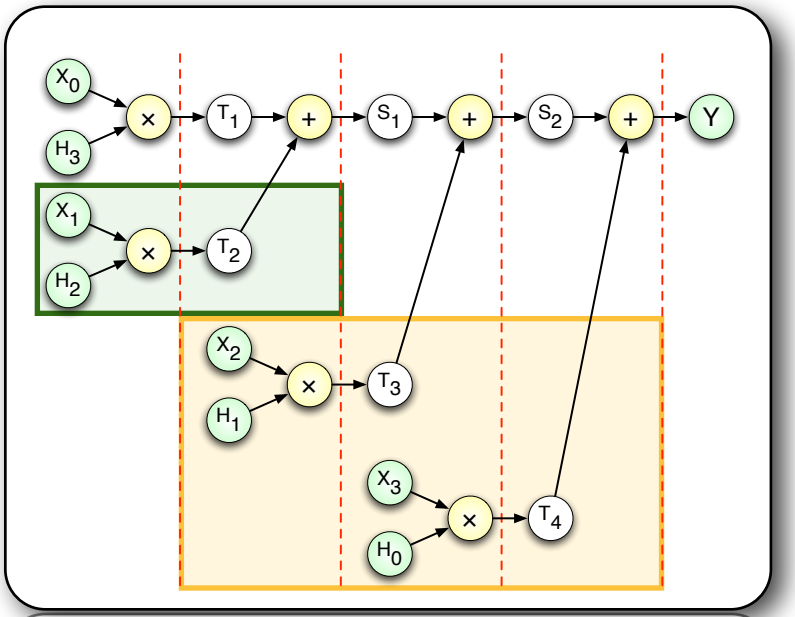
Assignation des opérations ordonnancées

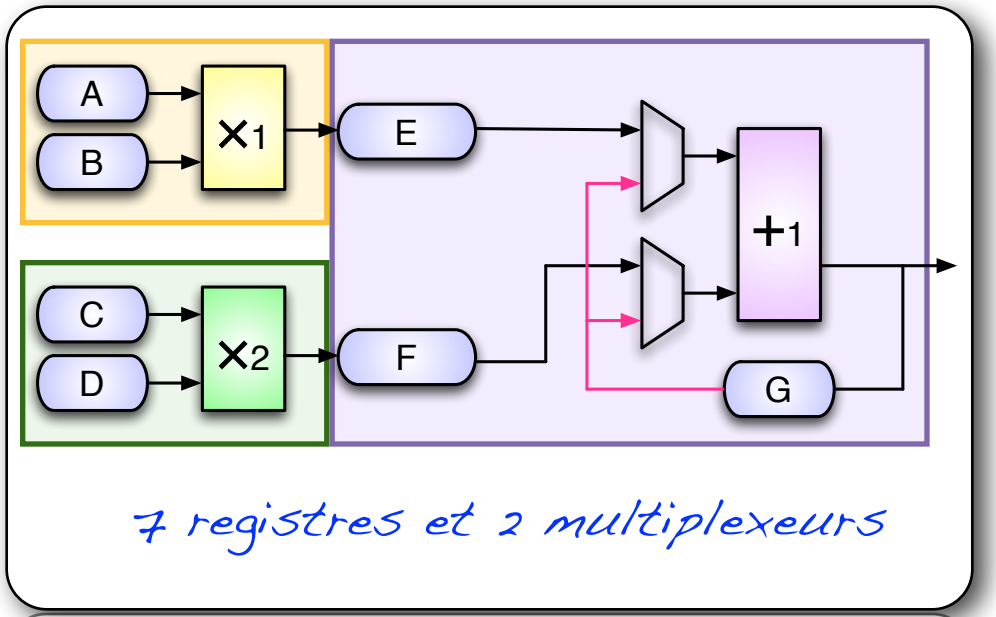
- Nous venons de réaliser l'ordonnancement en fonction des ressources matérielles mises à disposition => chaque opération possède une date d'exécution,
- Il faut maintenant décider sur quelle ressource matérielle chaque opération du modèle va être exécutée. Cela est réalisé durant la phase d'assignation,
- Cette étape est aussi importante que les autres car :
  - ➡ En fonction de l'assignation, les connexions entre les composants vont être différentes => modification des performances du système,
    - Augmentation de la surface,
    - Augmentation de la consommation,
    - Augmentation de la durée des chemins critiques,

Exemple de projection des opérations

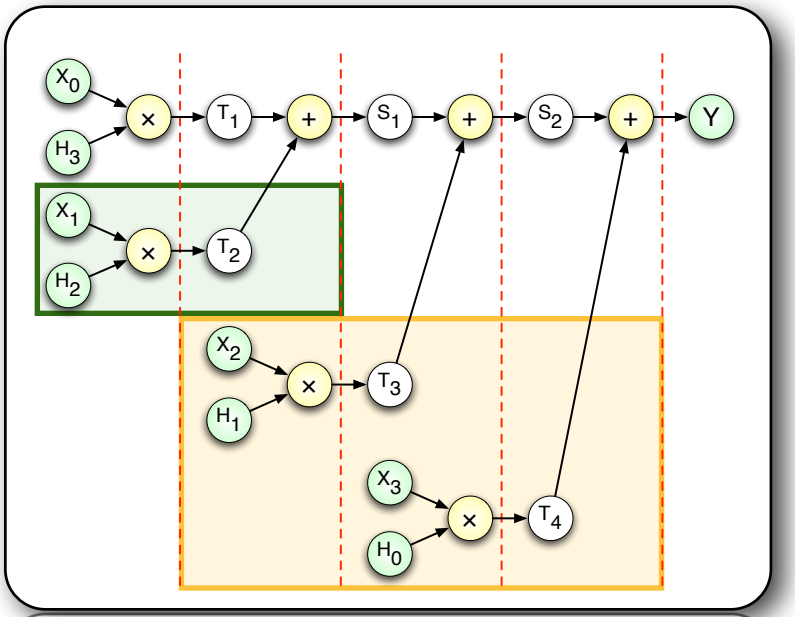


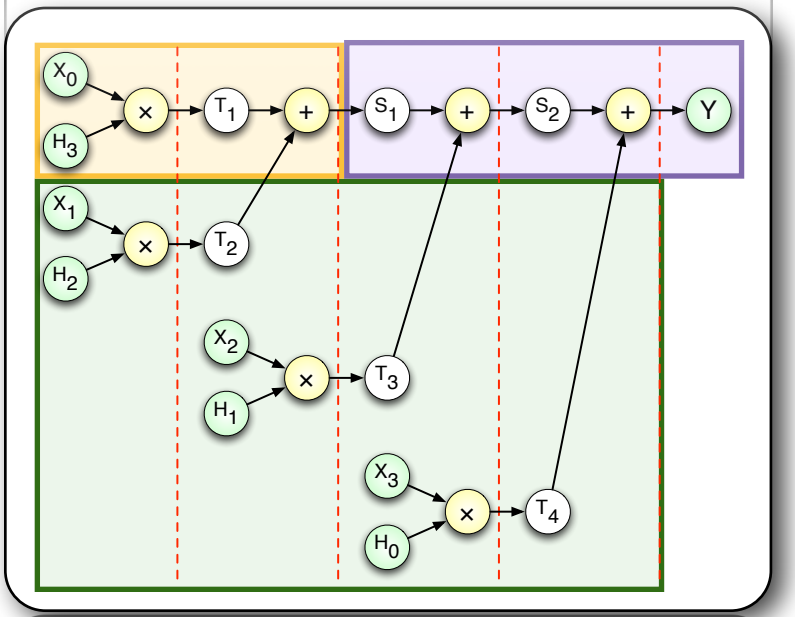
Exemple de projection des opérations

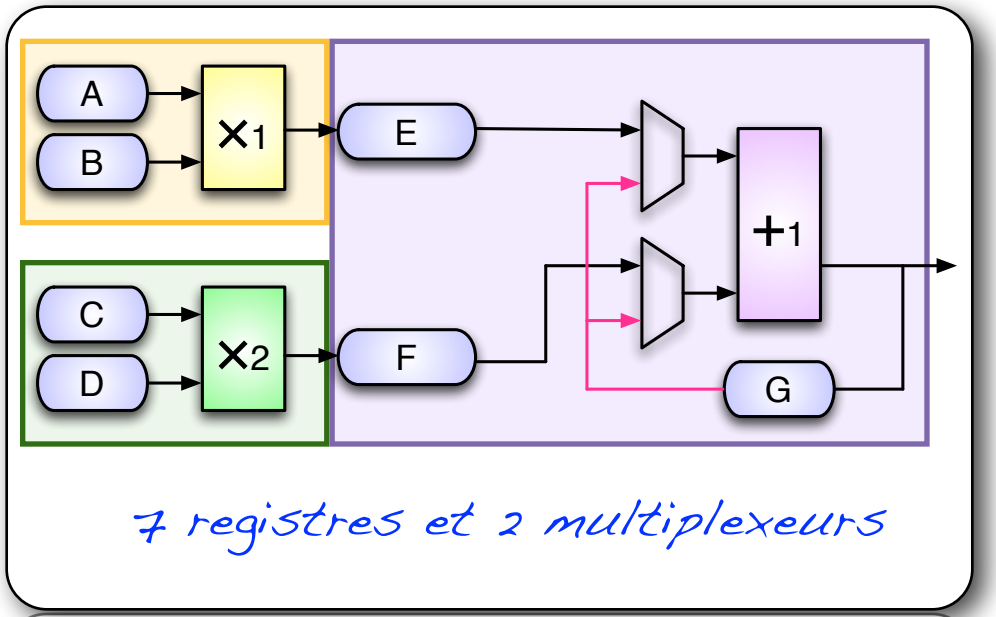




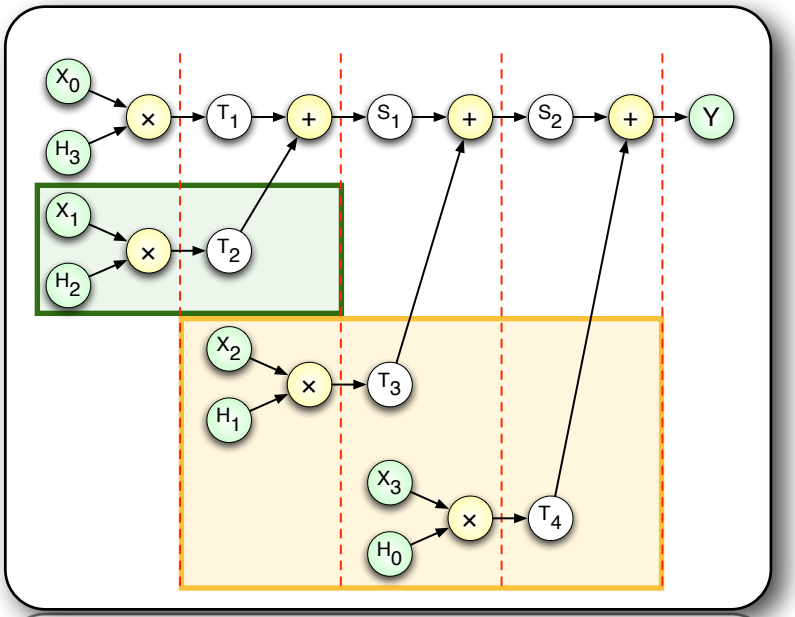
Exemple de projection des opérations

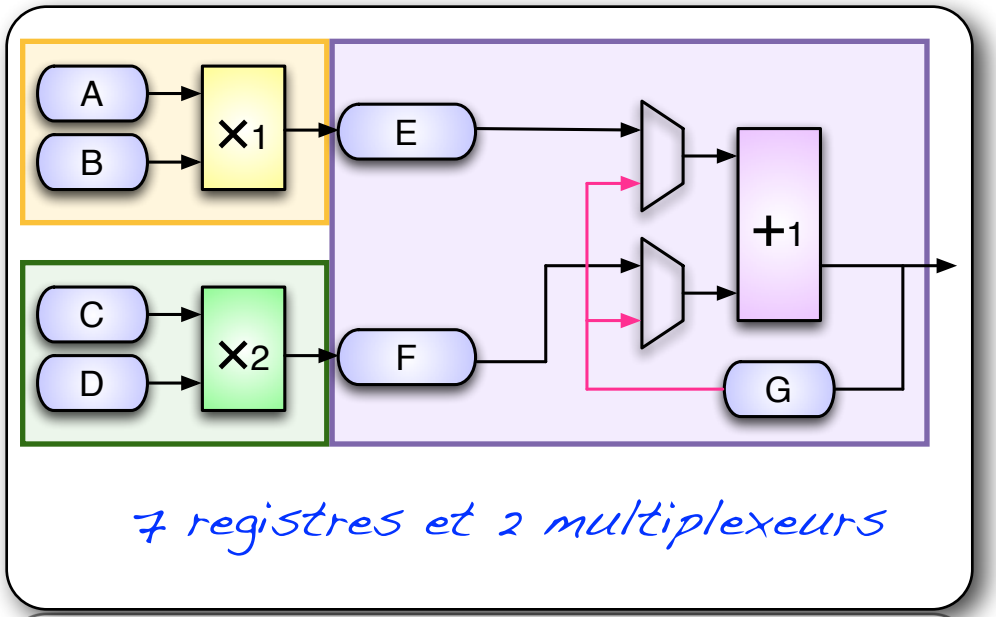


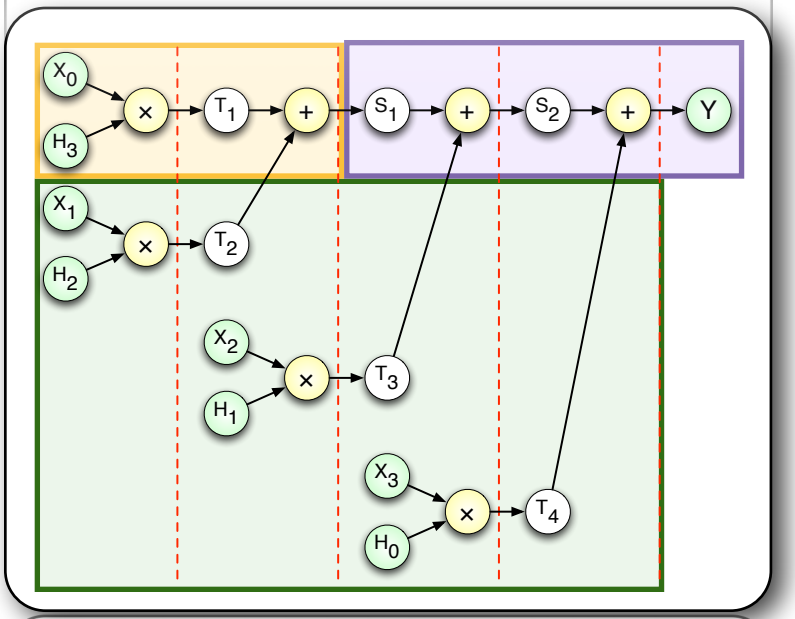


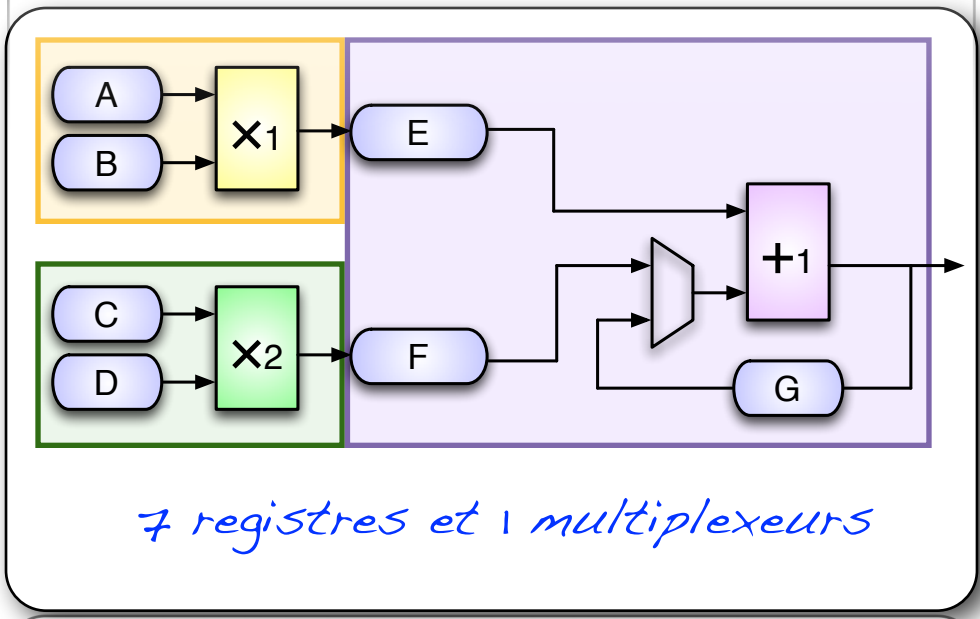


Exemple de projection des opérations







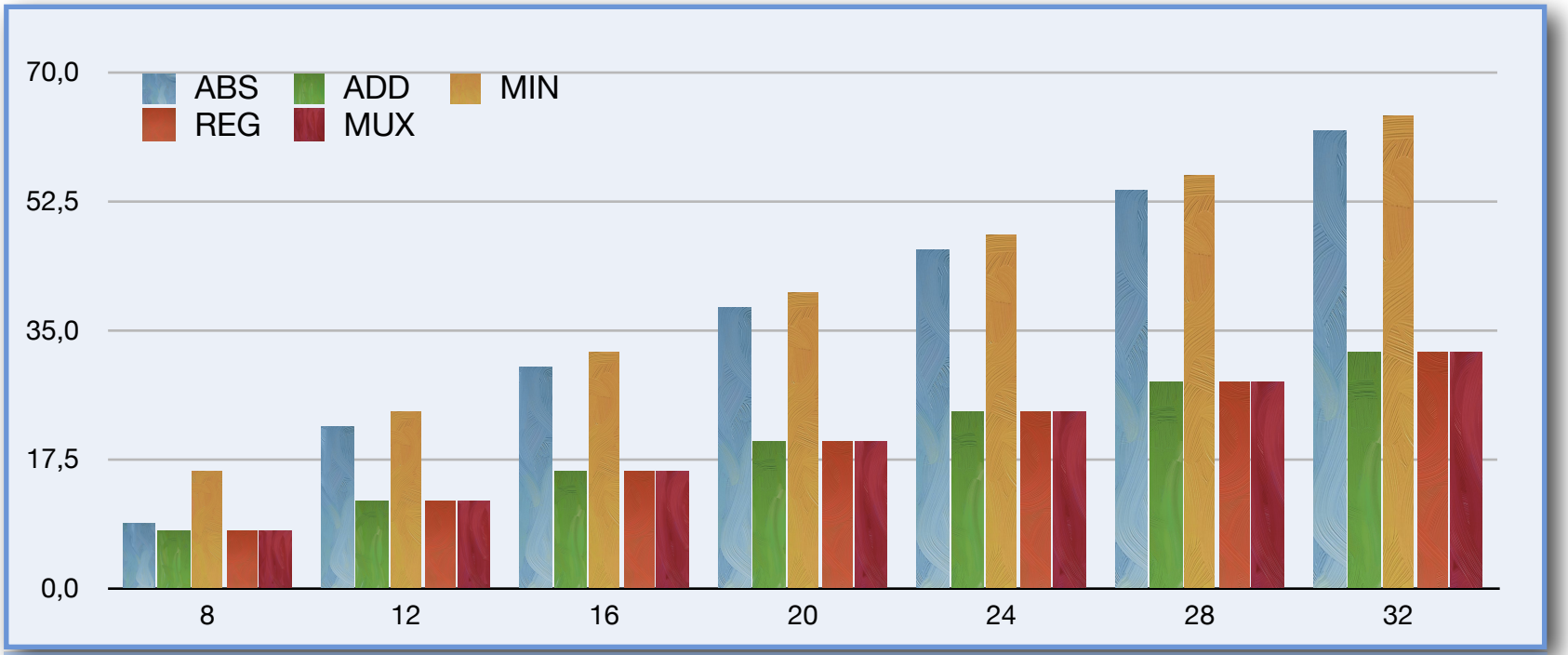


Coût réel de la glue logique sur FPGA

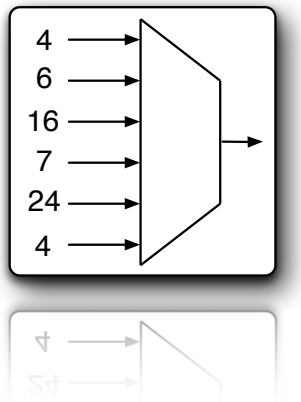
Surface (LUT) - Xilinx Virtex-5 (VLX30T-5)

Bitwidth	4	8	12	16	20	24	28	32
ABS	3	9	22	30	38	46	54	62
ADD	4	8	12	16	20	24	28	32
MULT	15	46	104	186	292	422	576	754
SUB	4	8	12	16	20	24	28	32
REG	4	8	12	16	20	24	28	32
MUX	4	8	12	16	20	24	28	32

Le coût de la glue logique est loin d'être négligeable...

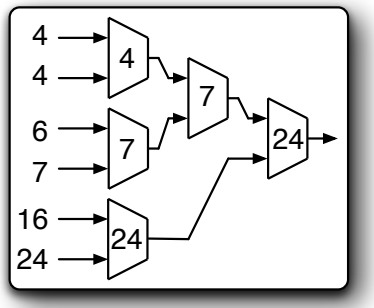


Problématique du coût de la logique de routage

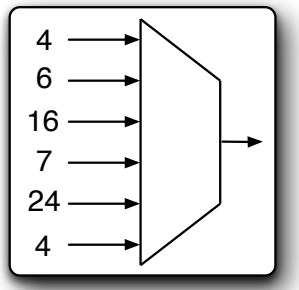


Imaginons un besoin de routage 6 vers 1

Problématique du coût de la logique de routage

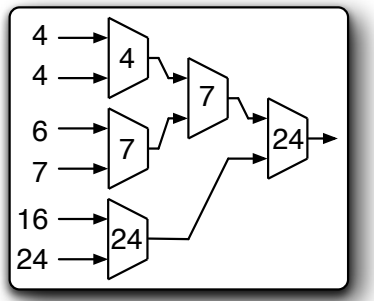


Implantation rapide mais couteuse

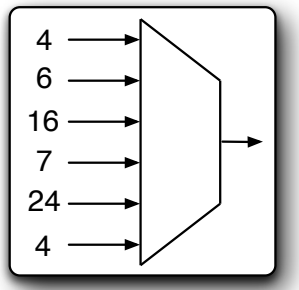


Imaginons un besoin de routage 6 vers 1

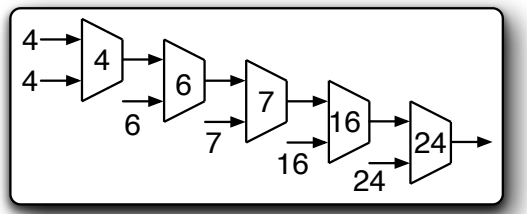
Problématique du coût de la logique de routage



Implantation rapide mais couteuse

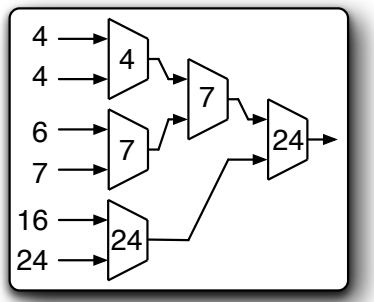


Imaginons un besoin de routage 6 vers 1

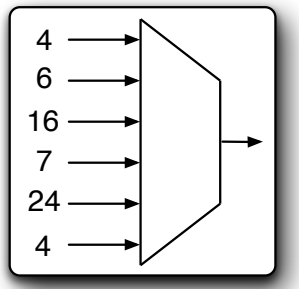


Implantation lente mais moins coûteuse

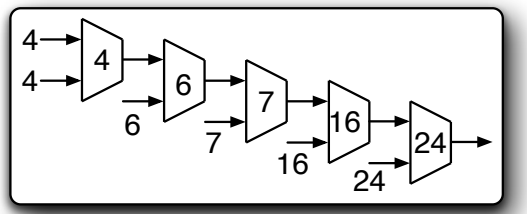
Problématique du coût de la logique de routage



Implantation rapide mais couteuse

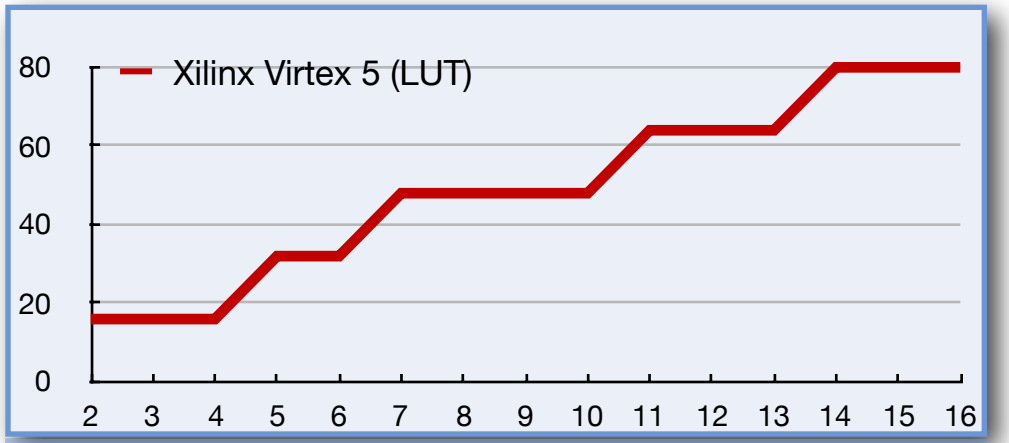


Imaginons un besoin de routage 6 vers 1

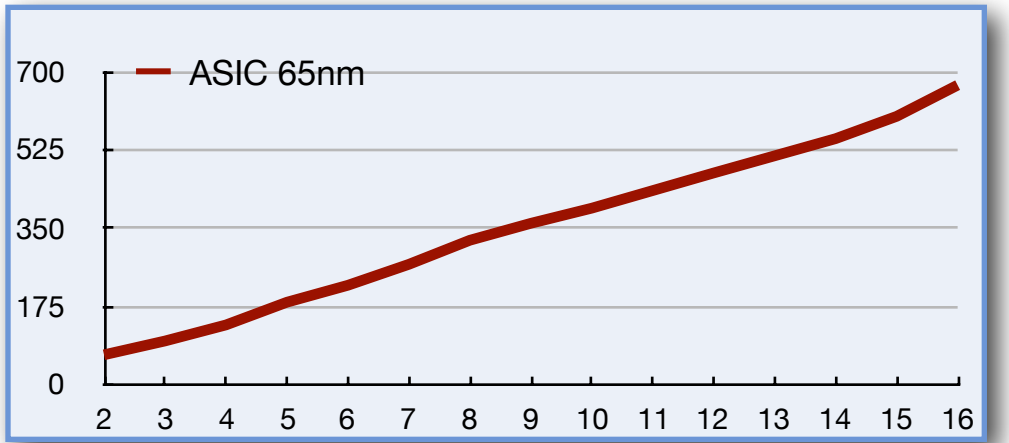


Implantation lente mais moins coûteuse

Multiplexer area (LUT) on Xilinx Virtex 5 - Fx(input ports)

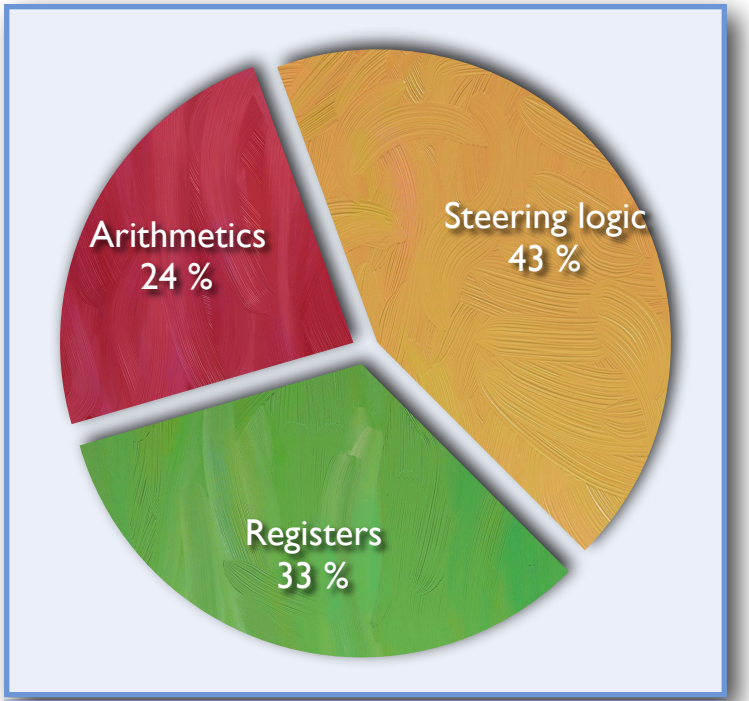


Multiplexer area (Nand gates equivalence) on 65nm low-power ASIC technology depending on the number of input ports.



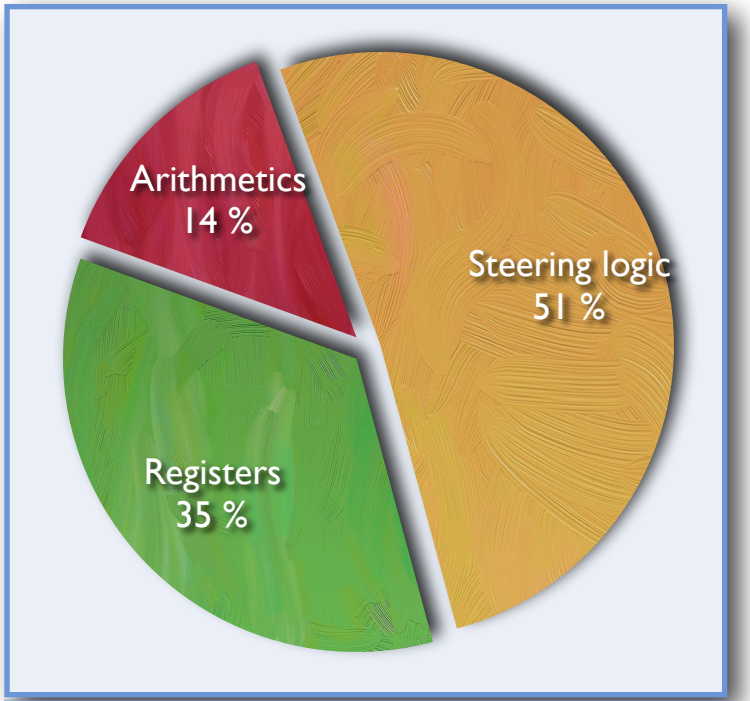
Modèle formel de la 2d-DCT 8x8 (Assignation)

Répartition des couts (80 cycles)



Area cost estimation = 49152 LE
(before logical synthesis)

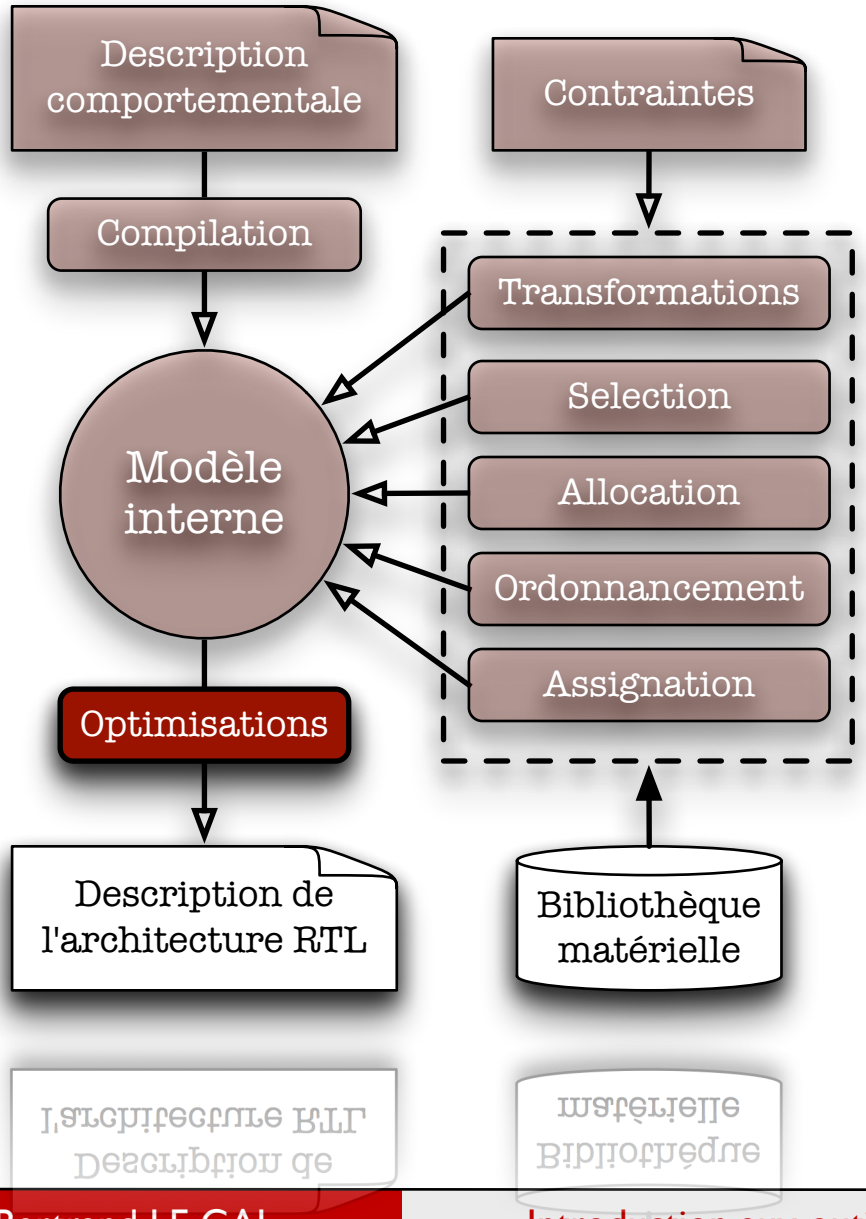
Répartition des couts (120 cycles)



Area cost estimation = 36853 LE
(before logical synthesis)

Expérimentations : IOs en ligne, technologie Altera Cyclone-III

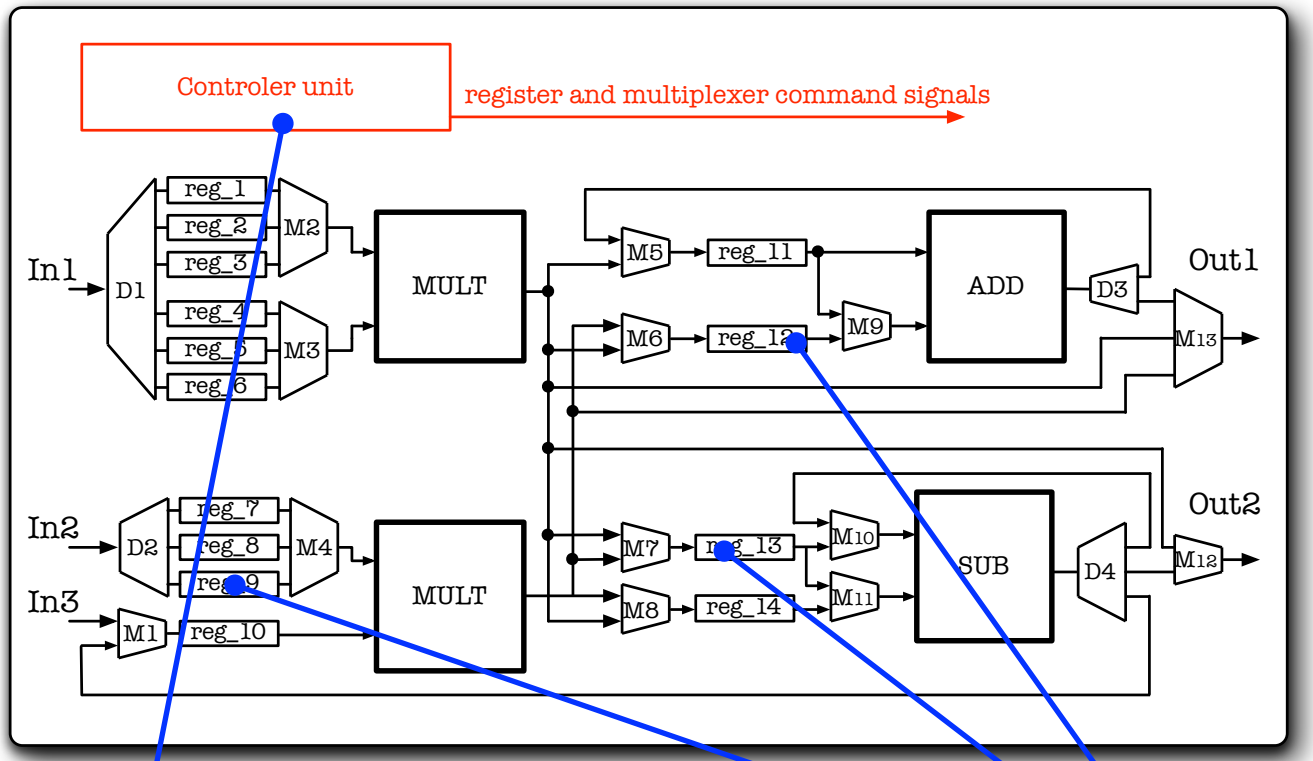
Les méthodes d'optimisation (post-synthèse)



La détermination des éléments composant l'architecture matérielle est déterminé

=> Peut'on encore améliorer ses caractéristiques (surface, chemin critique, consommation) ?

Optimisations possibles dans le chemin de données



=> Minimisation de la surface,

=> Réduction des chemins critiques,

=> Minimisation de la consommation d'énergie

Minimisation de la surface,

Réduction du chemin critique

Optimisations classiques du contrôleur du circuit

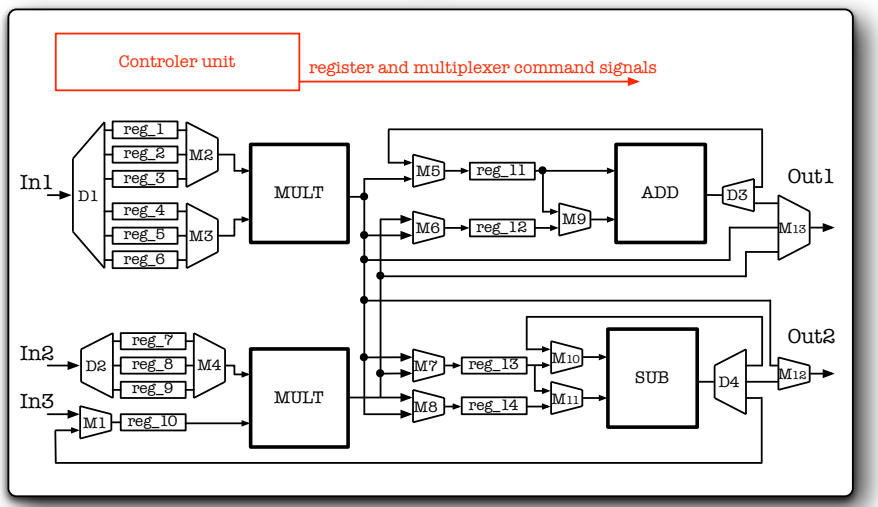
Codage des états dans le contrôleur

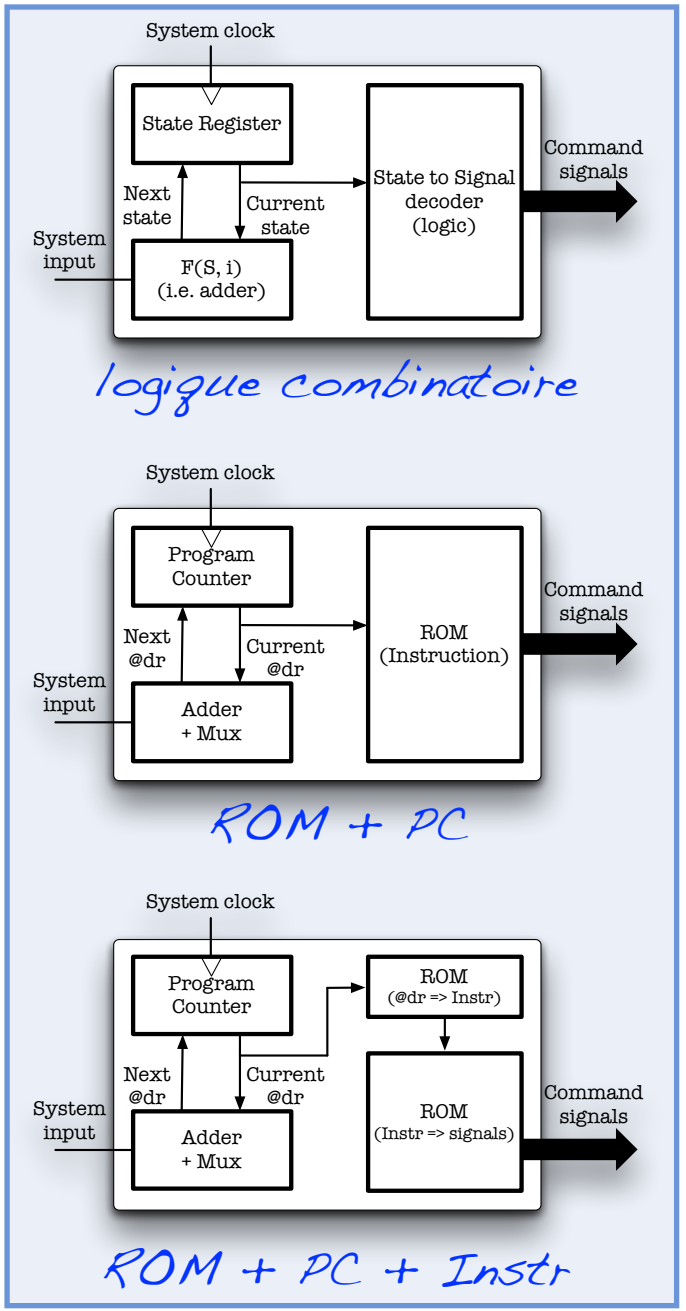
Etat	Codage
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	....
15	1111

Etat	Codage
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	....
15	1000

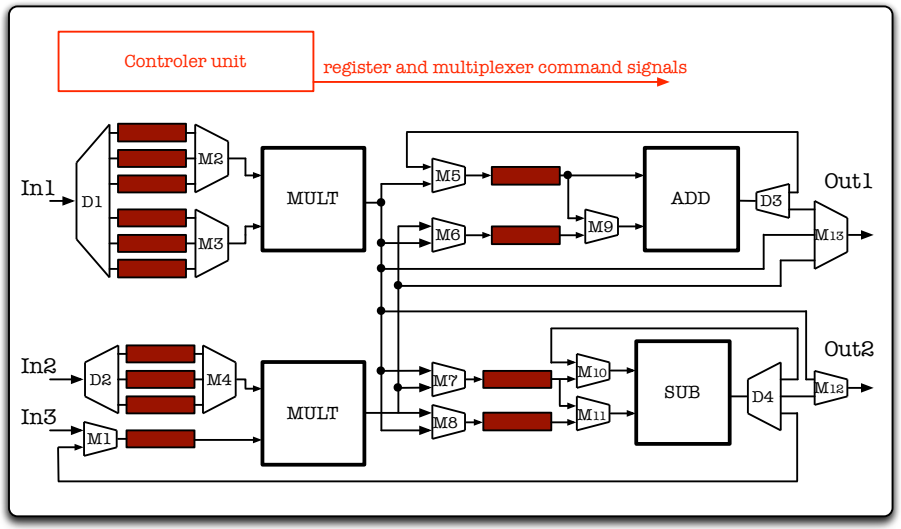
Etat	Codage
0	00 00 00 00
1	00 00 00 01
2	00 00 00 10
3	00 00 01 00
4	00 00 10 00
5	00 01 00 00
6	....
15	10 00 00 00

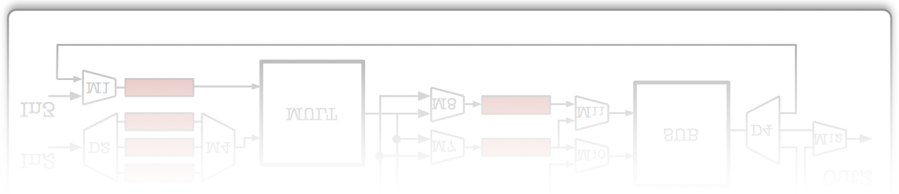
Etat	Codage
0	00 00 00 01
1	00 00 00 10
2	00 00 01 00
3	00 00 10 00
4	00 01 00 00
5	00 10 00 00
6	....
14	10 00 00 00

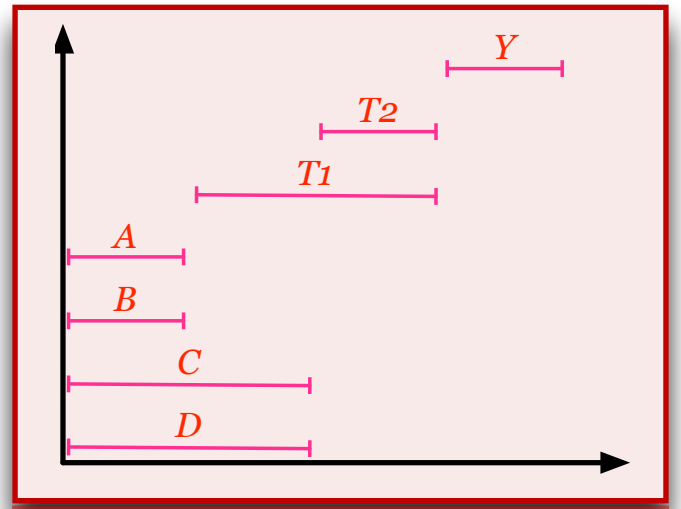


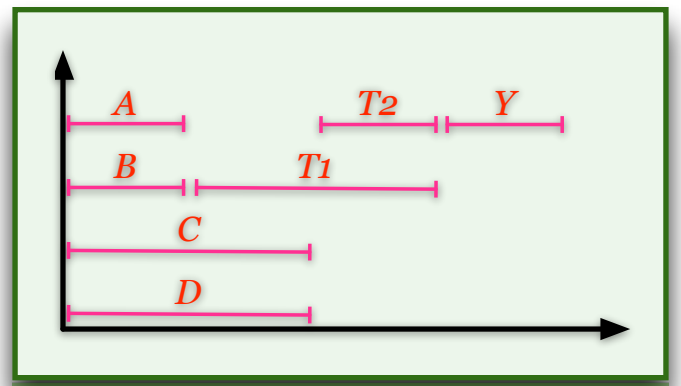


Réduction des ressources de mémorisation



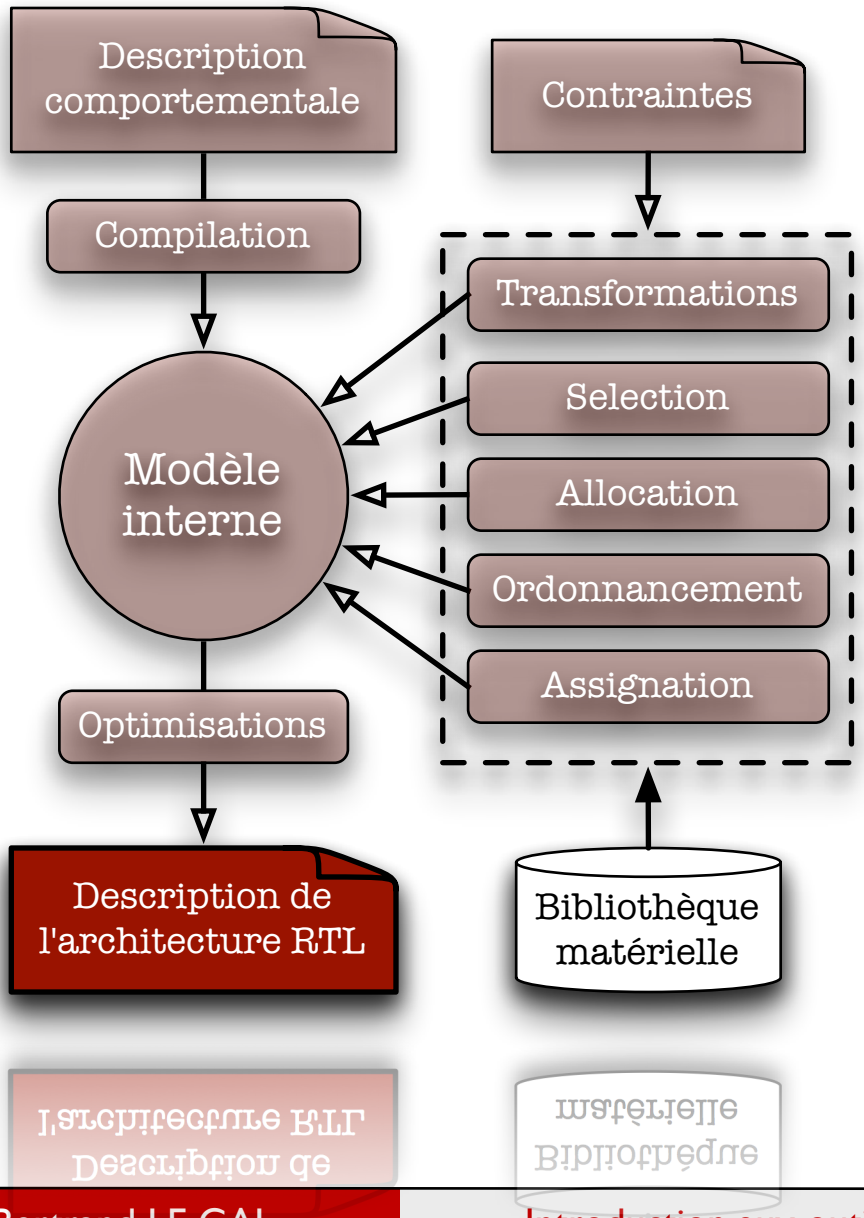






Apres l'optimisation spatio-temporelle des ressources de mémorisation, le système nécessite uniquement 4 registres au lieu de 7 au départ

Génération de l'architecture matérielle



L'architecture matérielle est définie très précisément (ensemble de ses éléments du contrôleur au chemin de données...)

Maintenant il faut restituer au concepteur un fichier contenant la description du circuit...

La génération architecturale de la solution

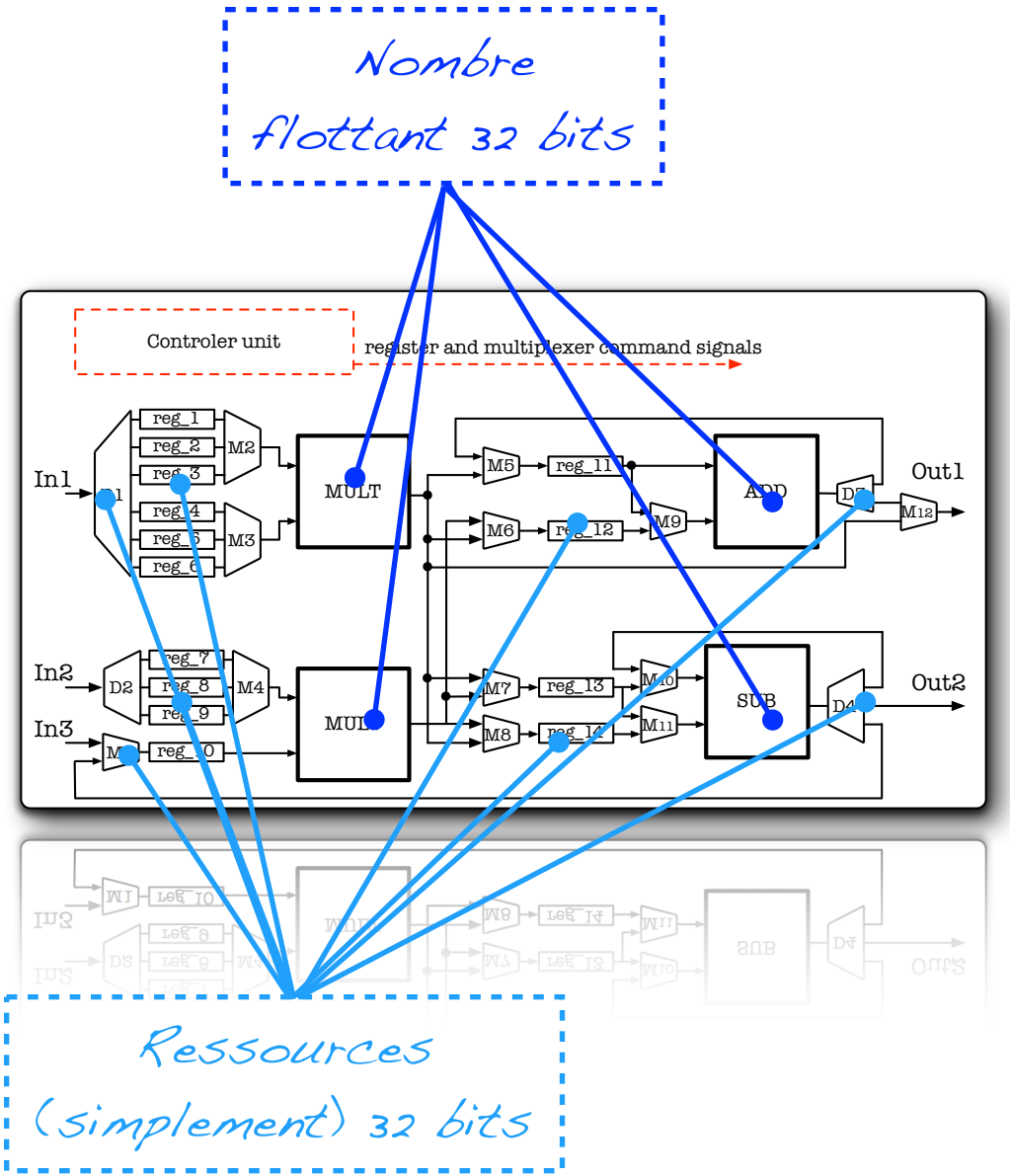
- Quels langages doit'on générer afin de pouvoir intégrer notre architecture dans le système en cours de développement ?
  - Le langage VHDL,
  - Le langage Verilog,
  - Le langage SystemC,
- Souhaite t'on générer plusieurs modèles du circuit à des niveaux d'abstraction différents afin de :
  - Accélérer la simulation du système (interfaces fixes),
  - Permettre une simulation cycle près (SystemC RTL),
  - Pour l'implémentation VHDL (niveau RTL).
    - Implantation lisible & compréhensible (par l'humain)
    - Implantation optimisée en vue de la synthèse logique.

Partie 3:
Les contraintes connexes

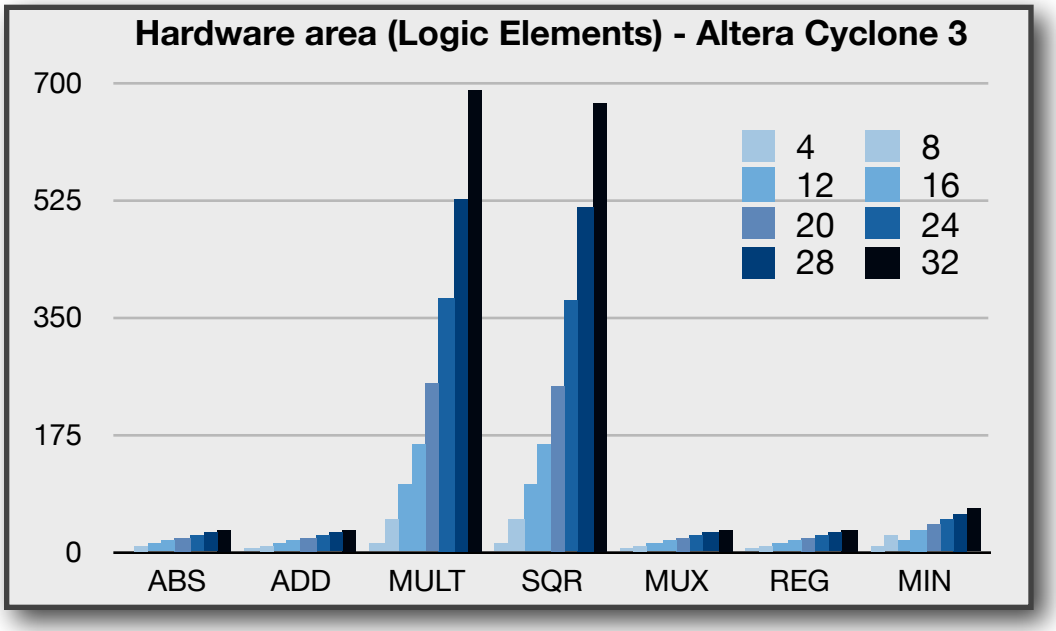
Partie 3.1
Gestion de la taille variable des données

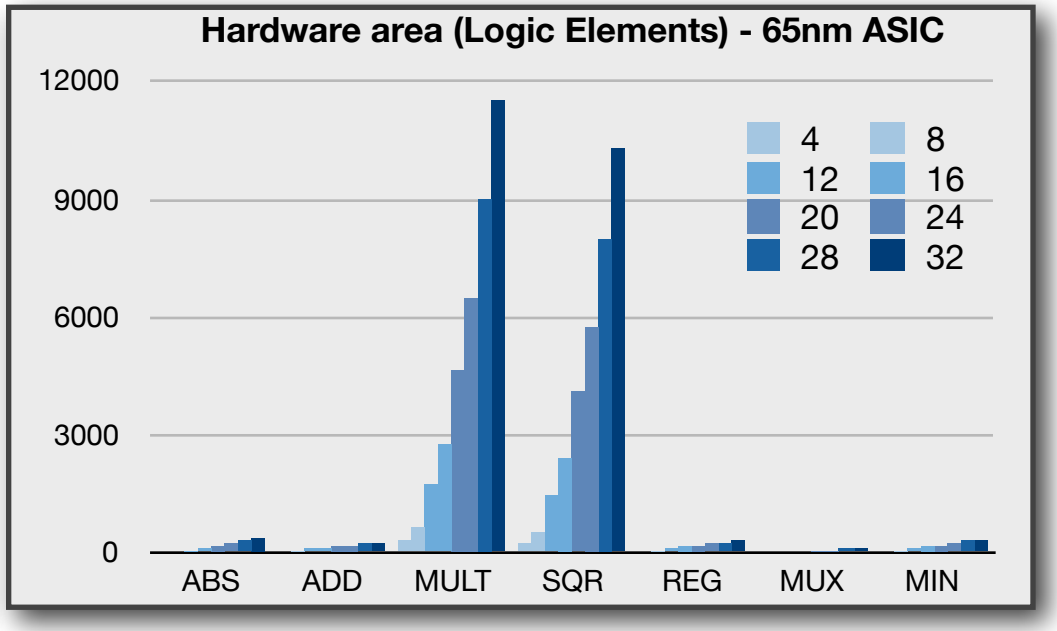
Dynamique des données - Introduction à la problématique

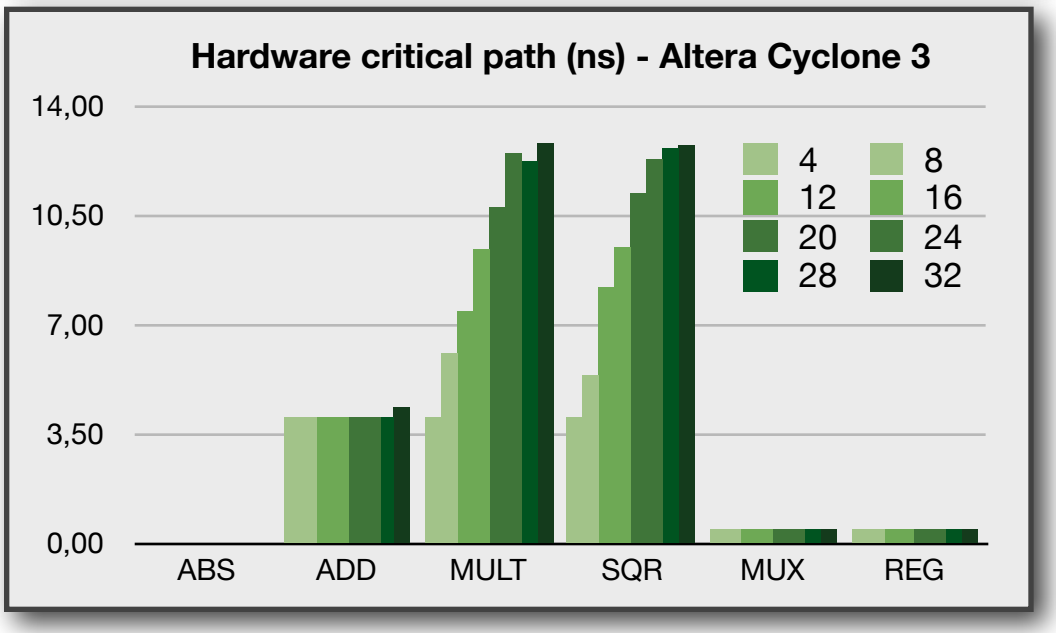
- Tous les algorithmes ne manipulent pas des données de taille constante (float, int, etc),
- La dynamique est données est liée à l'application traitée :
  - Format des données d'entrée,
  - Des calculs (et recadrages) réalisé,
- La dynamique des données est une contrainte supplémentaire,
  - La qualité de l'architecture est meilleure,
- Techniques plus complexes...

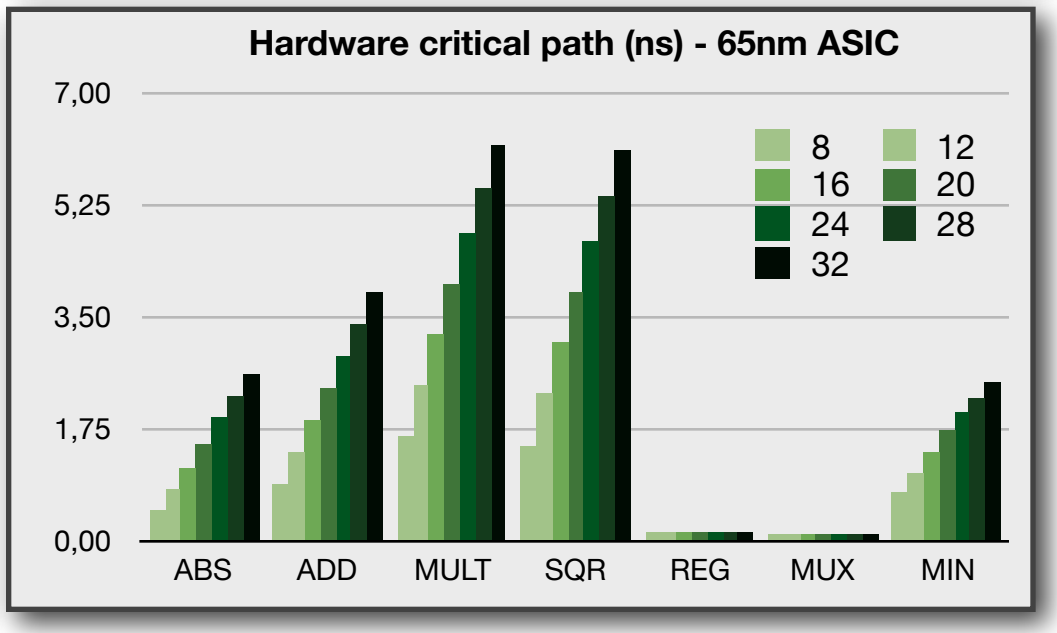


Evolution de la surface et de la latence : fx(word-length)



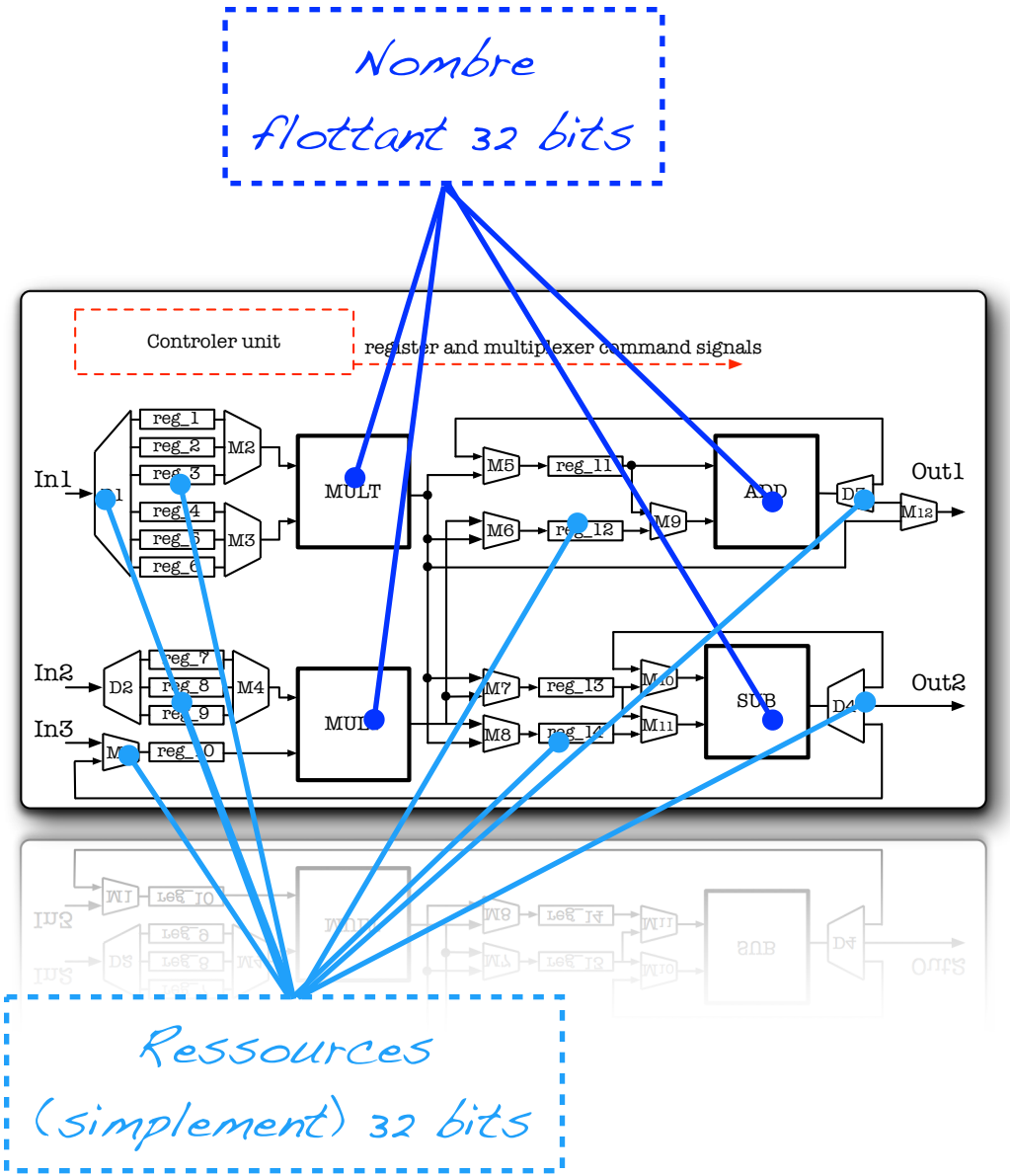






Gestion de la dynamique variable des opérations

- La gestion de la dynamique des calculs peut être gérer de différentes manières:
  - Taille constante du chemin de données (architecture type des uProcesseurs)



Gestion de la dynamique variable des opérations

- La gestion de la dynamique des calculs peut être gérer de différentes manières:
  - Taille constante du chemin de données (architecture type des uProcesseurs)
  - Expression manuelle des formats de données (concepteur),

Nombre flottant 32 bits

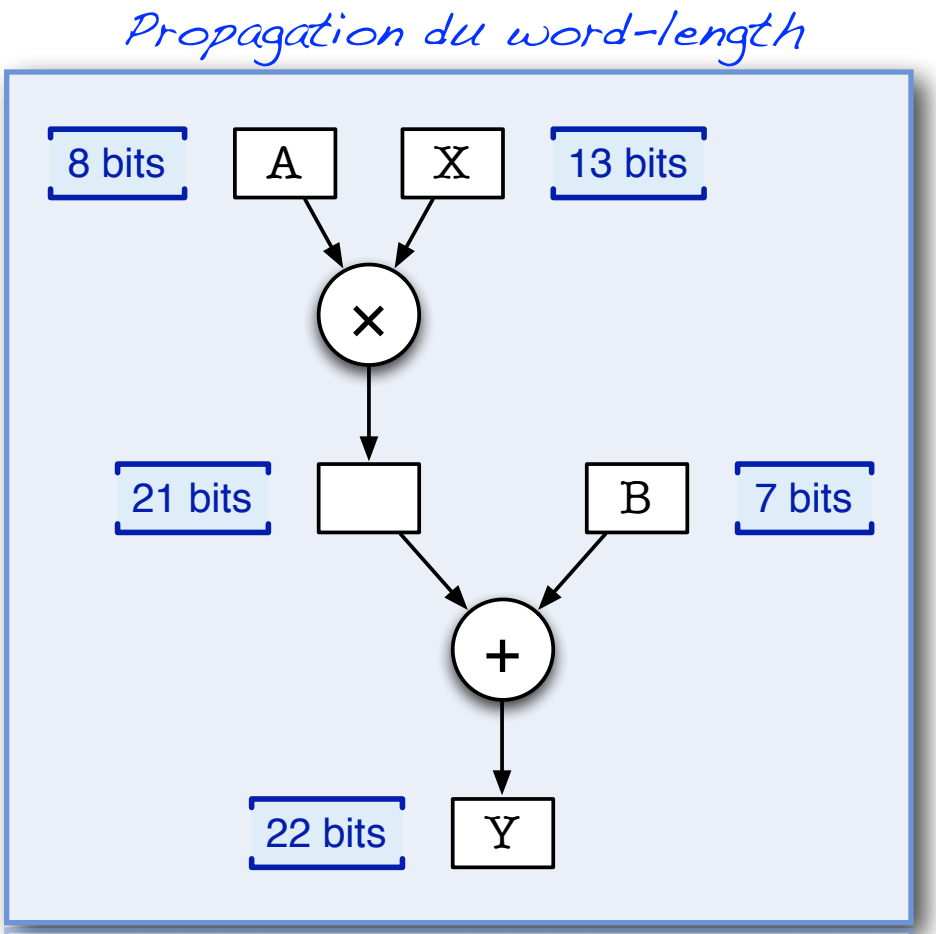
```
sc_uint<8> A = 129;
sc_int<13> X = in.read();
sc_uint<7> B = 41;
sc_int<22> T = (A * X) + B;
Y.write( tmp );
```

Nombre flottant 32 bits

```
sc_fixed< 8, 6> A = 2.746328;
sc_fixed<12, 6> X = in.read();
sc_fixed<10, 4> B = 33.764382;
sc_fixed<16, 8> T = (A * X) + B;
Y.write( X );
```

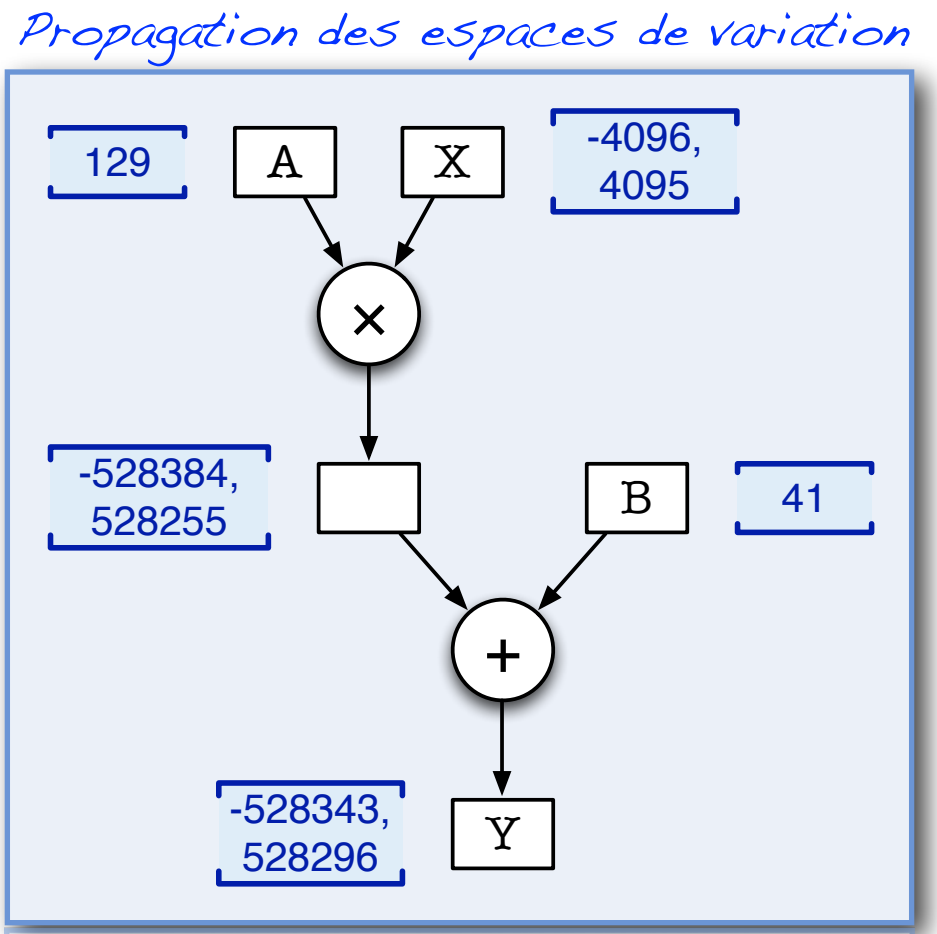
Gestion de la dynamique variable des opérations

- La gestion de la dynamique des calculs peut être gérer de différentes manières:
  - Taille constante du chemin de données (architecture type des uProcesseurs)
  - Expression manuelle des formats de données (concepteur),
  - Analyse automatique (bit-width) en fonction des entrées,



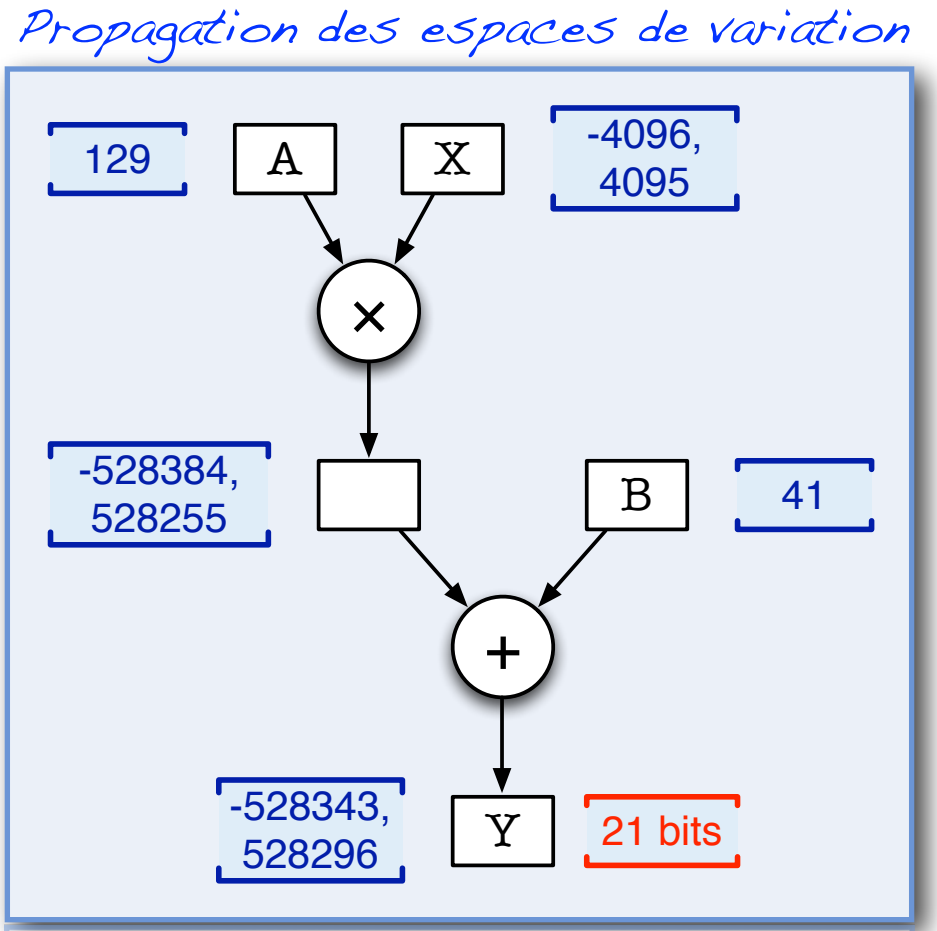
Gestion de la dynamique variable des opérations

- La gestion de la dynamique des calculs peut être gérer de différentes manières:
  - Taille constante du chemin de données (architecture type des uProcesseurs)
  - Expression manuelle des formats de données (concepteur),
  - Analyse automatique (bit-width) en fonction des entrées,
  - Analyse automatique (word-length) en fonction des entrées,

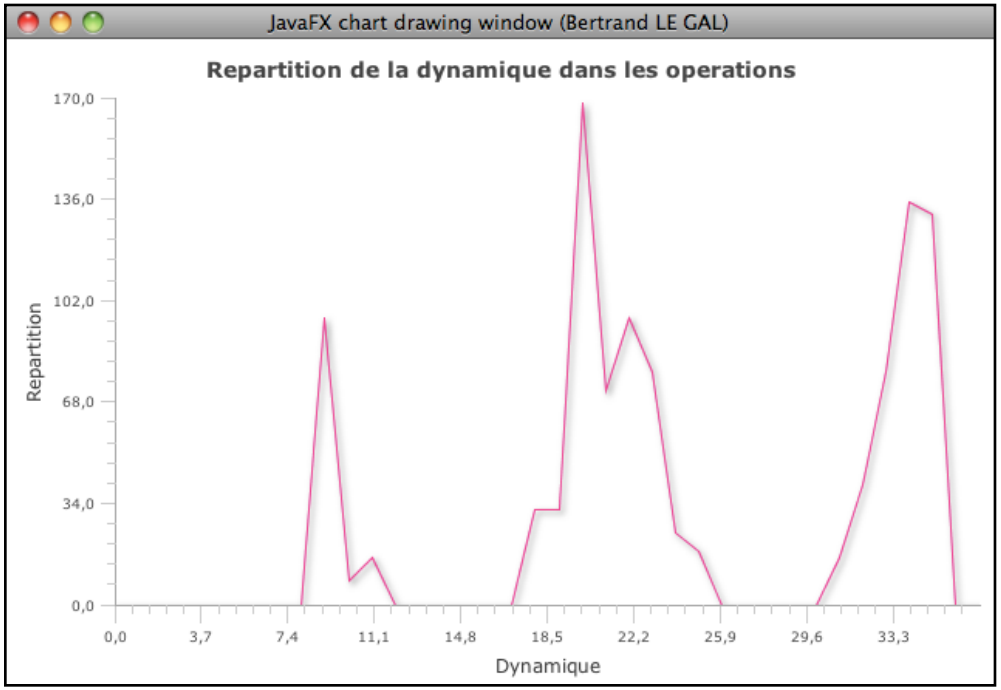


Gestion de la dynamique variable des opérations

- La gestion de la dynamique des calculs peut être gérer de différentes manières:
  - Taille constante du chemin de données (architecture type des uProcesseurs)
  - Expression manuelle des formats de données (concepteur),
  - Analyse automatique (bit-width) en fonction des entrées,
  - Analyse automatique (word-length) en fonction des entrées,
  - Intégration de paramètres statistiques,
- La taille des données peut aussi être extraite de simulations.

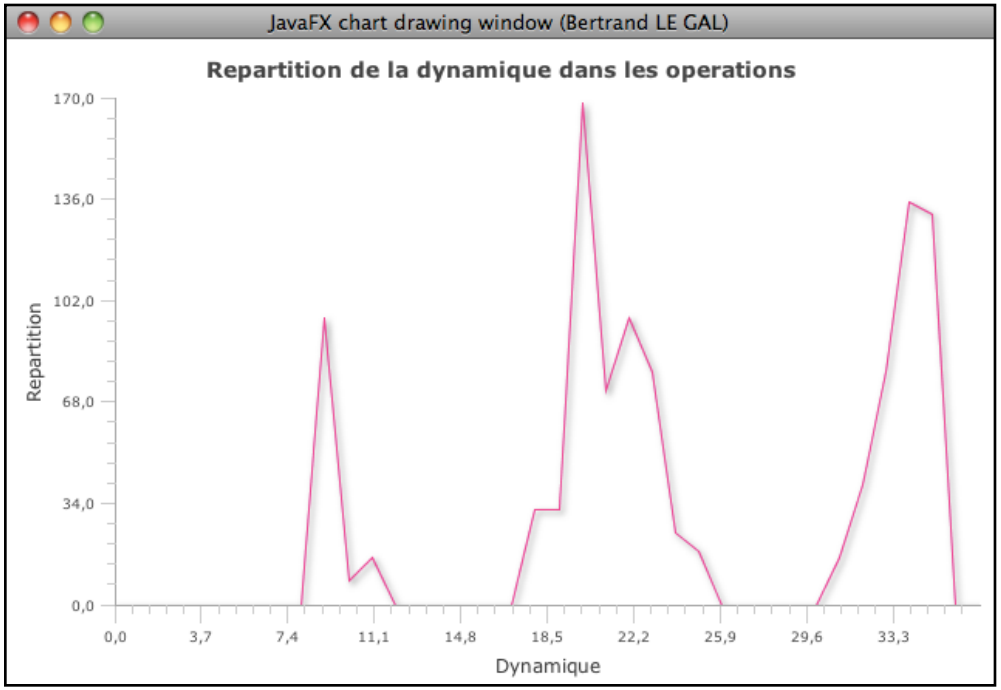


Modèle formel de la 2d-DCT 8x8 (Latence des opérations)

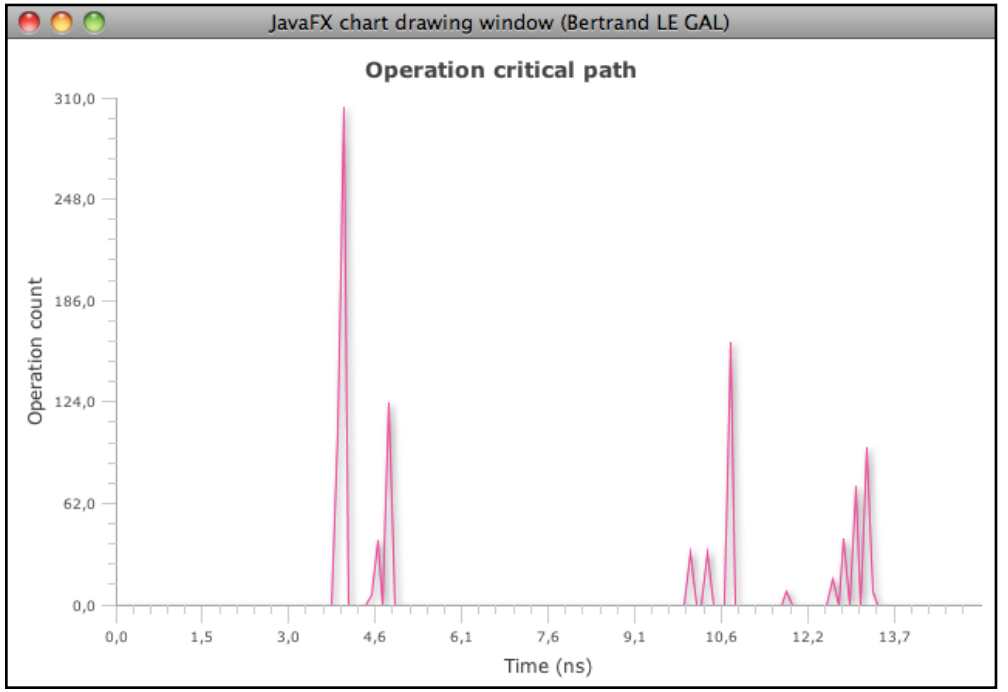


La dynamique des opérations
varie de 8 à 40 bits
(pas de recadrage dans notre matlab)

Modèle formel de la 2d-DCT 8x8 (Latence des opérations)



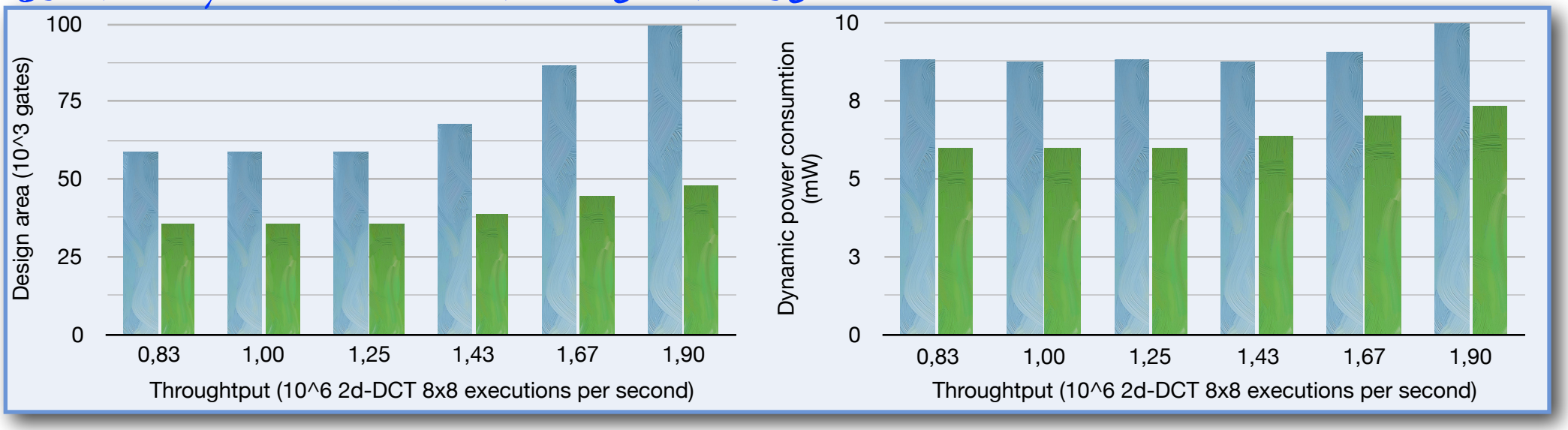
La dynamique des opérations
varie de 8 à 40 bits
(pas de recadrage dans notre matlab)



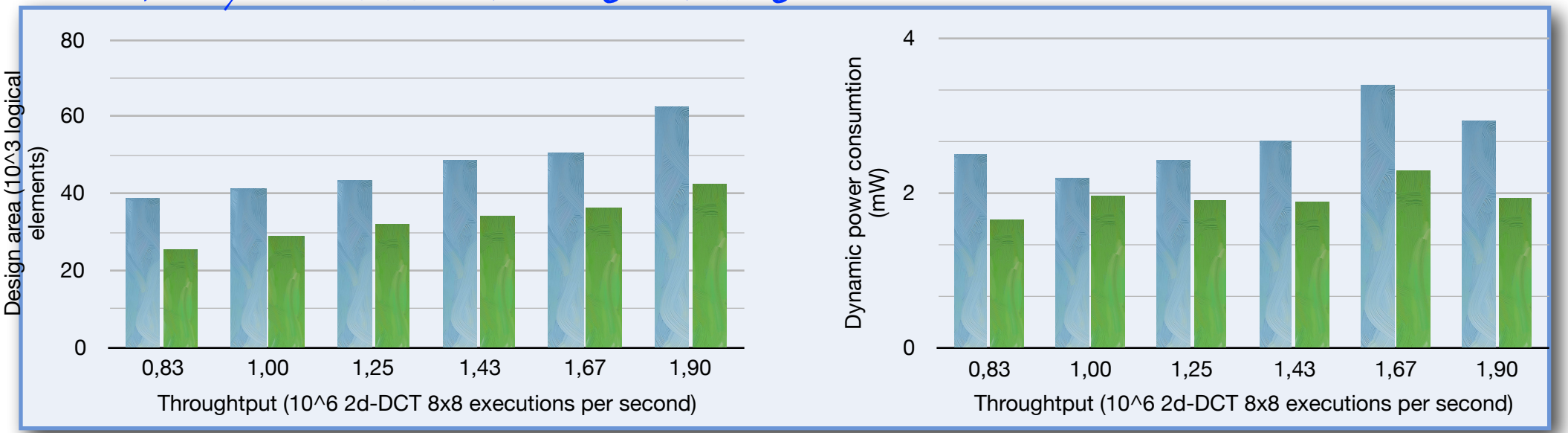
La durée des opérations varie en
fonction de leurs dynamiques
=> délai multi-cycle si nécessaire

Modèle formel de la 2d-DCT 8x8 (Surface)

65nm low power ASIC (min = 30% max=50%)



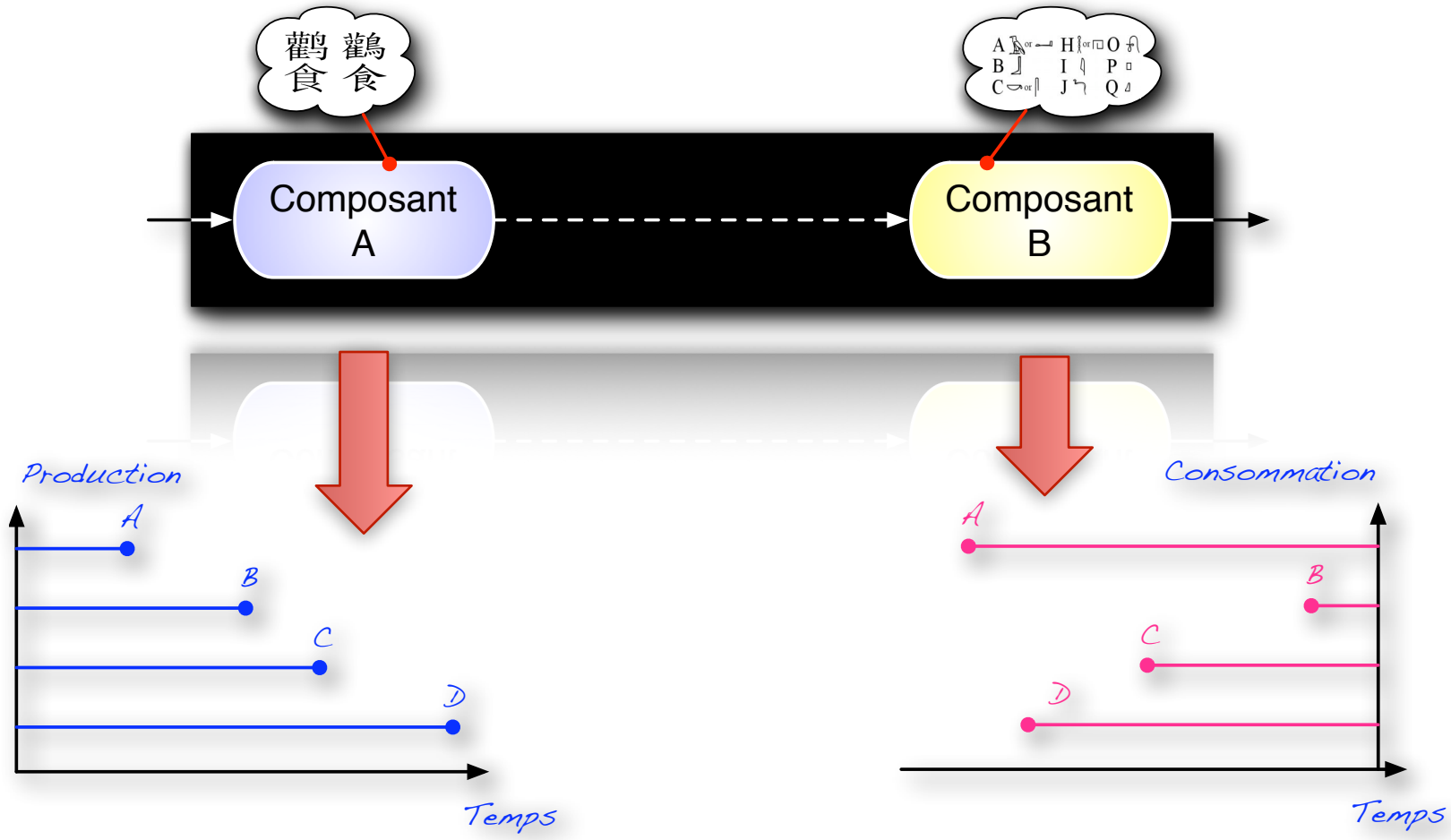
ALTERA Cyclone III (min = 20% max=30%)



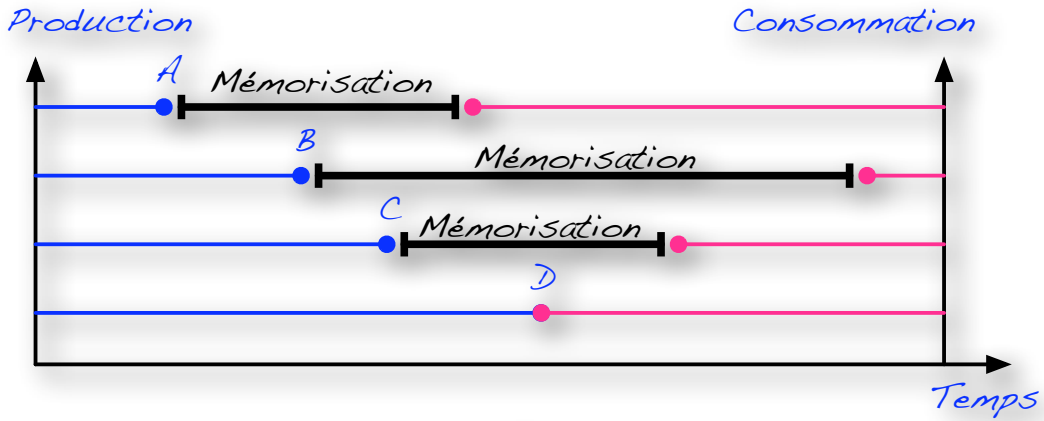
Partie 3.2
Contraintes provenant du système

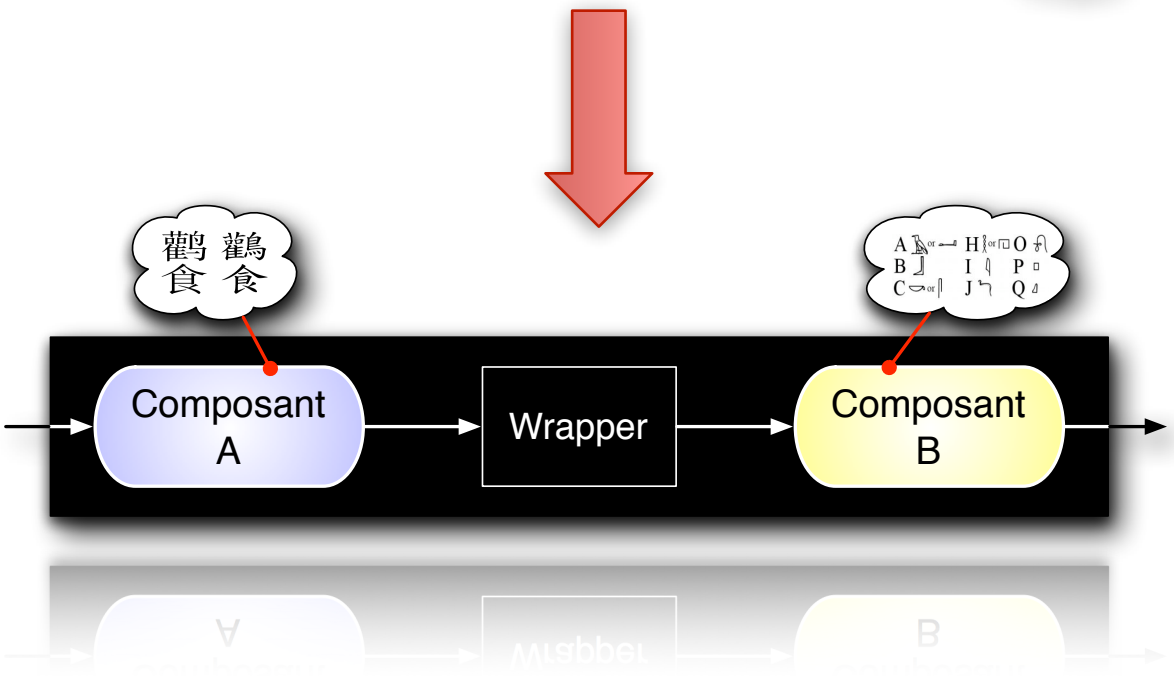
Gestion des IOs - Les contraintes liées au système

Dans un système il est rare que les communications entre composants soient immédiates: problèmes de synchronisation, d'ordre de production des données, débits différentes, horloges différentes...



Gestion des IOs - Le besoin de «wrappers»



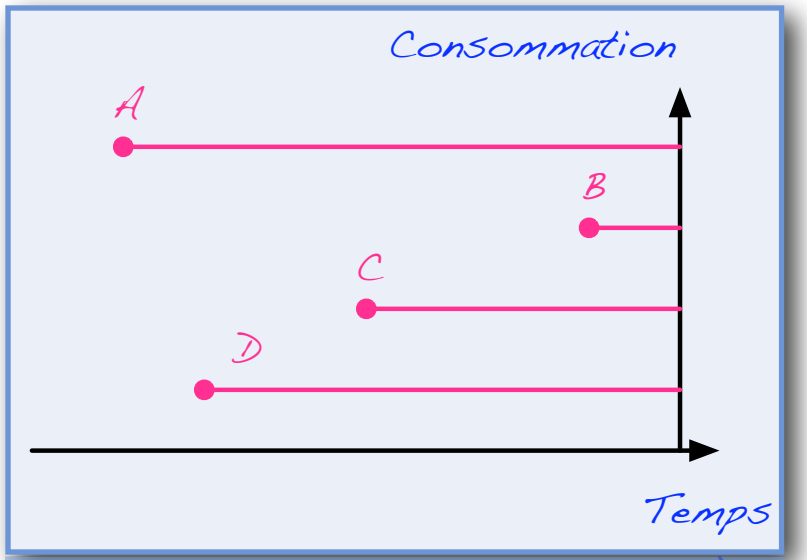


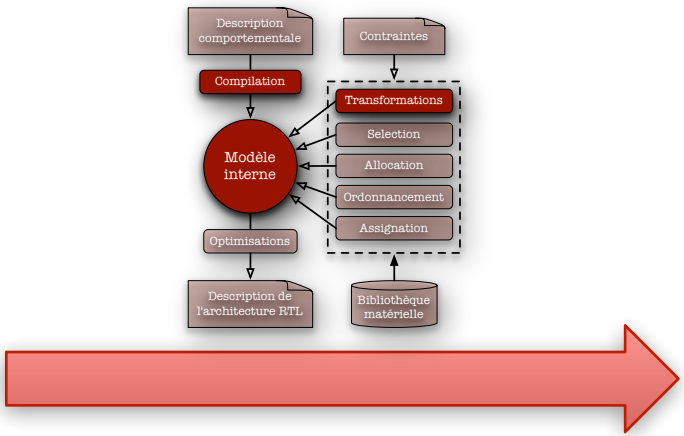
Une solution communément admise vise à concevoir des adaptateurs de protocoles.

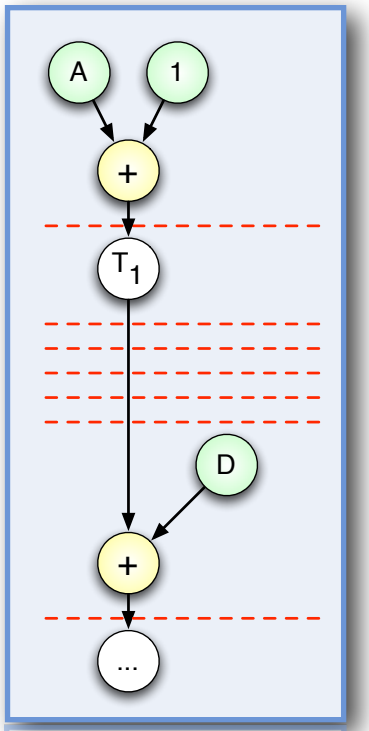
=> Les wrappers font chuter les performances globales du système (débit, latence),

=> Induisent une consommation supplémentaire de surface et d'énergie.

Gestion des IOs - Une contrainte de plus...







Il est nécessaire de considérer les dates (E/S) lors de la synthèse,

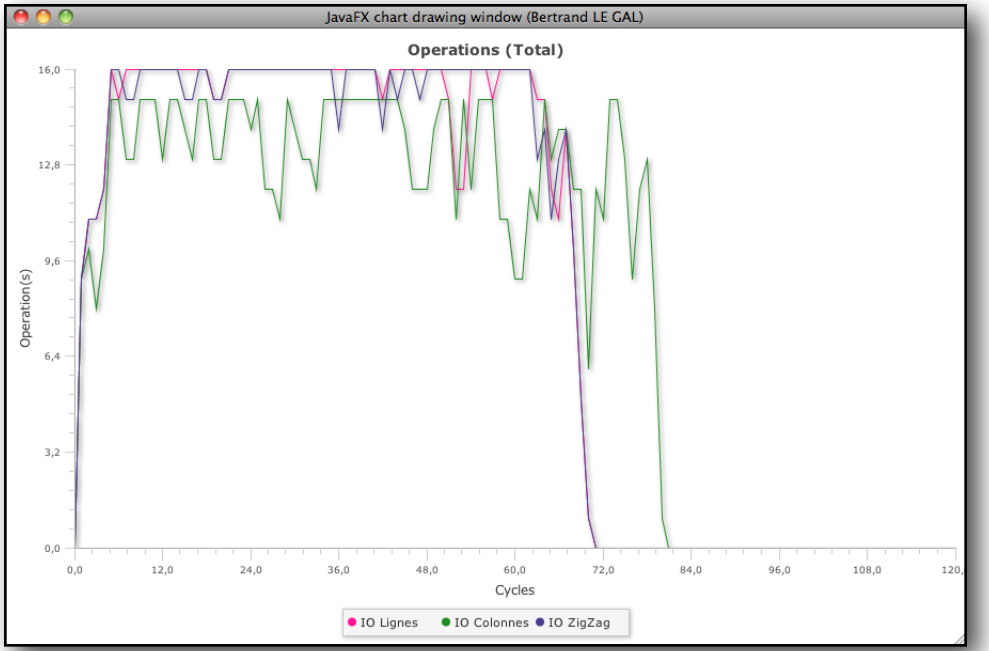
- Optimisation de l'ordre des traitements réalisés
- Gestion des contraintes de production (dates ALAP),

=> Ces informations vont limiter le parallélisme,

=> Des cycles vides doivent être insérés...

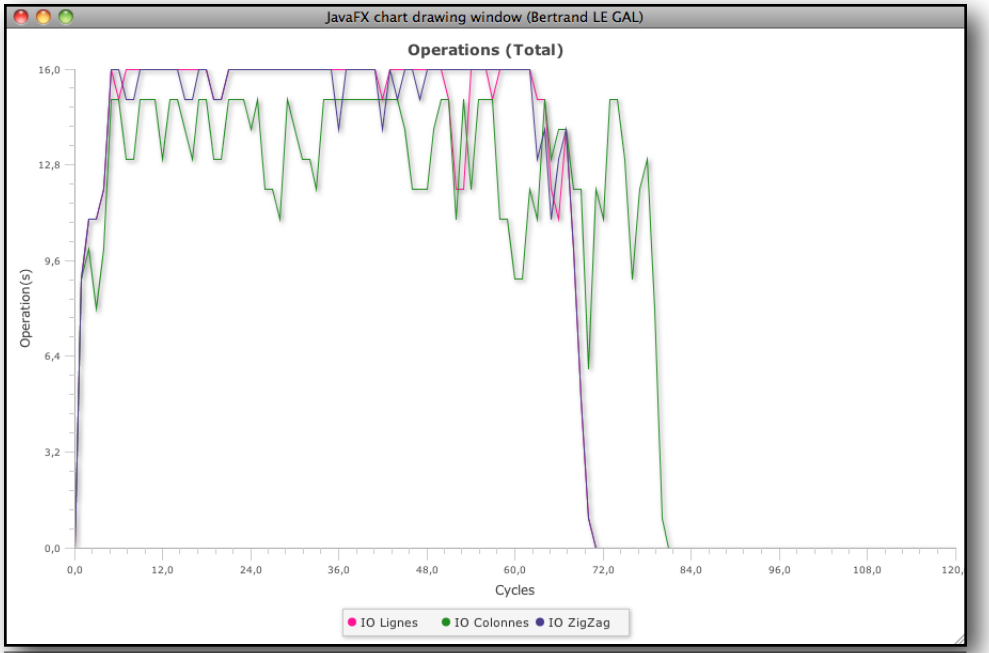
=> L'ordre des calculs est adapté aux dates de mise a disposition des données

Modèle formel de la 2d-DCT 8x8 (IO contraintes)

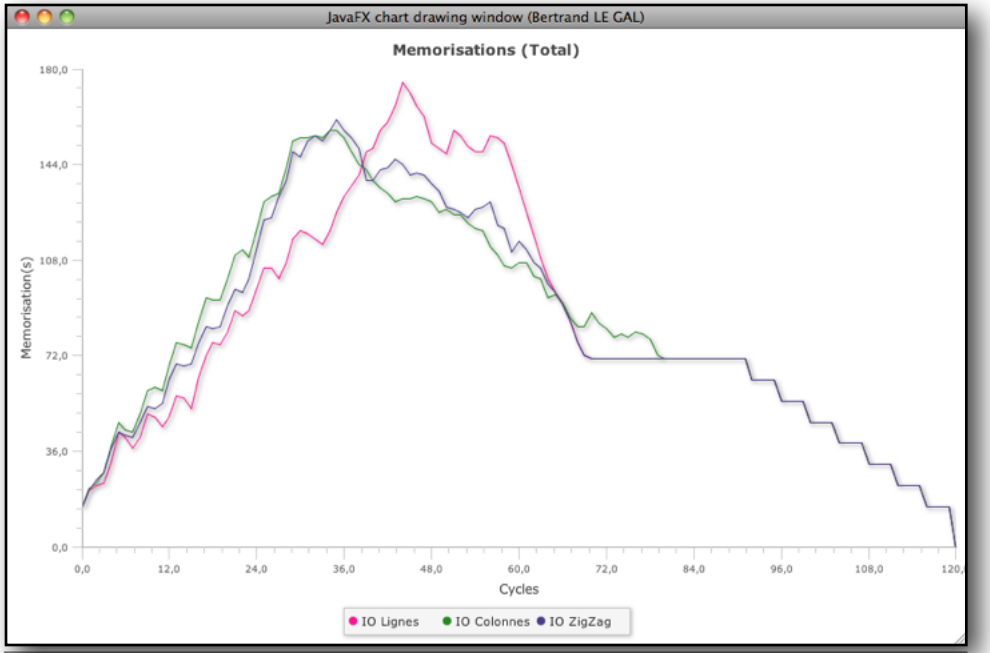


Modification des résultats d'ordonnancement des opérations

Modèle formel de la 2d-DCT 8x8 (IO contraintes)

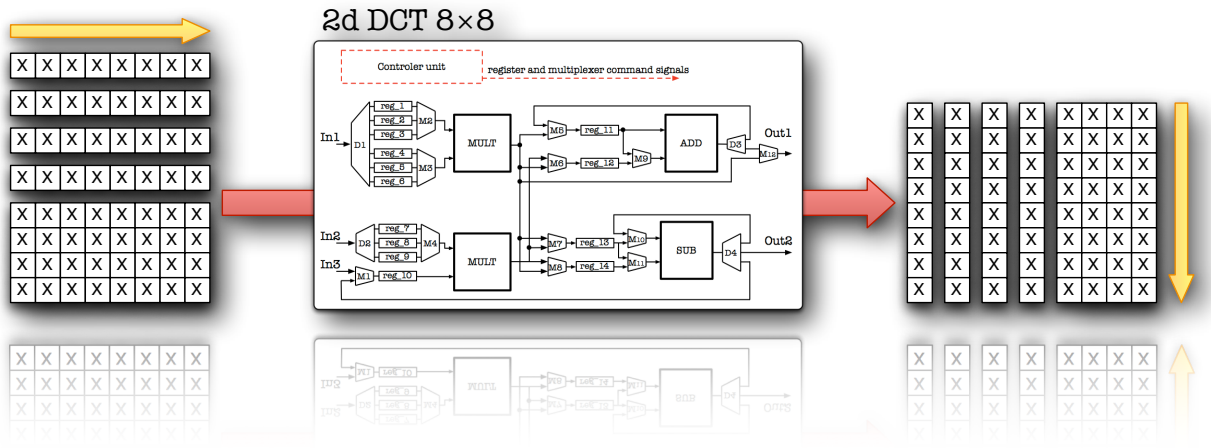


Modification des résultats d'ordonnancement des opérations



Modification des besoins de mémorisation durant l'exécution de l'application.

Modèle formel de la 2d-DCT 8x8 (IO contraintes)

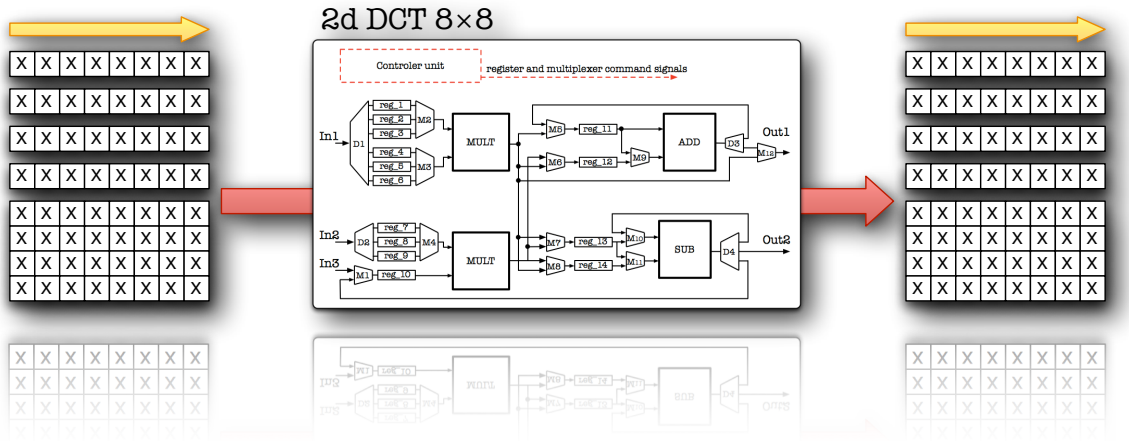


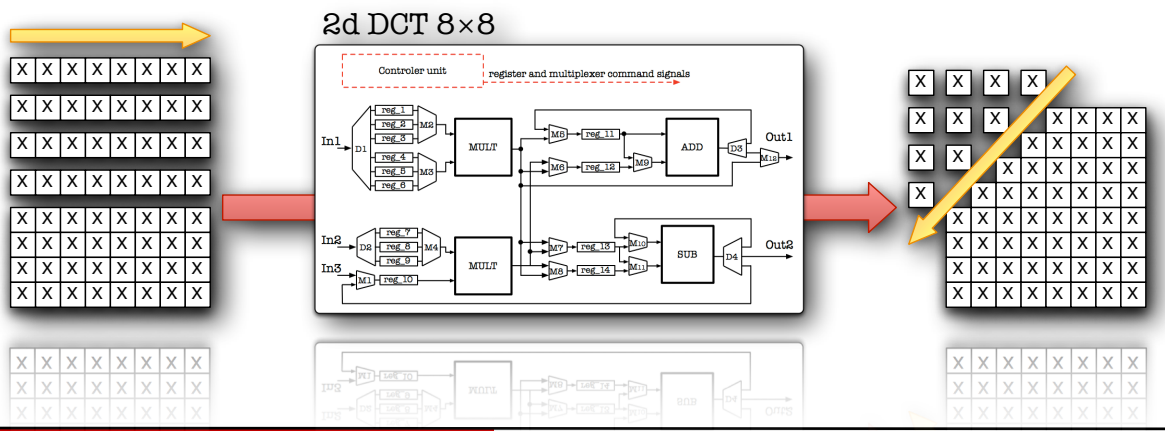
Ressources utilisées:

- 80C : 33757 L.E.
- 120C : 27566 L.E.

Ressources utilisées:

- 80C : 32679 L.E.
- 120C : 27666 L.E.





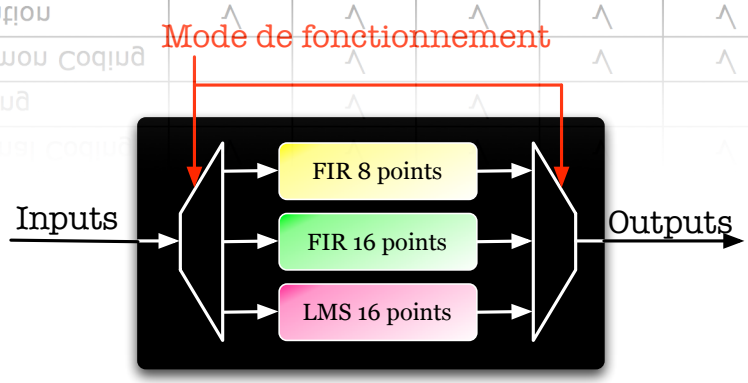
Ressources utilisées:

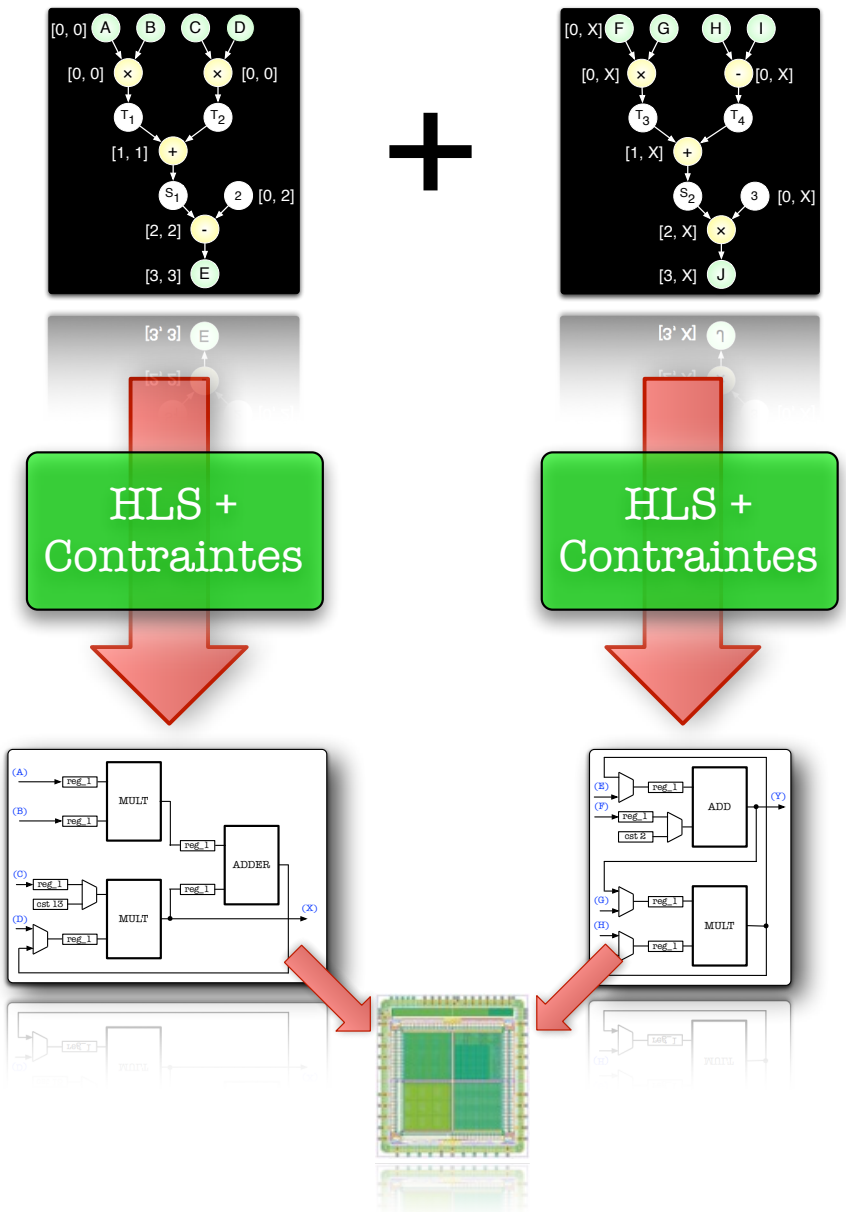
- 80C : 32695 L.E.
- 120C : 24788 L.E.

Partie 3.3
Fonctionnalités multiples

Les circuits multimodes - Contexte de l'étude

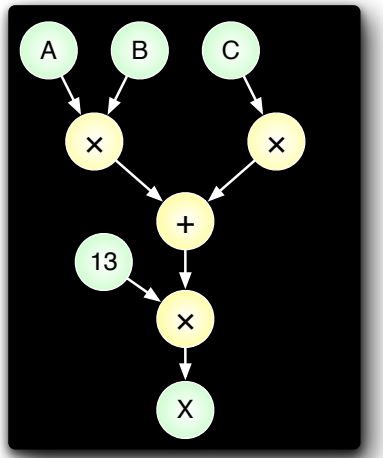
Algorithm	WiFi	WiMax	3G	DVB-T	UWB	60GHz
FIR / IIR	√	√	√	√	√	√
Correlation	√	√	√	√	√	√
Spreading			√			
FFT	√	√		√	√	√
Channel Estimation	√	√	√	√	√	√
QAM Mapping	√	√	√	√	√	√
Interleaving	√	√	√	√	√	√
Convolutional Coding	√	√	√	√	√	√
Turbo Coding		√	√			
Reed-Solomon Coding		√		√	√	√
Randomization	√	√	√	√	√	√
CRC	√	√	√		√	√





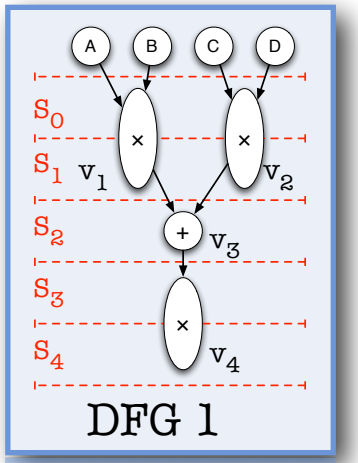
Architecture of the Scalable Communications Core
ACM/IEEE International Symposium on Networks-on-Chip

Approche HLS pour des systèmes monomodes



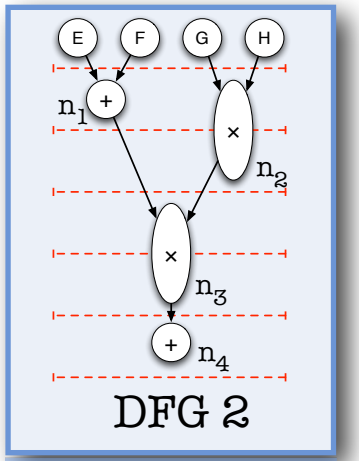
DFG 1

Contrainte temporelle (4 cycles)

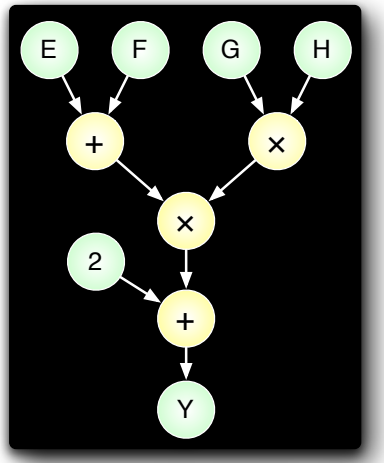


DFG 1

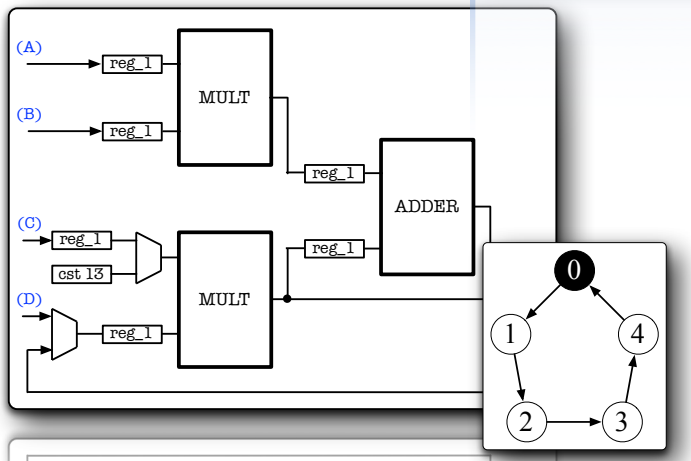
Contrainte matérielle (1 Add, 1 Mult)

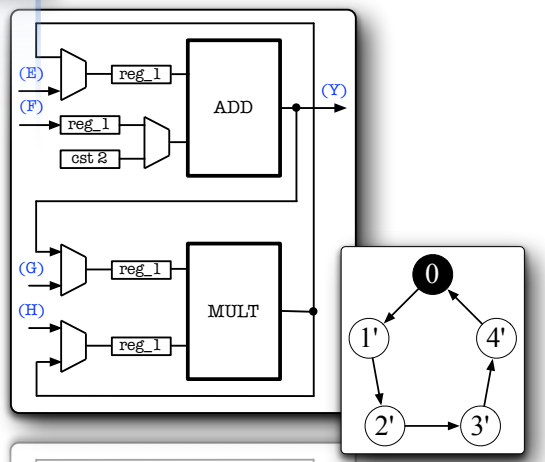


DFG 2

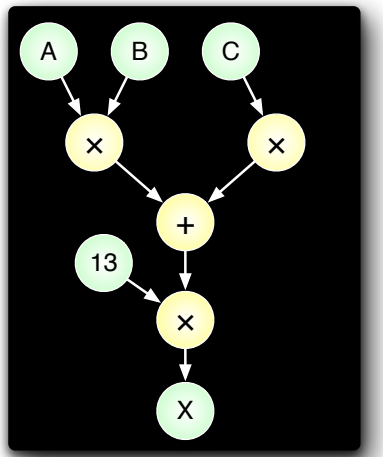


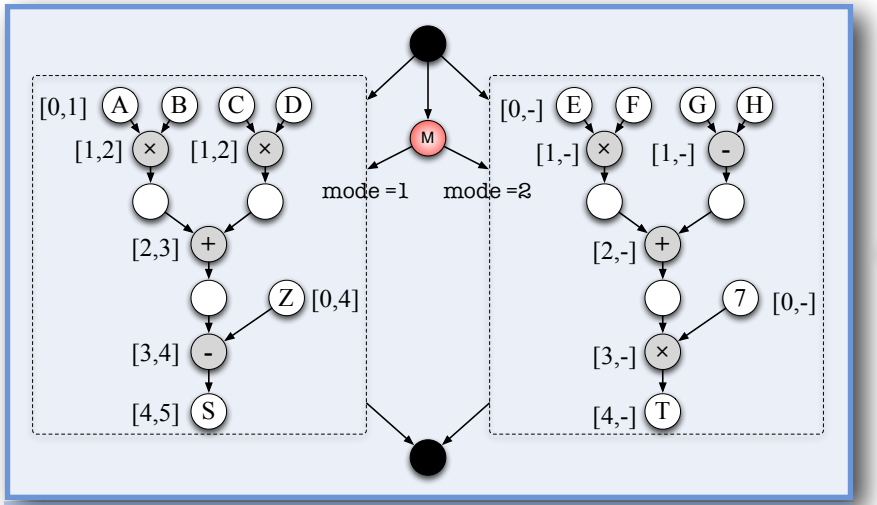
DFG 2

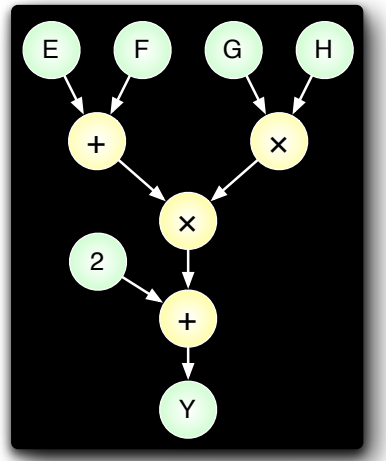




Approche HLS pour des systèmes multimodes



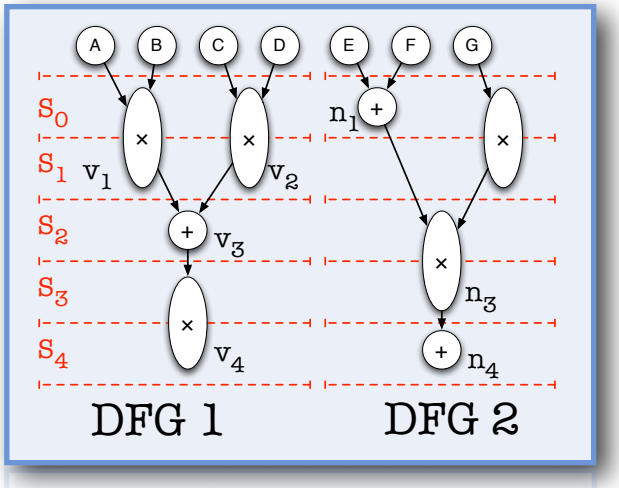


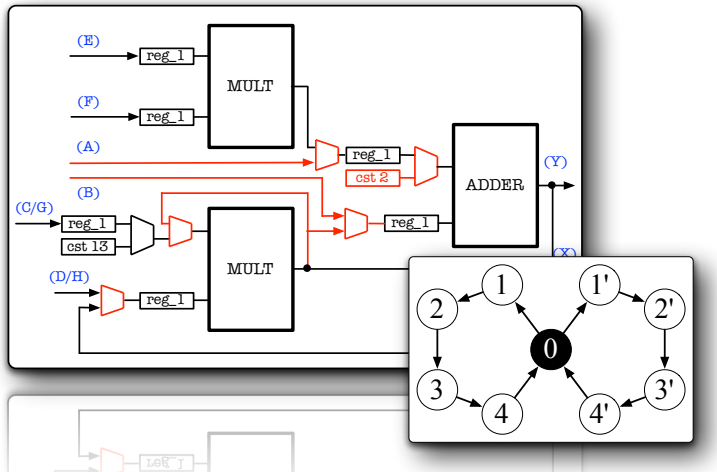


Contrainte temporelle

Contrainte matérielle

Synthèse conjointe avec fusion des ressources matérielles

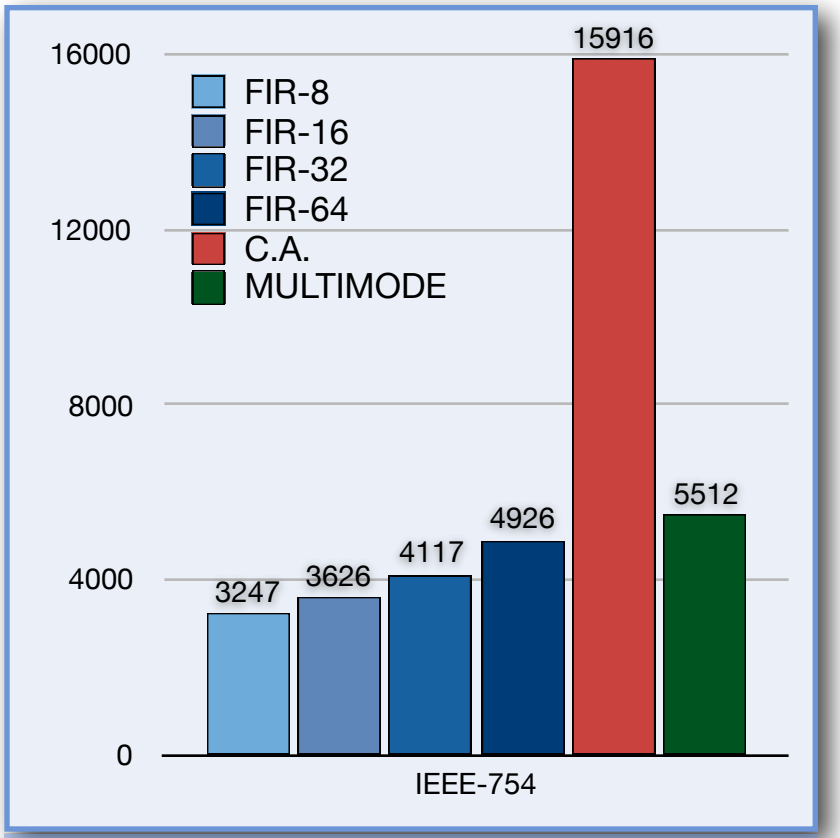




Les circuits multimodes - Performances obtenues...

4 Filtres FIR
(8, 16, 32, 64) points

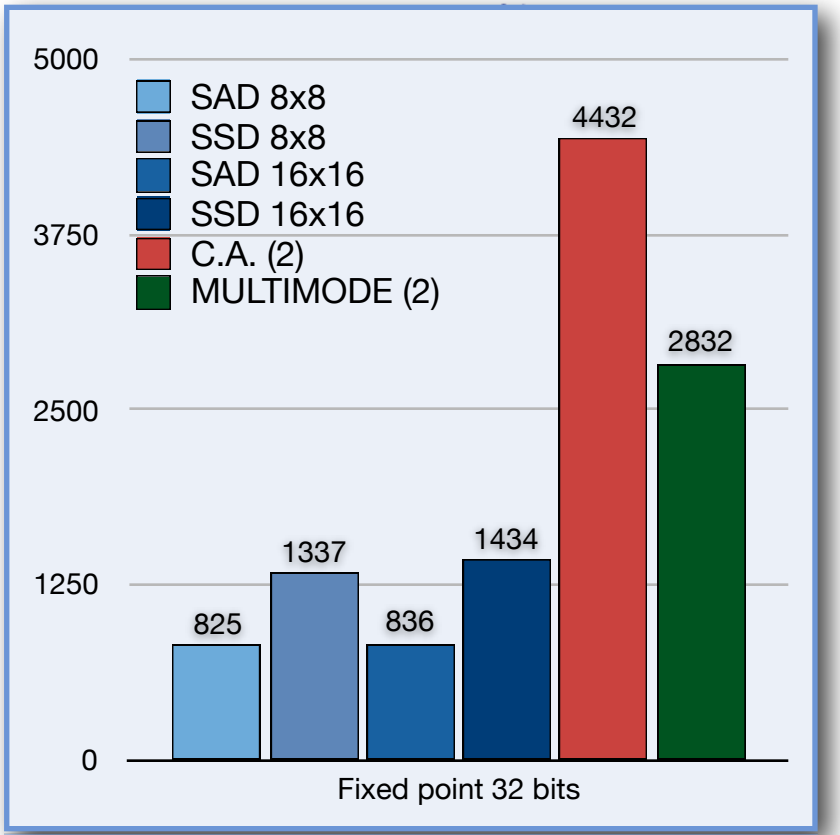
$$y(n) = \sum_{i=0}^{N-1} x(i) \times H(n-i)$$



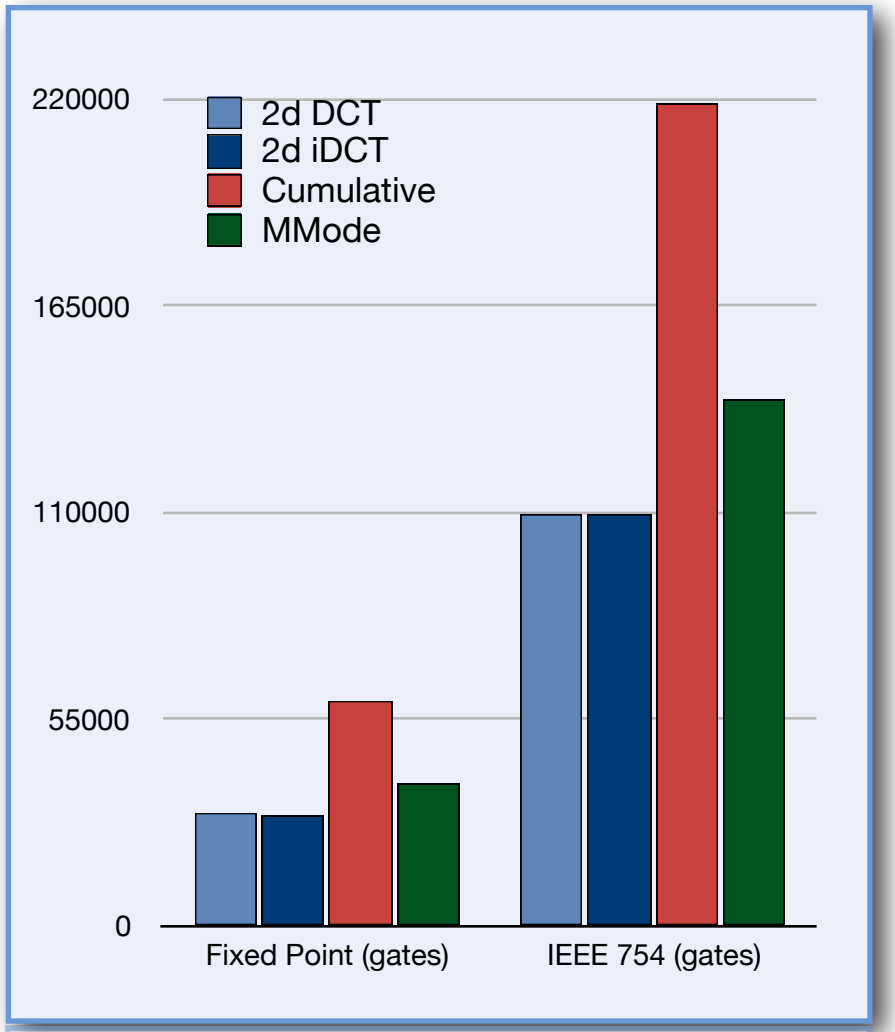
4 Corrélateurs SAD & SSD
(8x8 et 16x16) points

$$\delta = \sum_{y=0}^{7}\sum_{x=0}^{7}(I_1(x,y) - I_2(x,y))^2$$

$$\delta = \sum_{y=0}^{7}\sum_{x=0}^{7}|I_1(x,y) - I_2(x,y)|$$



Les circuits multimodes - Performances obtenues...



Implantation :

- 2d DCT 8x8
- 2d iDCT 8x8

Technologie:

- 65nm low-power ASIC

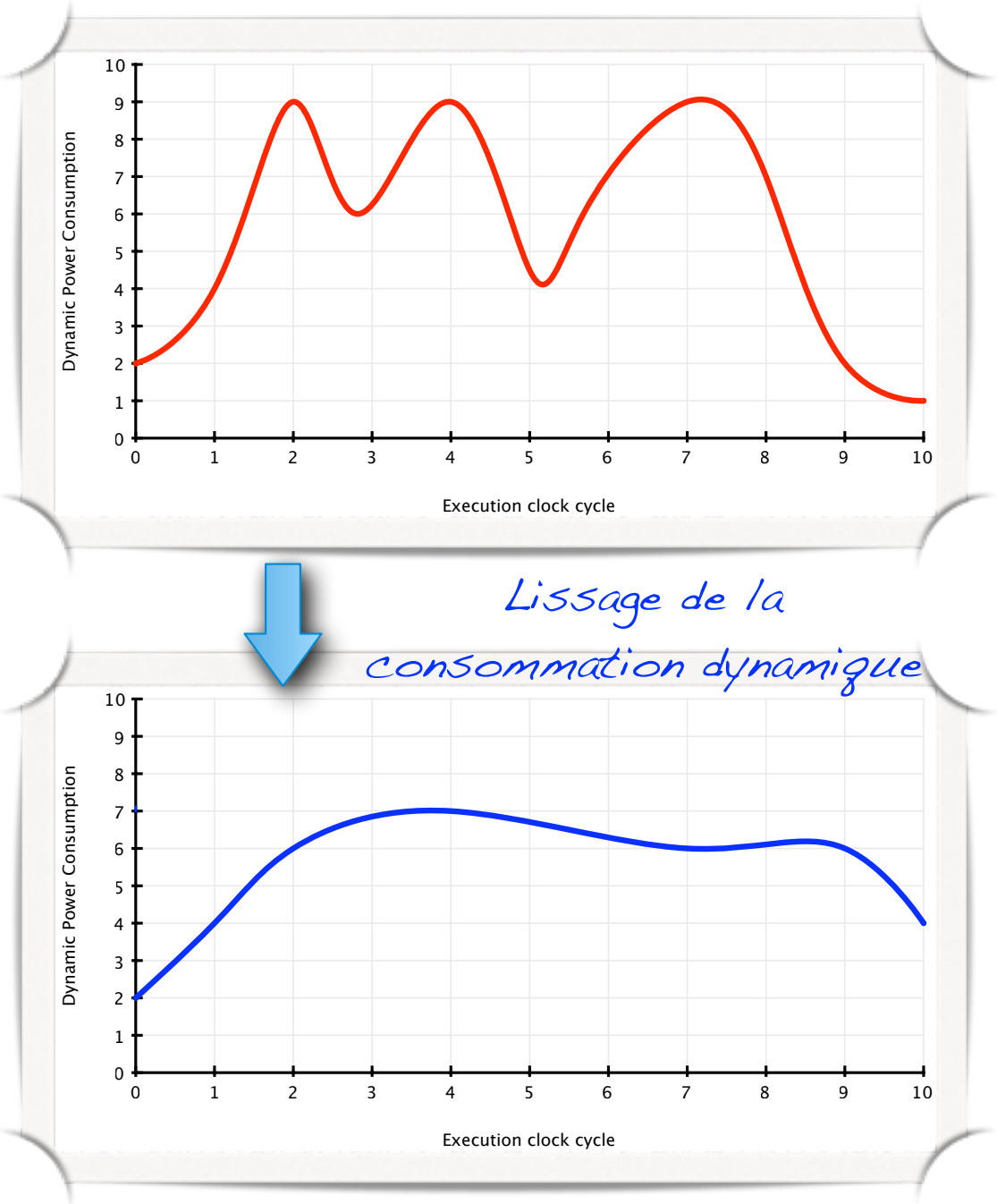
Performances:

- DCT => 41 cycles
- iDCT => 38 cycles

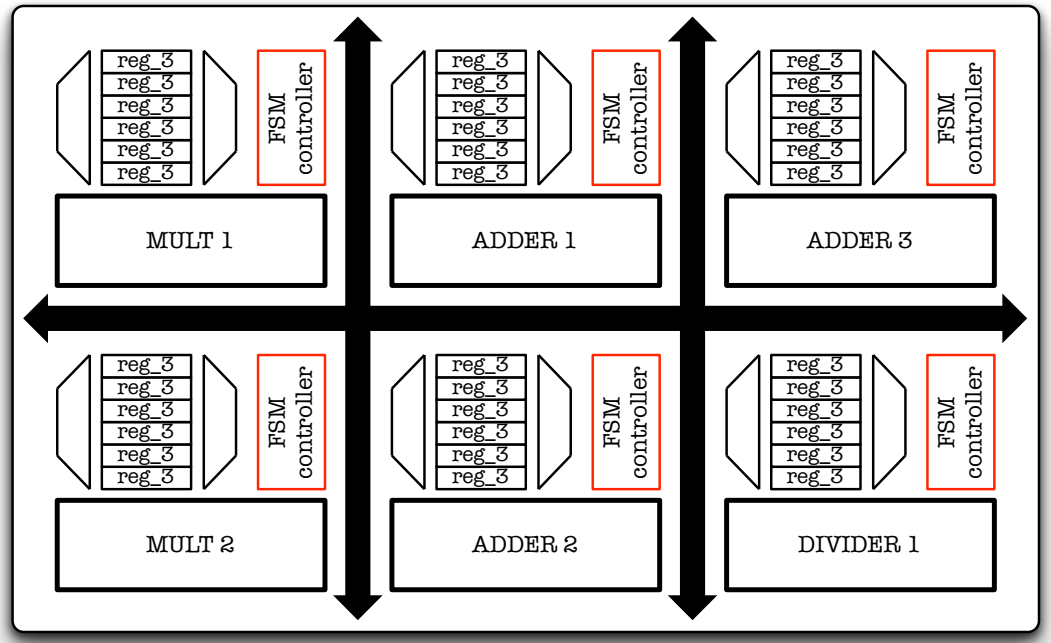
Partie 3.4
Gestion de la consommation d'énergie

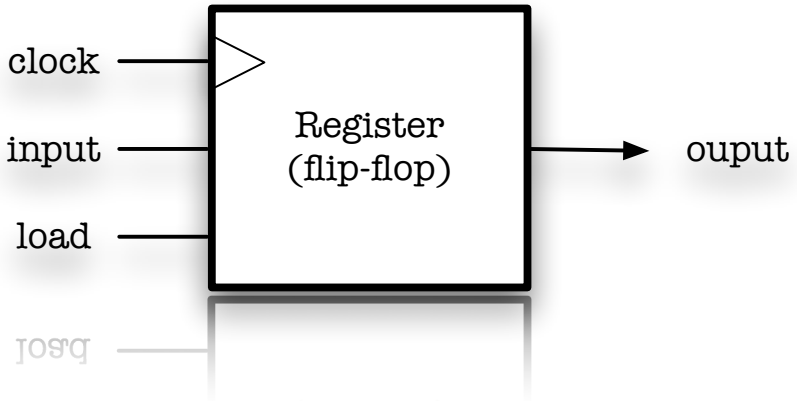
Le problème de la consommation d'énergie

- Les batteries supportent mal les pics de consommation.
- La consommation est une contrainte a part entière,
  - ➡ L'objectif est de lisser la consommation d'énergie sur l'ensemble des cycles d'exécution,
  - ➡ Cela peut impliquer une limitation du nombre de calculs en parallèle et donc une augmentation de la latence,
- Ces techniques permettent en même temps de limiter l'échauffement des circuits.



Réduction de la consommation d'énergie





Le registre consomme de l'énergie lorsque l'horloge commute peu importe la valeur du signal «load»

Le registre consomme de l'énergie uniquement lorsque le signal «load» égal 1 et que l'on a un front montant d'horloge

Partie 3.5
Problématiques de fiabilité

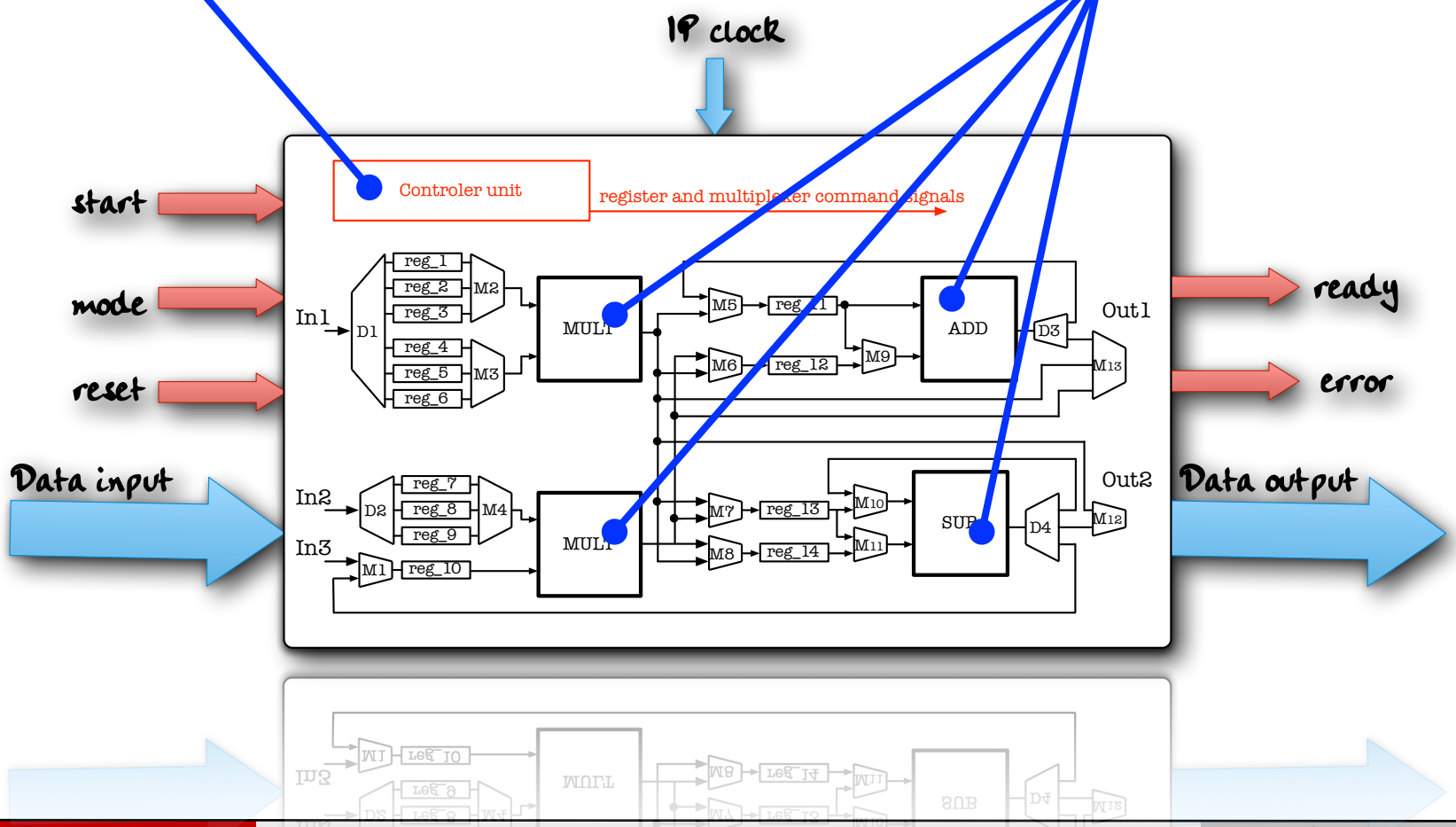
Gestion des contraintes de sureté de fonctionnement

- Certaines classes d'applications (systèmes) ont des besoins très spécifiques en terme de fiabilité,
- Les circuits doivent être «résistants» aux problèmes matériels,
  - Circuits «fault secure»
  - Circuits «fault tolerent»
- Les sources de faute sont différents mais les conséquences sont souvent similaires,

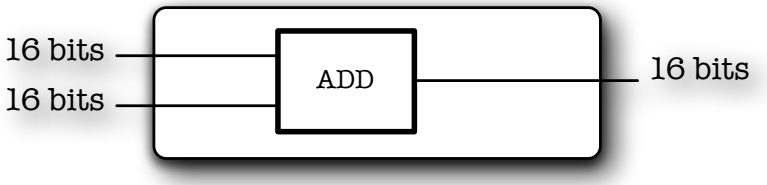
Intégration des contraintes de fiabilité

Transformation de la structure du contrôleur (redondance)

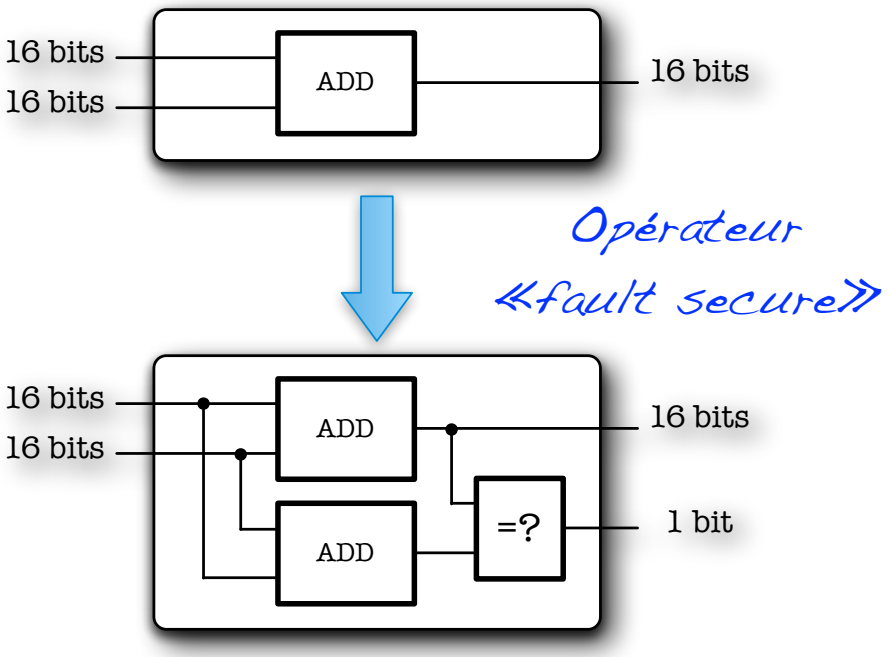
Duplication des calculs (redondance spatiale et/ou temporelle) dans le chemin de données



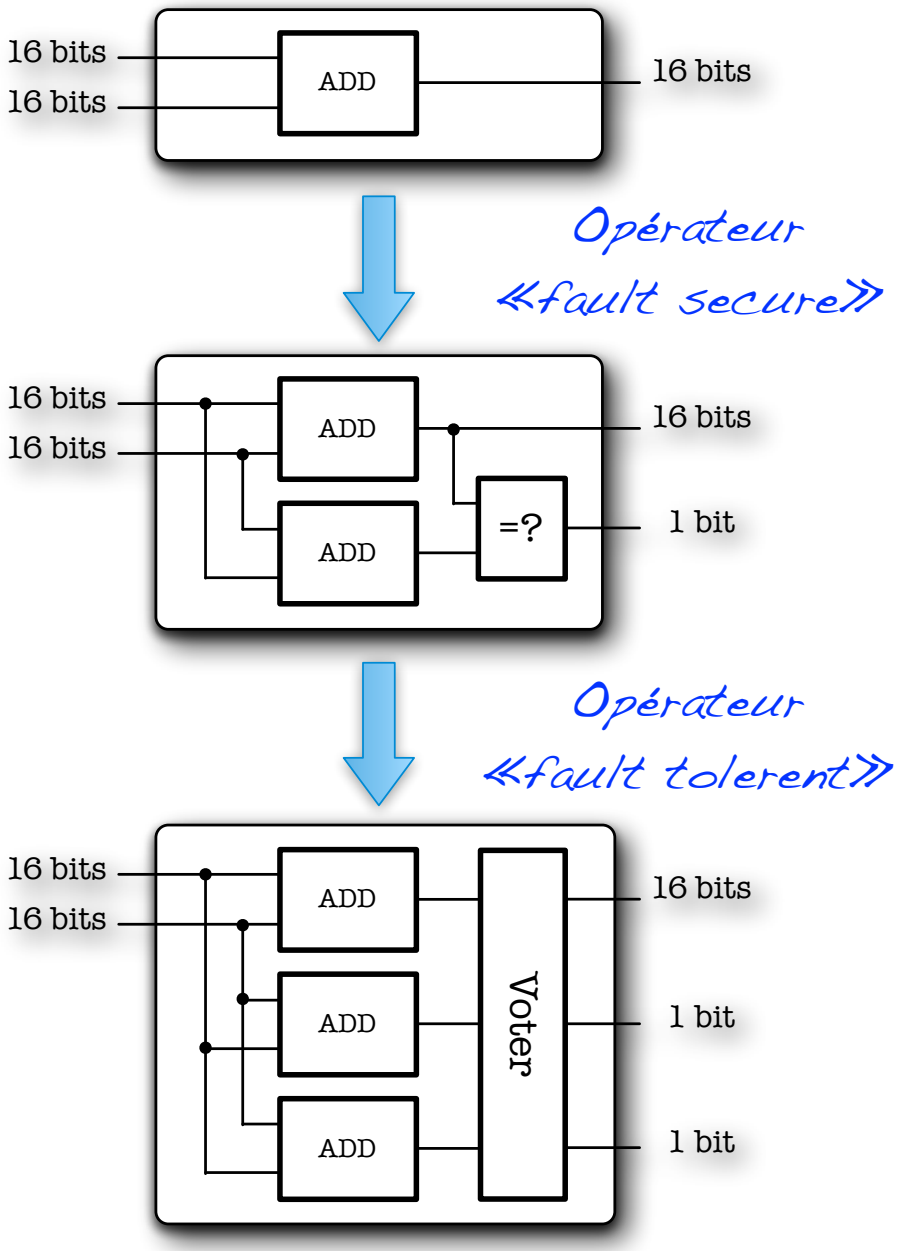
Fiabilisation du chemin de données



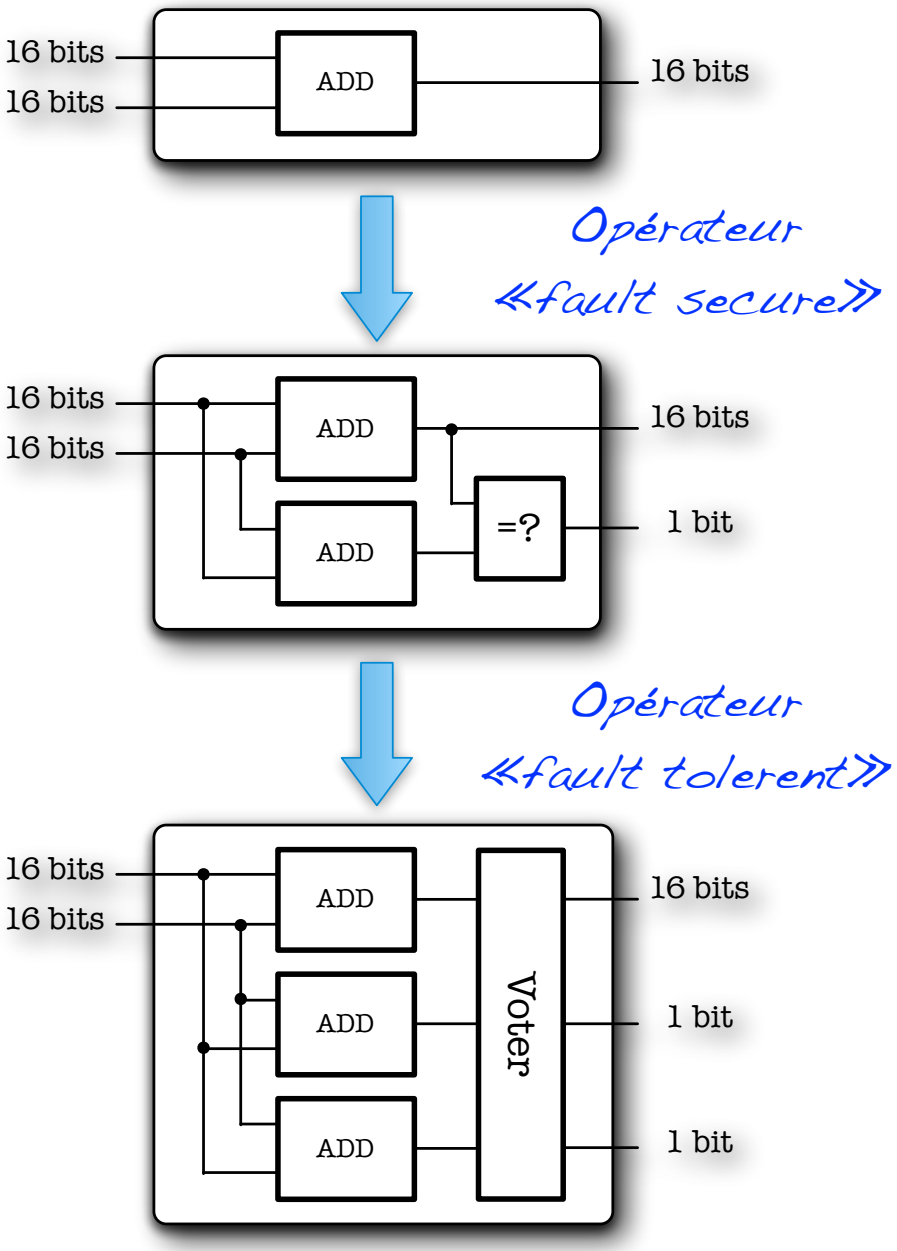
Fiabilisation du chemin de données

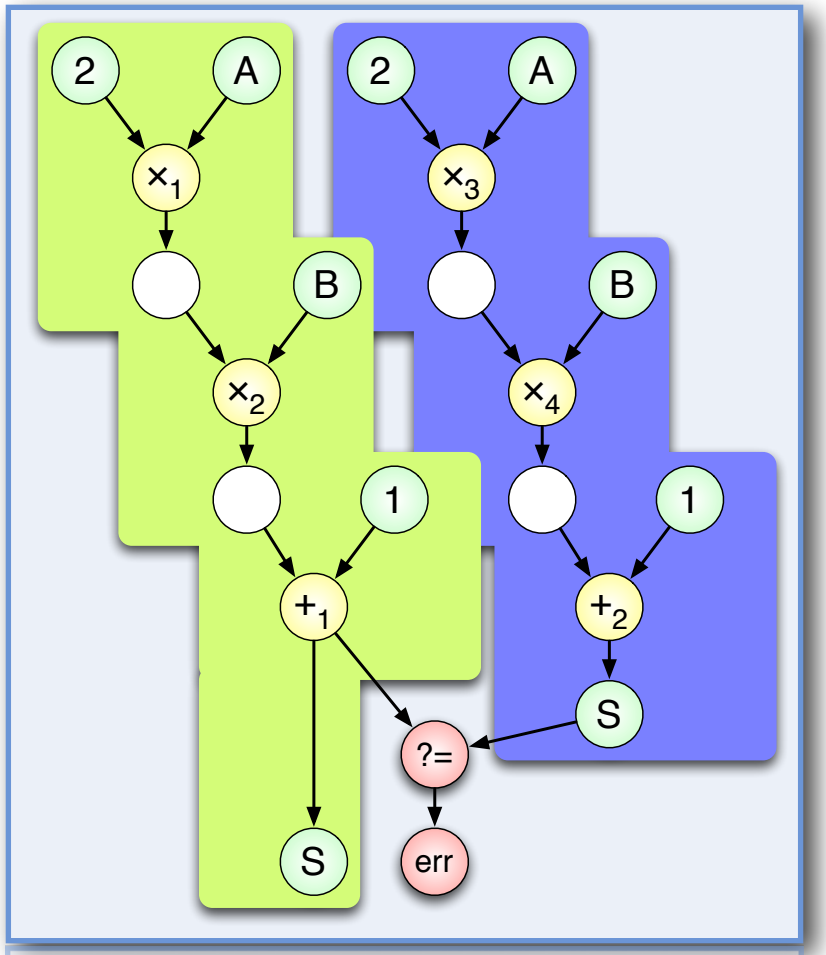


Fiabilisation du chemin de données

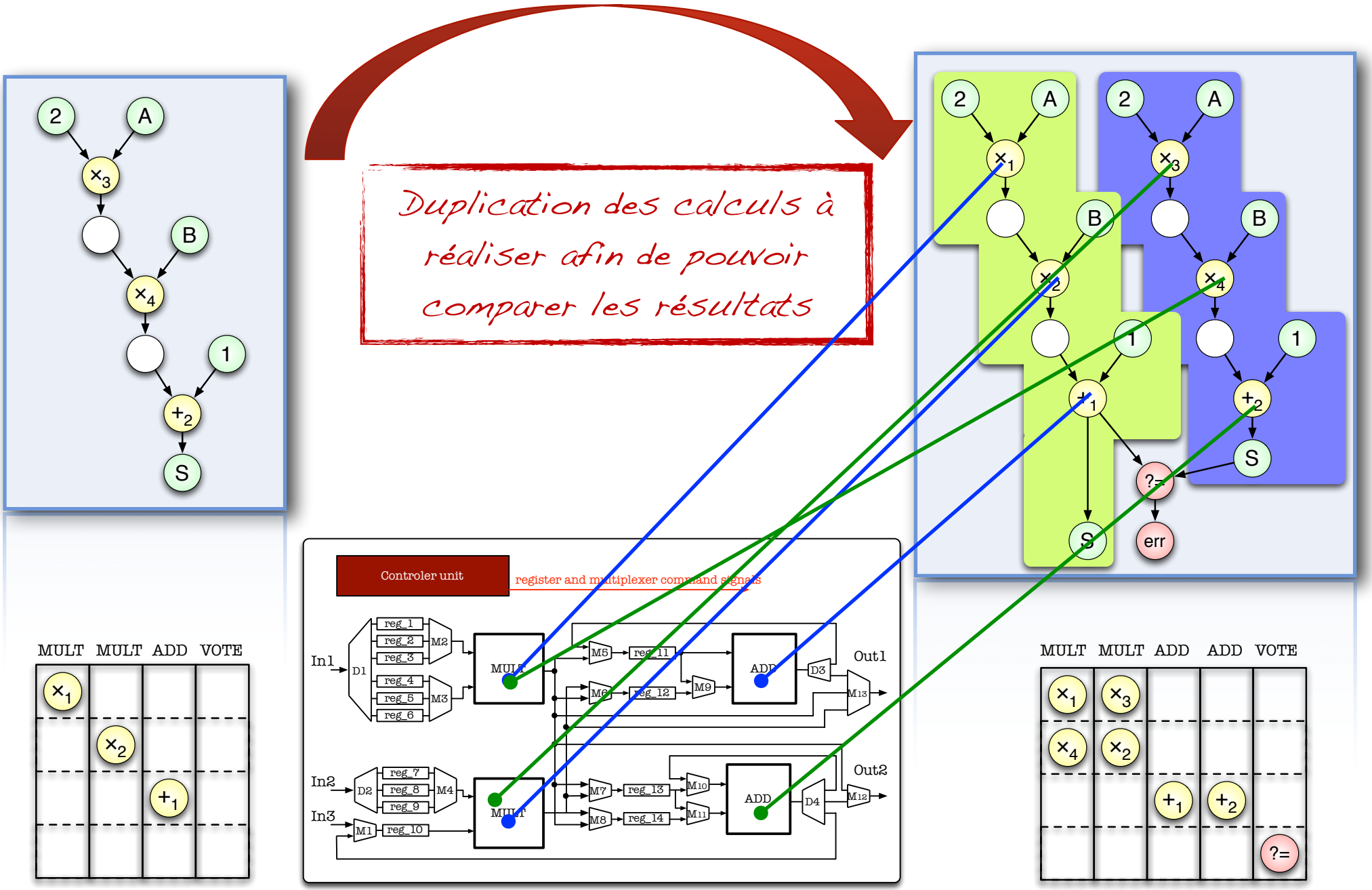


Fiabilisation du chemin de données





Redondance temporelle des calculs



Partie 4:
Conclusion & Perspectives

Quelques outils de HLS disponibles...

- Outils de HLS industriels
  - Mentor Catapult-C,
  - Forte Cynthesizer,
  - Celoxica Handel-C,
  - Synfora Pico,
  - Impulse C™,
  - Et bien d'autres...
- Outils de HLS universitaires
  - GraphLab (Université de Bordeaux),
  - Spark (University Of California),
  - GAUT (Lab-STICC),
  - Et bien d'autres...

Mise en oeuvre par les concepteurs...

- La synthèse de haut-niveau est traitée en recherche depuis plus de 20 ans...
- Les outils ont eu du mal à percer dans le milieu industriel,
  - Architectures sous optimales,
  - Difficultés de mises en oeuvre,
  - Machinerie très complexe à comprendre et à maîtriser,
  - Méthodologies connexes (d'IPs)
- Une problématique majeure restera la simplicité d'utilisation pour le concepteur...

Quel flot utiliser ?

Quel métrique d'ordonnancement prendre ?

Quelles options d'optimisation choisir ?

Quelles contraintes imposer ?

Perspectives de progression de la HLS

- Les flots de conception sont généralement mono-contrainte,
  - ➡ Améliorer les techniques d'optimisation,
  - ➡ Améliorer l'adéquation des méthodes avec les technologies ciblées,
- Gérer de nouvelles contraintes d'intégration,
  - ➡ Sureté de fonctionnement (gestion des erreurs de conception / système)
  - ➡ Sécurité de la propriété intellectuelle
    - Tatouage des circuits générés,
    - Adaptation des techniques aux contraintes des applications cryptographiques,
    - Résistance aux attaques matérielles.