Notes : XDD

Romain PRETET[1] and Camille VEDANI[2]

1 **Université III Paul Sabatier, France,** `romain.pretet@univ-tlse3.fr`

2 **Université III Paul Sabatier, France,** `camille.vedani@univ-tlse3.fr`

Résumé

Ce document relève des prises de notes durant la lecture des articles donnés par Monsieur Cassé. Le but de cela est d'accéder rapidement aux propriétés des XDD pour les optimiser.

Lien pour éditer le projet : `https://fr.overleaf.com/8287738732qzzdrfgbzvqc#929726`

Introduction

Les **eXecution Decision Diagrams (XDD)** sont une structure de données utilisée pour représenter de manière compacte les temps d'exécution d'une séquence d'instructions en présence de latences variables. Ils sont particulièrement utiles dans l'analyse du **Worst-Case Execution Time (WCET)** pour les systèmes temps réel. Ces prises de notes présente les concepts fondamentaux des XDD, leurs propriétés mathématiques, et leur utilisation dans l'analyse des temps d'exécution.

1 Bloc de Base (BB)

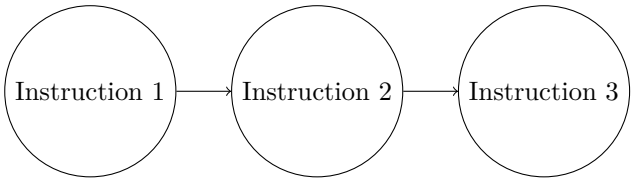
Un **bloc de base (BB)** est une séquence d'instructions dans un programme où :

1. Le flux de contrôle ne peut entrer dans le bloc qu'à partir de la première instruction.
2. Le flux de contrôle ne peut sortir du bloc qu'à partir de la dernière instruction.

En d'autres termes, un BB est une séquence linéaire d'instructions sans sauts (branches) internes, ce qui simplifie l'analyse du temps d'exécution. Les BB sont utilisés dans l'analyse WCET pour diviser le programme en unités plus petites et plus gérables.

2 Notes sur le modèle XG = eXecution Graph

L'idée principale de XG (eXecution Graph) est de modéliser le comportement temporel en tenant compte des dépendances qui apparaissent entre les instructions pendant leur exécution dans les différentes étapes d'un pipeline. Par exemple, une instruction doit quitter une étape du pipeline pour commencer son exécution dans l'étape suivante. De plus, une instruction doit lire un registre après qu'une autre instruction a écrit dans ce registre, etc. Cela aboutit à un graphe de dépendances : un sommet représente l'avancement d'une instruction dans une étape du pipeline, et les arêtes représentent les relations de précédence entre ces sommets.



2.1 Graphe de dépendances

— **Sommets** : Représentent le progrès d'une instruction dans une étape du pipeline.
— **Arêtes** : Représentent les relations de précédence entre les sommets.

2.2 Définition formelle

— I : Ensemble des instructions machine.
— XG : Graphe acyclique orienté (DAG) $G_{XG} = (V_{XG}, E_{XG})$.
— $V_{XG} = \{[I_i/s] | I_i \in Seq \wedge s \in P\}$ où P est l'ensemble des étapes du pipeline.
— $E_{XG} \subset V_{XG} \times V_{XG}$: Ensemble des arêtes basé sur les dépendances du pipeline.

2.3 Informations temporelles

— $\lambda_v \in \mathbb{N}$: Latence du sommet v.
— $\delta_{v \to w} \in \{0, 1\}$: Effet des dépendances de l'arête $v \to w$.
  — $\delta_{v \to w} = 1$: w commence après la fin de v.
  — $\delta_{v \to w} = 0$: w peut commencer en même temps ou après le début de v.

2.4 Exemple de pipeline

— Pipeline à 5 étapes : FE (Fetch), DE (Decode), EX (Execute), ME (Memory), WB (Write-back).
— Pipeline in-order, 2-scalar.

2.5 Types de dépendances

— **Ordre du pipeline** : Une instruction passe par les étapes du pipeline dans l'ordre.
— **Exécution parallèle** : Instructions exécutées en parallèle dans les étapes superscalaires.
— **Limite de capacité** : Limite du nombre d'instructions par cycle.
— **Capacité des files FIFO** : Capacité des files entre les étapes.
— **Dépendances de données** : Une instruction lit un registre écrit par une instruction précédente.

Calcul des temps

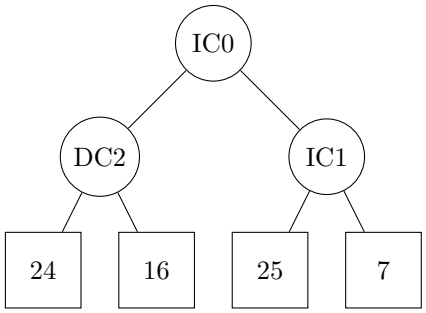
— Temps de début ρ_w et temps de fin ρ_w^* d'une instruction dans une étape.
— $\rho_w = \max_{v \to w \in E_{XG}} \rho_v + \delta_{v \to w} \times \lambda_v$
— $\rho_w^* = \rho_w + \lambda_v$

2.6 Complexité

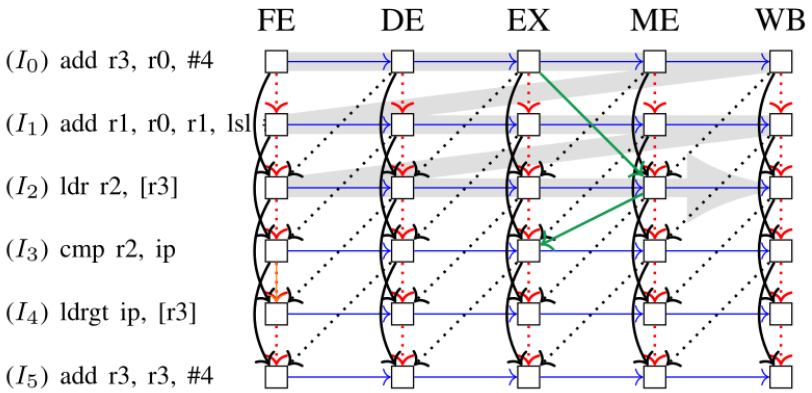
— Calcul rapide et efficace.
— Complexité augmente avec les temps variables (ex : Cache Hits/Misses).

2.7 Schema XDD vs XG

2.7.1 Schema XDD



2.7.2 Schema XG



3 Notes sur les XDD (eXecution Decision Diagram)

4 Définition des XDD

Un XDD est une structure de données récursive qui représente un ensemble de temps induits par différentes configurations d'événements. Il est défini comme suit :

▶ **Définition 1.** Un XDD est défini récursivement par :

$$\text{XDD} = \text{LEAF}(k) \mid \text{NODE}(e, \bar{f}, f)$$

où :

- $k \in \mathbb{Z}$ est une constante représentant un temps d'exécution.
- $e \in \mathcal{E}$ est un événement (par exemple, un accès mémoire qui peut être un hit ou un miss).
- $\bar{f}$ et f sont des sous-XDD représentant les temps d'exécution si l'événement e est inactif ou actif, respectivement.
- Inspiré des BDD (Binary Decision Diagram) et MTBDD (Multi-Terminal BDD).
- DAG défini récursivement :

$$XDD = LEAF(k) \mid NODE(e, f^+, f^-)$$

— $e \in E$: Événements (ex : Cache Hit/Miss).
— $f^+, f^- \in XDD$: Sous-arbres pour e vrai/faux.
— $k \in \mathbb{Z}^\# = \mathbb{Z} \cup \{+\infty, -\infty\}$: Temps d'exécution dans les feuilles.

4.1 Techniques

— **Hash consing** : Unicité des sous-arbres pour compression et accélération.
— **Configurations** : $\gamma \in \Gamma = \mathcal{P}(E)$ (combinaisons d'événements).
— **Isomorphisme** : $XDD \cong (\Gamma \to \mathbb{Z}^\#)$ (compression sans perte).

4.2 Exemple

— Figure 2a : XDD avec 8 configurations.
— Figure 2b : Représentation explicite des configurations et temps.
— Événements : IC (Instruction Cache), DC (Data Cache).

4.3 Opérations

— Opérations binaires sur $\mathbb{Z}^\#$ transférables aux XDD :

$$\forall s_1, s_2 \in (\Gamma \to \mathbb{Z}^\#)^2, \forall \gamma \in \Gamma,$$

$$s_1[\gamma] + s_2[\gamma] = (\alpha(s_1) \otimes \alpha(s_2))[\gamma]$$

$$\max(s_1[\gamma], s_2[\gamma]) = (\alpha(s_1) \oplus \alpha(s_2))[\gamma]$$

— $\oplus$: Opération max (réduction de taille, Figure 2c).
— $\otimes$: Opération addition.

4.4 Avantages

— Représentation efficace des relations entre configurations et temps.
— Calcul précis pour l'analyse XG (règle de résolution des dépendances, Équation 1).
— Support pour l'analyse au niveau CFG et accès hors ordre au bus.

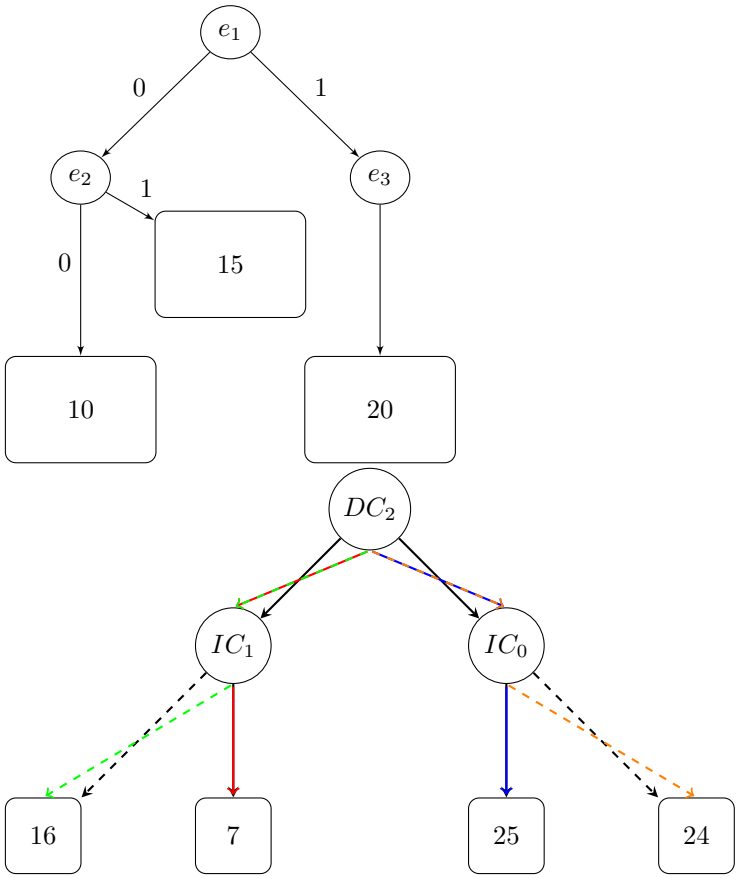
▶ **Exemple 2.** Un XDD représentant le temps d'exécution d'une instruction en fonction de deux événements e_1 et e_2 peut être écrit comme suit :

$$\text{NODE}\left(e_1, \text{NODE}\left(e_2, \text{LEAF}(10), \text{LEAF}(15)\right), \text{LEAF}(20)\right)$$

Cela signifie :
— Si e_1 est inactif, le temps d'exécution est 10 si e_2 est inactif, et 15 si e_2 est actif.
— Si e_1 est actif, le temps d'exécution est 20.

5 Schéma des XDD

Voici un schéma illustrant la structure d'un XDD :



(a) Example XDD

6 Propriétés des XDD

Les XDD possèdent plusieurs propriétés importantes qui garantissent leur efficacité et leur compacité.

6.1 Canonicité

Un XDD est dit **canonique** s'il satisfait les propriétés suivantes :

1. **Ordre** : Les événements dans un XDD suivent un ordre total prédéfini.
2. **Compacité** : Aucun nœud dans un XDD ne peut avoir deux sous-XDD identiques.

▶ **Définition 3.** Un XDD est canonique si pour tout nœud NODE$(e, \bar{f}, f)$:
— Les sous-XDD $\bar{f}$ et f sont distincts.
— Les événements dans les sous-XDD suivent l'ordre total prédéfini.

6.2 Opérations sur les XDD

Les XDD supportent deux opérations principales : l'addition ($\otimes$) et le maximum ($\oplus$). Ces opérations sont définies récursivement comme suit :

▶ **Définition 4.** $\forall f \in XDD, \gamma \in \Gamma$,

$$f^{[\gamma]} = \begin{cases} k & \text{si } f = \text{LEAF}(k) \\ \bar{g}^{[\gamma]} & \text{si } f = \text{NODE}(e, \bar{g}, g) \text{ et } \neg\gamma(e) \\ g^{[\gamma]} & \text{si } f = \text{NODE}(e, \bar{g}, g) \text{ et } \gamma(e) \end{cases}$$

▶ **Définition 5.** Pour deux XDD f_1 et f_2, et une opération binaire $\square$ sur $\mathbb{Z}$:

$$f_1 \odot f_2 = \begin{cases} \text{LEAF}(k_1 \square k_2) & \text{si } f_1 = \text{LEAF}(k_1) \text{ et } f_2 = \text{LEAF}(k_2) \\ g_1 \odot g_2 & \text{si } f_1 = \text{NODE}(e, \bar{g_1}, g_1) \text{ et } f_2 = \text{NODE}(e, \bar{g_2}, g_2) \\ f_1 \odot \bar{g_2} & \text{si } f_2 = \text{NODE}(e, \bar{g_2}, g_2) \text{ et } \text{evt}(f_1) \leq e \\ \bar{g_1} \odot f_2 & \text{si } f_1 = \text{NODE}(e, \bar{g_1}, g_1) \text{ et } \text{evt}(f_2) \leq e \\ \text{NODE}(e, \bar{g_1} \odot \bar{g_2}, g_1 \odot g_2) & \text{si } f_1 = \text{NODE}(e, \bar{g_1}, g_1) \text{ et } f_2 = \text{NODE}(e, \bar{g_2}, g_2) \\ \text{NODE}(e, \bar{g_1} \odot \bar{g_2}, g_1 \odot g_2) & \text{si } f_1 = \text{NODE}(e, \bar{g_1}, g_1) \text{ et } f_2 = \text{NODE}(e, \bar{g_2}, g_2) \\ \text{NODE}(e_2, f_1 \odot \bar{g_2}, f_1 \odot g_2) & \text{si } f_2 = \text{NODE}(e_2, \bar{g_2}, g_2) \text{ et } \text{evt}(f_1) < e_2 \\ \text{NODE}(e_1, f_2 \odot \bar{g_1}, f_2 \odot g_1) & \text{si } f_1 = \text{NODE}(e_1, \bar{g_1}, g_1) \text{ et } \text{evt}(f_2) < e_1 \end{cases}$$

▶ **Exemple 6.** Pour deux XDD $f_1 = \text{NODE}(e_1, \text{LEAF}(10), \text{LEAF}(20))$ et $f_2 = \text{NODE}(e_1, \text{LEAF}(15), \text{LEAF}(25))$, l'addition $f_1 \otimes f_2$ donne :

$$f_1 \otimes f_2 = \text{NODE}(e_1, \text{LEAF}(25), \text{LEAF}(45))$$

▶ **Définition 7.** Nous définissons d'abord $\lambda_e^\# : \mathcal{E} \to$ xDD, convertissant en un xDD un événement e qui a un coût de k_e lorsqu'il est actif et 0 lorsqu'il est inactif.

$$\lambda_e^\# = \text{NODE}(e, \text{LEAF}(0), \text{LEAF}(k_e))$$

λ_v, le temps passé dans un nœud xG pour une configuration particulière, peut maintenant être représenté par $\lambda_v^\#$.

▶ **Définition 8.** Si le nœud v subit un ensemble d'événements $\mathcal{E}_v$, $\lambda_v^\#$ est exprimé par :

$$\lambda_v^\# = \text{LEAF}(\lambda_v) \otimes \bigotimes_{e \in \mathcal{E}_v} \lambda_e^\#$$

Le temps passé dans une étape est le temps par défaut passé dans l'étape plus la somme de tous les coûts d'événements possibles.

Le texte décrit une méthode d'analyse des graphes d'exécution (XG) pour évaluer le comportement des instructions dans un pipeline de processeur. Cette analyse est divisée en quatre étapes principales :

Étape 1 : Attente des dépendances

— Avant qu'une instruction ne soit exécutée dans une étape du pipeline, elle doit attendre que toutes ses dépendances soient satisfaites.
— Le pointeur de temps est réinitialisé à zéro (ou à $-\infty$).
— Chaque temps de dépendance est accumulé dans le pointeur de temps à l'aide de l'opérateur $\oplus$.

Étape 2 : Mise à jour des dépendances

— Certaines ressources sont libérées au début d'un nœud XG.
— Les dépendances correspondantes (comme l'ordre du programme et la capacité de la file d'attente) sont mises à jour avec le temps de début ρ.

Étape 3 : Exécution de l'instruction

— L'instruction en cours passe un certain nombre de cycles $\lambda[I_i/s]$ dans l'étape.
— Le temps passé est calculé et consommé.

Étape 4 : Fin de l'exécution

— L'instruction termine son exécution et les dépendances enregistrant le temps de fin du nœud actuel sont mises à jour.

Le modèle computationnel utilise des diagrammes de décision d'exécution (xDD) pour représenter les temps d'exécution possibles, en tenant compte des variations de temps dues aux événements. Les fonctions de transition τ sont utilisées pour mettre à jour l'état temporel, et ces fonctions peuvent être exprimées comme des multiplications de matrices, ce qui permet d'accélérer l'analyse du pipeline au niveau du graphe de flux de contrôle (CFG).

▶ **Définition 9.** La multiplication scalaire est définie par :

$$\cdot : \mathrm{XDD}^N \times \mathrm{XDD}^N \to \mathrm{XDD},$$

$$[f_0, f_1, \ldots, f_{N-1}] \cdot [f'_0, f'_1, \ldots, f'_{N-1}] = \bigoplus_{0 \le i \le N-1} f_i \otimes f'_i$$

▶ **Définition 10.** La multiplication matricielle est définie par :

$$\cdot : \mathrm{XDD}^{N \times M} \times \mathrm{XDD}^{M \times L} \to \mathrm{XDD}^{N \times L},$$

$$B \cdot C = [A_{i,j}] \text{ où } A_{i,j} = \bigoplus_{1 \le k \le M} B_{i,k} \otimes C_{k,j}$$

▶ **Définition 11.** **Définition III.4**

La matrice identité Id sur le semiring XDD est définie par :

$$Id = [A_{i,j}] \text{ où } A_{i,j} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

Remarquez que, par définition, $\vec{S} \cdot Id = \vec{S}$: toute colonne de matrice à l'index i composée de 0 sauf avec un 1 dans la ligne i maintient inchangé la valeur de $\vec{S}[i]$ dans le vecteur résultant. Pour implémenter les fonctions de transition τ comme des multiplications de matrices, la matrice Id est prise comme base et seules les cellules qui ont un effet sur le vecteur doivent être changées.

Réinitialisation du Timer (Équation 12)

Pour réinitialiser le pointeur de temps dans le vecteur d'état temporel, on utilise une matrice de réinitialisation M_{reset}. Cette matrice est basée sur la matrice identité, mais avec un 0 à la position du pointeur de temps.

$$\tau_{reset}(\vec{S}) = \vec{S} \cdot M_{reset}$$

$$= \vec{S} \cdot [A_{i,j}] \text{ où } A_{i,j} = \begin{cases} 0 & \text{si } i = j = i_\rho \\ Id_{i,j} & \text{sinon} \end{cases}$$

7 Utilisation des XDD dans l'analyse WCET

Les XDD sont utilisés pour représenter les temps d'exécution des blocs de base (BB) dans l'analyse WCET. Ils permettent de prendre en compte les latences variables dues aux événements tels que les accès mémoire (cache hit/miss) ou les prédictions de branchement.

7.1 Calcul du WCET avec XDD

Le WCET d'un BB est calculé en utilisant les XDD pour représenter les temps d'exécution pour chaque configuration d'événements. Les opérations $\otimes$ et $\oplus$ sont utilisées pour combiner les temps d'exécution des instructions en tenant compte des dépendances dans le pipeline.

▶ **Exemple 12.** Pour un BB avec deux instructions I_1 et I_2, les temps d'exécution peuvent être représentés par des XDD f_1 et f_2. Le WCET du BB est calculé comme suit :

$$\text{WCET} = f_1 \oplus f_2$$

8 Conclusion

Les XDD sont une structure de données puissante pour représenter de manière compacte les temps d'exécution en présence de latences variables. Ils permettent de réduire la complexité combinatoire de l'analyse WCET et d'améliorer la précision des estimations. Les propriétés mathématiques des XDD garantissent leur efficacité et leur compacité, ce qui les rend particulièrement adaptés à l'analyse des systèmes temps-réel.