Recompilation et optimisation dynamique de code

Florent Bouchez

Présentation du Lundi 31 janvier 2005

Plan

Plan

Optimisation de code : à la compilation.

Mais il existe aussi des méthodes dynamiques, pendant l'exécution :

- techniques matérielles (exécution dans le désordre, prédiction de branches)
- optimisations dynamiques logicielles (utilisation de traces pour créer du code linéaire)
- compilation dynamique (compilateurs JIT, Crusoe^{TM})

Seule la première est utilisée industriellement.

Présentation

Optimisation de code : à la compilation.

Mais il existe aussi des méthodes dynamiques, pendant l'exécution :

- techniques matérielles (exécution dans le désordre, prédiction de branches)
- optimisations dynamiques logicielles (utilisation de traces pour créer du code linéaire)
- compilation dynamique (compilateurs JIT, Crusoe^{TM})

Seule la première est utilisée industriellement.

Présentation

Optimisation de code : à la compilation.

Mais il existe aussi des méthodes dynamiques, pendant l'exécution :

- techniques matérielles (exécution dans le désordre, prédiction de branches)
- optimisations dynamiques logicielles (utilisation de traces pour créer du code linéaire)
- compilation dynamique (compilateurs JIT, Crusoe^{TM})

Seule la première est utilisée industriellement.

Plan

Principe

Les processeurs Crusoe sont performants et faibles consommateurs.

- un assembleur compatible x86
- mais un CPU de type VLIW avec quatre unités parallèles

Floating-Point unit	Integer ALU unit	Load/Store unit	Branch unit

- *CodeMorphing*™ + mémoire cache

Principe

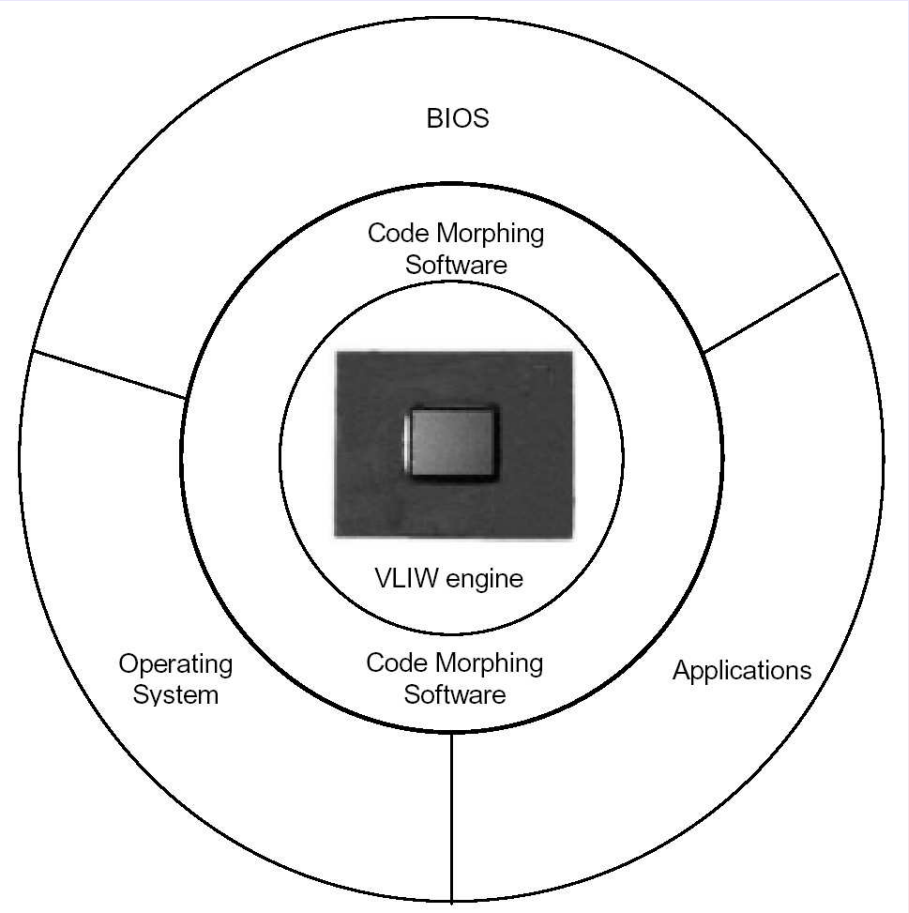
Les processeurs Crusoe sont performants et faibles consommateurs.

- un assembleur compatible x86
- mais un CPU de type VLIW avec quatre unités parallèles

Floating-Point unit	Integer ALU unit	Load/Store unit	Branch unit

- *CodeMorphing*™ + mémoire cache

CodeMorphing



Avantages

- changement d'architecture ne nécessite que le changement du logiciel
- mais cycles « perdus » pour exécuter le logiciel
- et permet les optimisations dynamiques

Plan

Dynamo

Dynamo est un projet d'optimisation dynamique de code (1996).

Entrée : code natif PA-8000
Sortie : code natif PA-8000

Interprétation de code. Lent.

Dynamo

Dynamo est un projet d'optimisation dynamique de code (1996).

Entrée : code natif PA-8000
Sortie : code natif PA-8000

Interprétation de code. Lent.

Schéma

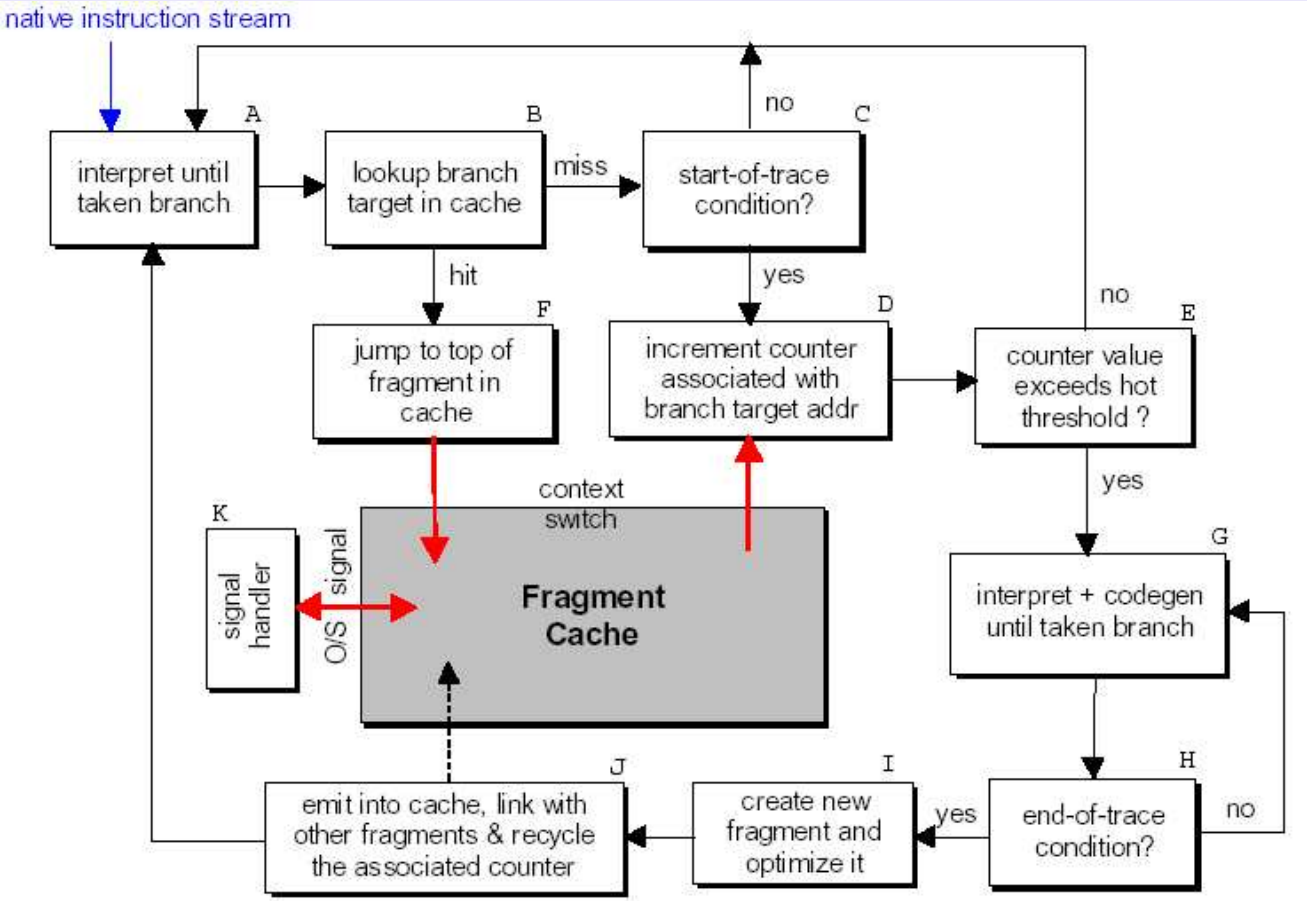
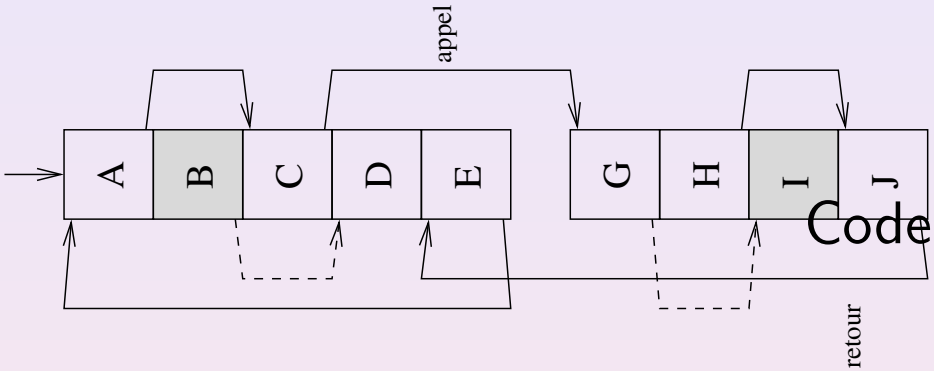


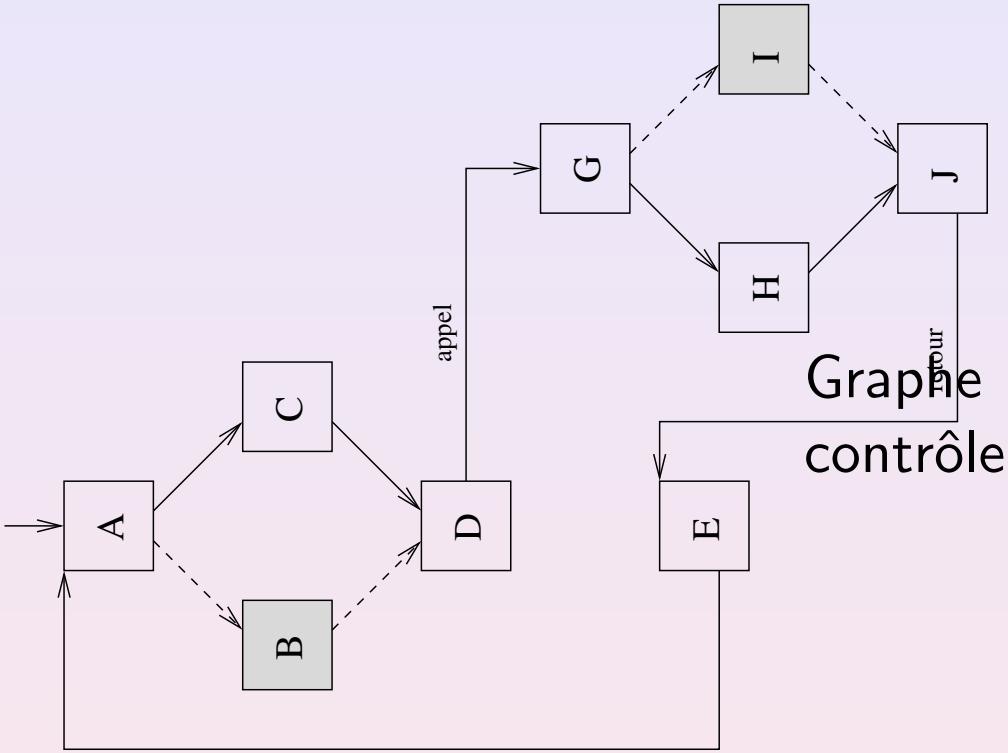
Figure 1. How Dynamo works

Exemple



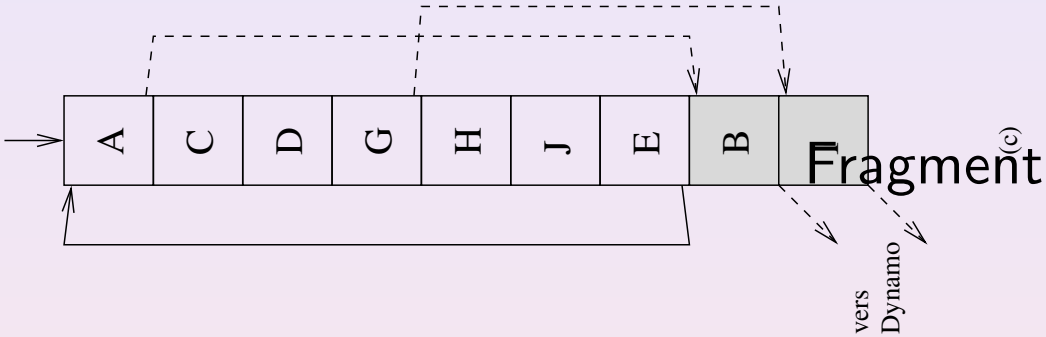
(b) Code natif initial.

Exemple



Graphe de flot de contrôle. (a)

Exemple



(c) Fragment optimisé.

Plan

Deli

Deli un un projet plus récent (2002). (HP encore)

- Pas d'interprétation mais points de contrôle.
- Fournit une API pour l'observation et la modification des instructions.

Plan

Problèmes

Seules les techniques matérielles sont utilisées industriellement.

	problème	Optimisation dynamique	Compilation dynamique
1.	sur-coût	interprétation	spécialisation du code
2.	décisions	faibles	annotations
3.		optimisation aggressive	code source disponible
4.	fiabilité	bogues non-reproductibles	bogues

Problèmes

Seules les techniques matérielles sont utilisées industriellement.

	problème	Optimisation dynamique	Compilation dynamique
1.	sur-coût	interprétation	spécialisation du code
2.	décisions	faibles	annotations
3.		optimisation aggressive	code source disponible
4.	fiabilité	bogues non-reproductibles	bogues

Problèmes

Seules les techniques matérielles sont utilisées industriellement.

	problème	Optimisation dynamique	Compilation dynamique
1.	sur-coût	interprétation	spécialisation du code
2.	décisions	faibles	annotations
3.		optimisation aggressive	code source disponible
4.	fiabilité	bogues non-reproductibles	bogues

Problèmes

Seules les techniques matérielles sont utilisées industriellement.

	problème	Optimisation dynamique	Compilation dynamique
1.	sur-coût	interprétation	spécialisation du code
2.	décisions	faibles	annotations
3.		optimisation aggressive	code source disponible
4.	fiabilité	bogues non-reproductibles	bogues

Problèmes

Seules les techniques matérielles sont utilisées industriellement.

	problème	Optimisation dynamique	Compilation dynamique
1.	sur-coût	interprétation	spécialisation du code
2.	décisions	faibles	annotations
3.		optimisation aggressive	code source disponible
4.	fiabilité	bogues non-reproductibles	bogues

Améliorations

David Hodgdon avance l'idée de la coopération :

1. le compilateur statique sait mais partiellement ;
2. le dynamisme n'a pas de vue d'ensemble.

Le compilateur peut alors laisser des « notes » à l'optimiseur dynamique.

Améliorations

David Hodgdon avance l'idée de la coopération :

1. le compilateur statique sait mais partiellement ;
2. le dynamisme n'a pas de vue d'ensemble.

Le compilateur peut alors laisser des « notes » à l'optimiseur dynamique.

Améliorations

1. Constantes à l'exécution : définies une fois pour toutes, mais à l'exécution.
2. Accès mémoire : ordonner plus tôt les *load*.
3. Limites : *bound-checking* évite les bogues incompréhensibles, mais sur-coût.
4. Environnements distribués : calculs similaires sur plusieurs processus : partage des traces.

Améliorations

1. Constantes à l'exécution : définies une fois pour toutes, mais à l'exécution.
2. Accès mémoire : ordonner plus tôt les *load*.
3. Limites : *bound-checking* évite les bogues incompréhensibles, mais sur-coût.
4. Environnements distribués : calculs similaires sur plusieurs processus : partage des traces.

Améliorations

1. Constantes à l'exécution : définies une fois pour toutes, mais à l'exécution.
2. Accès mémoire : ordonner plus tôt les *load*.
3. Limites : *bound-checking* évite les bogues incompréhensibles, mais sur-coût.
4. Environnements distribués : calculs similaires sur plusieurs processus : partage des traces.

Améliorations

1. Constantes à l'exécution : définies une fois pour toutes, mais à l'exécution.
2. Accès mémoire : ordonner plus tôt les *load*.
3. Limites : *bound-checking* évite les bogues incompréhensibles, mais sur-coût.
4. Environnements distribués : calculs similaires sur plusieurs processus : partage des traces.

Plan

Conclusion

- Compilateurs et optimiseurs dynamiques améliorent l'exécution du code.
- Utilisation de traces importante.
- Coopération statique/dynamique mènerait à plus d'améliorations.
- Influence du matériel nécessaire (mémoire cache) sur le processeur ?