

# LLVM - Présentation du framework

Auroux Lionel

9 juillet 2011

## LLVM - Présentation du framework

Auroux Lionel

### État de l'art sur les compilateurs

Présentation

Processus classique de  
compilation

LLVM

Questions

1 État de l'art sur les compilateurs

2 LLVM

3 Questions

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

Présentation

Processus classique de  
compilation

LLVM

Questions

- 1 État de l'art sur les compilateurs
  - Présentation
  - Processus classique de compilation

- 1 Transformer un langage source vers un langage cible ;
- 2 Plus généralement, d'un langage de haut niveau vers un langage machine.

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

Présentation

Processus classique de  
compilation

LLVM

Questions

## LLVM - Présentation du framework

Auroux Lionel

### État de l'art sur les compilateurs

#### Présentation

Processus classique de  
compilation

#### LLVM

#### Questions

- 1 Transformer un langage source vers un langage cible ;
- 2 Plus généralement, d'un langage de haut niveau vers un langage machine.

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

Présentation

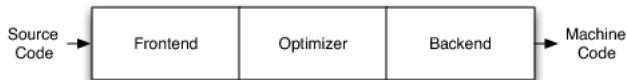
Processus classique de  
compilation

LLVM

Questions

- 1 État de l'art sur les compilateurs
  - Présentation
  - **Processus classique de compilation**

Ce qui donne le plus souvent ce modèle :



Toutefois on va chercher

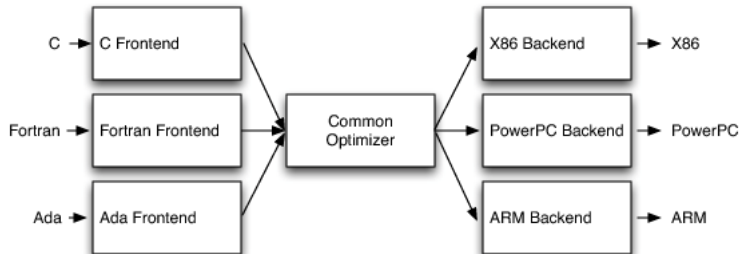
- ❶ Écrire une seule fois les passes d'optimisation ;
- ❷ Limiter le travail pour chaque nouveau langage ;
- ❸ Limiter le travail pour chaque nouvelle architecture.

Toutefois on va chercher

- ① Écrire une seule fois les passes d'optimisation ;
- ② Limiter le travail pour chaque nouveau langage ;
- ③ Limiter le travail pour chaque nouvelle architecture.

Toutefois on va chercher

- ① Écrire une seule fois les passes d'optimisation ;
- ② Limiter le travail pour chaque nouveau langage ;
- ③ Limiter le travail pour chaque nouvelle architecture.



- ① Frontend : Pour permettre le support de plusieurs langages en entrée, il faut une représentation unique et langage « agnostique » sur laquelle va pouvoir travailler l'optimiseur ;
- ② Optimizer : Faciliter la création de passes intermédiaires pour la transformation du code ;
- ③ Backend : Fournir un mécanisme commun de représentation des machines.

- ① Frontend : Pour permettre le support de plusieurs langages en entrée, il faut une représentation unique et langage « agnostique » sur laquelle va pouvoir travailler l'optimiseur ;
- ② Optimizer : Faciliter la création de passes intermédiaires pour la transformation du code ;
- ③ Backend : Fournir un mécanisme commun de représentation des machines.

- ① Frontend : Pour permettre le support de plusieurs langages en entrée, il faut une représentation unique et langage « agnostique » sur laquelle va pouvoir travailler l'optimiseur ;
- ② Optimizer : Faciliter la création de passes intermédiaires pour la transformation du code ;
- ③ Backend : Fournir un mécanisme commun de représentation des machines.

- ❶ Origine du modèle. . . période des machines Risc ;
- ❷ Représentation intermédiaire. . . une instruction simple ;
- ❸ Backend = automate qui « match » une ou peu d'instructions internes et retourne une ou peu d'instructions machines.

- 1 Origine du modèle. . . période des machines Risc ;
- 2 Représentation intermédiaire. . . une instruction simple ;
- 3 Backend = automate qui « match » une ou peu d'instructions internes et retourne une ou peu d'instructions machines.

- 1 Origine du modèle. . . période des machines Risc ;
- 2 Représentation intermédiaire. . . une instruction simple ;
- 3 Backend = automate qui « match » une ou peu d'instructions internes et retourne une ou peu d'instructions machines.

Il en résulte :

- ❶ Non prévu pour architectures hybrides (Cell, GPU) ;
- ❷ Difficulté sur les très grosses instructions, exemple :  
multiplication matrice ;
- ❸ Backend optimisé == HACK (gcc == 24 ans de hack).

Il en résulte :

- ① Non prévu pour architectures hybrides (Cell, GPU) ;
- ② Difficulté sur les très grosses instructions, exemple :  
multiplication matrice ;
- ③ Backend optimisé == HACK (gcc == 24 ans de hack).

Il en résulte :

- ① Non prévu pour architectures hybrides (Cell, GPU) ;
- ② Difficulté sur les très grosses instructions, exemple : multiplication matrice ;
- ③ Backend optimisé == HACK (gcc == 24 ans de hack).

## LLVM - Présentation du framework

Auroux Lionel

État de l'art sur les  
compilateurs

### LLVM

- Présentation
- Comment cela marche
- Exemples d'utilisation
- Optimisation hors  
compilation
- Écriture du backend
- Frontend CLANG

Questions

1 État de l'art sur les compilateurs

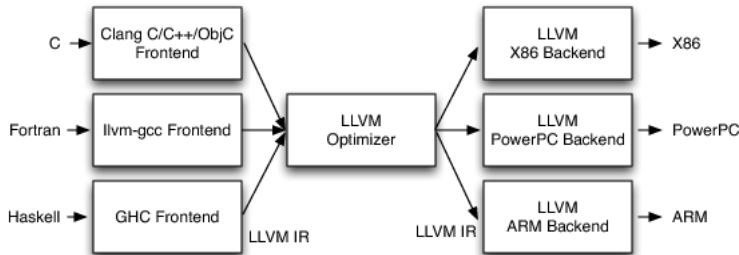
2 LLVM

3 Questions

## 2 LLVM

- **Présentation**
- Comment cela marche
- Exemples d'utilisation
- Optimisation hors compilation
- Écriture du backend
- Frontend CLANG

LLVM est un backend de compilation « retargetable »



État de l'art sur les  
compilateurs

LLVM

Présentation

Comment cela marche

Exemples d'utilisation

Optimisation hors  
compilation

Écriture du backend

Frontend CLANG

Questions

LLVM, c'est :

- ① Un certain nombre de librairie C++ (+ de 30) ;
- ② Un certain nombre d'outil CLI ( 15).

LLVM, c'est :

- ① Un certain nombre de librairie C++ (+ de 30) ;
- ② Un certain nombre d'outil CLI ( 15).

## 2 LLVM

- Présentation
- **Comment cela marche**
- Exemples d'utilisation
- Optimisation hors compilation
- Écriture du backend
- Frontend CLANG

Un langage unique pour la représentation intermédiaire des instructions (LLVM IR). Un langage sous trois formes :

- 1 un format texte (cela ressemble à de l'assembleur) ;
- 2 une représentation en mémoire manipulable via API (cela ressemble a un Ast) ;
- 3 un format fichier binaire (cela ressemble à du bytecode).

Un langage unique pour la représentation intermédiaire des instructions (LLVM IR). Un langage sous trois formes :

- 1 un format texte (cela ressemble à de l'assembleur) ;
- 2 une représentation en mémoire manipulable via API (cela ressemble a un AST) ;
- 3 un format fichier binaire (cela ressemble à du bytecode).

Un langage unique pour la représentation intermédiaire des instructions (LLVM IR). Un langage sous trois formes :

- ① un format texte (cela ressemble à de l'assembleur) ;
- ② une représentation en mémoire manipulable via API (cela ressemble a un AST) ;
- ③ un format fichier binaire (cela ressemble à du bytecode).

Format texte pour le code C suivant :

---

```
1 unsigned add1(unsigned a, unsigned b) {
2     return a+b;
3 }
4
5 unsigned add2(unsigned a, unsigned b) {
6     if (a == 0) return b;
7     return add2(a-1, b+1);
8 }
```

---

## Code LLVM

---

```
1 define i32 @add1(i32 %a, i32 %b) {
2   entry:
3     %tmp1 = add i32 %a, %b
4     ret i32 %tmp1
5 }
6
7 define i32 @add2(i32 %a, i32 %b) {
8   entry:
9     %tmp1 = icmp eq i32 %a, 0
10    br i1 %tmp1, label %done, label %recurse
11
12   recurse:
13     %tmp2 = sub i32 %a, 1
14     %tmp3 = add i32 %b, 1
15     %tmp4 = call i32 @add2(i32 %tmp2, i32 %tmp3)
16     ret i32 %tmp4
17
18   done:
19     ret i32 %b
20 }
```

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

LLVM

Présentation

Comment cela marche

Exemples d'utilisation

Optimisation hors  
compilation

Écriture du backend

Frontend CLANG

Questions

## 2 LLVM

- Présentation
- Comment cela marche
- **Exemples d'utilisation**
- Optimisation hors compilation
- Écriture du backend
- Frontend CLANG

## Initialisation des N composants LLVM

- 1 Notion de Target
- 2 Notion de Context
- 3 Notion de Module
- 4 Notion de Builder
- 5 Notion de PassManager
- 6 Notion d' ExecutionEngine

État de l'art sur les  
compilateurs

### LLVM

Présentation

Comment cela marche

**Exemples d'utilisation**

Optimisation hors  
compilation

Écriture du backend

Frontend CLANG

Questions

## Initialisation des N composants LLVM

- 1 Notion de Target
- 2 Notion de Context
- 3 Notion de Module
- 4 Notion de Builder
- 5 Notion de PassManager
- 6 Notion d' ExecutionEngine

État de l'art sur les  
compilateurs

### LLVM

Présentation

Comment cela marche

**Exemples d'utilisation**

Optimisation hors  
compilation

Écriture du backend

Frontend CLANG

Questions

## Initialisation des N composants LLVM

- 1 Notion de Target
- 2 Notion de Context
- 3 Notion de Module
- 4 Notion de Builder
- 5 Notion de PassManager
- 6 Notion d' ExecutionEngine

## Initialisation des N composants LLVM

- 1 Notion de Target
- 2 Notion de Context
- 3 Notion de Module
- 4 Notion de Builder
- 5 Notion de PassManager
- 6 Notion d' ExecutionEngine

## Initialisation des N composants LLVM

- 1 Notion de Target
- 2 Notion de Context
- 3 Notion de Module
- 4 Notion de Builder
- 5 Notion de PassManager
- 6 Notion d' ExecutionEngine

## Initialisation des N composants LLVM

- 1 Notion de Target
- 2 Notion de Context
- 3 Notion de Module
- 4 Notion de Builder
- 5 Notion de PassManager
- 6 Notion d' ExecutionEngine

# Exemple de construction

## On peut construire des nodes via l'API C++

---

```
1 // Dans une fonction... void f(int ac, int ad)
2 // (ac + 66) + (ad * 12)
3 /* Cree 2 constante */
4 Value *v1 = ConstantInt::get(getGlobalContext(),
5                               APInt(32, 66, false));
6 Value *v2 = ConstantInt::get(getGlobalContext(),
7                               APInt(32, 12, false));
8 /* Recup la value d'un parametre de la fonction
9    f */
10 Value *vac = f->getValueSymbolTable().lookup("ac"
11                                               );
12 Value *vad = f->getValueSymbolTable().lookup("ad"
13                                               );
14 /* Cree une addition entre les 2 */
15 Value *r1 = Builder.CreateAdd(vac, v1, "res1");
16 Value *r2 = Builder.CreateMul(vad, v2, "res2");
17 Value *r = Builder.CreateAdd(r1, r2, "res");
```

---

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

LLVM

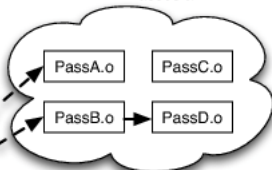
Présentation  
Comment cela marche  
Exemples d'utilisation  
Optimisation hors  
compilation  
Écriture du backend  
Frontend CLANG

Questions

# Exemple de création d'une passe

```
1 namespace {
2   class XYZ : public FunctionPass {
3   public:
4     /* Print out the names of functions in the LLVM
5       IR being optimized. */
6     virtual bool runOnFunction(Function &F) {
7       cerr << "Hello: " << F.getName() << "\n";
8       return false;
9     }
10 };
11
12 FunctionPass *createXYZPass()
13 { return new Hello(); }
```

LLVMPasses.a



XYZOptimizer.cpp

```
PassManager PM;  
PM.add(createPassA());  
PM.add(createPassB());  
PM.add(createXYZPass());  
...
```

XYZPasses.a



---

```
1  Function *f = Function::Create(ft, Function::
      ExternalLinkage, "zefunc", TheModule);
2  ....
3  ExecutionEngine *exec = EngineBuilder(TheModule).
      create();
4  MyJITListener jit_listen;
5  exec->RegisterJITEventListener(&jit_listen);
6  exec->DisableLazyCompilation();
7  void *pf = exec->recompileAndRelinkFunction(f);
8  ((PtrZeFunc)pf)(1, 2.0, "3");
```

---

- 2 LLVM
  - Présentation
  - Comment cela marche
  - Exemples d'utilisation
  - **Optimisation hors compilation**
  - Écriture du backend
  - Frontend CLANG

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

LLVM

Présentation

Comment cela marche

Exemples d'utilisation

**Optimisation hors  
compilation**

Écriture du backend

Frontend CLANG

Questions

# Link Time Optimization

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

LLVM

Présentation

Comment cela marche

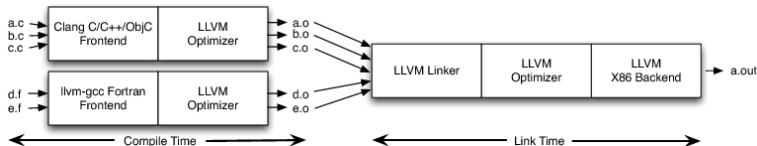
Exemples d'utilisation

Optimisation hors  
compilation

Écriture du backend

Frontend CLANG

Questions



# Install Time Optimization

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

LLVM

Présentation

Comment cela marche

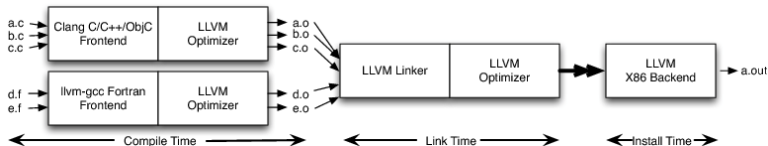
Exemples d'utilisation

Optimisation hors  
compilation

Écriture du backend

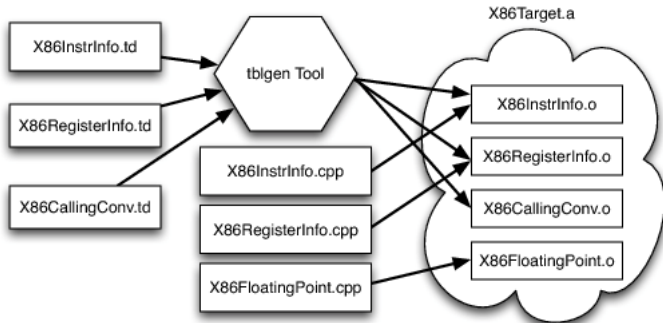
Frontend CLANG

Questions



## 2 LLVM

- Présentation
- Comment cela marche
- Exemples d'utilisation
- Optimisation hors compilation
- **Écriture du backend**
- Frontend CLANG



Exemple de description de groupe de registre :

```
def GR32 : RegisterClass<[i32], 32,  
    [EAX, ECX, EDX, ESI, EDI, EBX, EBP, ESP,  
    R8D, R9D, R10D, R11D, R14D, R15D, R12D, R13D]>  
    { ... }
```

## LLVM - Présentation du framework

Auroux Lionel

État de l'art sur les  
compilateurs

### LLVM

Présentation

Comment cela marche

Exemples d'utilisation

Optimisation hors  
compilation

Écriture du backend

**Frontend CLANG**

Questions

- 2 LLVM
  - Présentation
  - Comment cela marche
  - Exemples d'utilisation
  - Optimisation hors compilation
  - Écriture du backend
  - **Frontend CLANG**

## LLVM - Présentation du framework

Auroux Lionel

État de l'art sur les  
compilateurs

### LLVM

- Présentation
- Comment cela marche
- Exemples d'utilisation
- Optimisation hors  
compilation
- Écriture du backend
- Frontend CLANG**

Questions

- 1 Contient un parseur (C, C++, ObjC) ;
- 2 Passe de transformation CPP ;
- 3 Désucre C++, ObjC ;
- 4 Notion de SourceManager (source de données à compiler).

## LLVM - Présentation du framework

Auroux Lionel

État de l'art sur les  
compilateurs

### LLVM

- Présentation
- Comment cela marche
- Exemples d'utilisation
- Optimisation hors  
compilation
- Écriture du backend
- Frontend CLANG**

Questions

- 1 Contient un parseur (C, C++, ObjC);
- 2 Passe de transformation CPP;
- 3 Désucre C++, ObjC;
- 4 Notion de SourceManager (source de données à compiler).

## LLVM - Présentation du framework

Auroux Lionel

État de l'art sur les  
compilateurs

### LLVM

- Présentation
- Comment cela marche
- Exemples d'utilisation
- Optimisation hors  
compilation
- Écriture du backend
- Frontend CLANG

Questions

- 1 Contient un parseur (C, C++, ObjC);
- 2 Passe de transformation CPP;
- 3 Désucre C++, ObjC;
- 4 Notion de SourceManager (source de données à compiler).

- 1 Contient un parseur (C, C++, ObjC);
- 2 Passe de transformation CPP;
- 3 Désucre C++, ObjC;
- 4 Notion de SourceManager (source de données à compiler).

- ① StringC2JIT ;
- ② Nouveau Qualifier sur l'ownership de pointeur +  
passe de type.

- ① StringC2JIT ;
- ② Nouveau Qualifier sur l'ownership de pointeur +  
passe de typage.

- 1 État de l'art sur les compilateurs
- 2 LLVM
- 3 Questions**

LLVM -  
Présentation du  
framework

Auroux Lionel

État de l'art sur les  
compilateurs

LLVM

Questions