Béatrice Creusillet <bcreusillet@quarkslab.com>
Juan Manuel Martinez <jmmartinez@quarkslab.com>

Compilation pour le reverseur

Table of Contents

Table of Contents

Comment peut-on être binaire ?

ordinateur = tas de composants électroniques (image)
parle en niveaux de tension = binaire (verbatim encadré de 0100100010)
(image d'un être humain perplexe.)
=> nécessité d'un langage 'humain' => langages de programmation =>
qu'il faut 'traduire en binaire'
En première approche compilation = traduction en binaire

Compilation everywhere

Table of Contents

Un gradient d'approches

- Compilation : traduction statique
  - réutilisable
  - code unique qui s'adapte à des inputs variés
  - code adapté à un environnement particulier (windows, linux, osx,...)
  - ex: C, C++, Fortran
- Pré-compilation : traduction statique vers un 'byte-code'
  - byte-code générique
  - adaptation à l'environnement au run-time
  - ex: Java
- JIT : Just-in-Time compilation
  - traduction dynamique partiellement 'réutilisable'
  - ex: Matlab, Julia,
- Interprétation : traduction dynamique = à la volé
  - utilisation unique
  - code optimisé pour des inputs particuliers
  - ex: Basic, R

Compilateurs C/C++

Compilateurs commerciaux

- Intel C/C++ compiler (icc) (Windows, linux, MacOs, Intel)
- Visual Studio C++ (Windows, Microsoft)
- IAR C/C++ compiler (Windows, Linux, others, IAR Systems)
- Edison Design Group (Windows, Linux, others, EDG)
- VisualAge C++, XL C/C++ (Windows, LInux, Aix, OS/2, OS/400,... IBM)
- ...

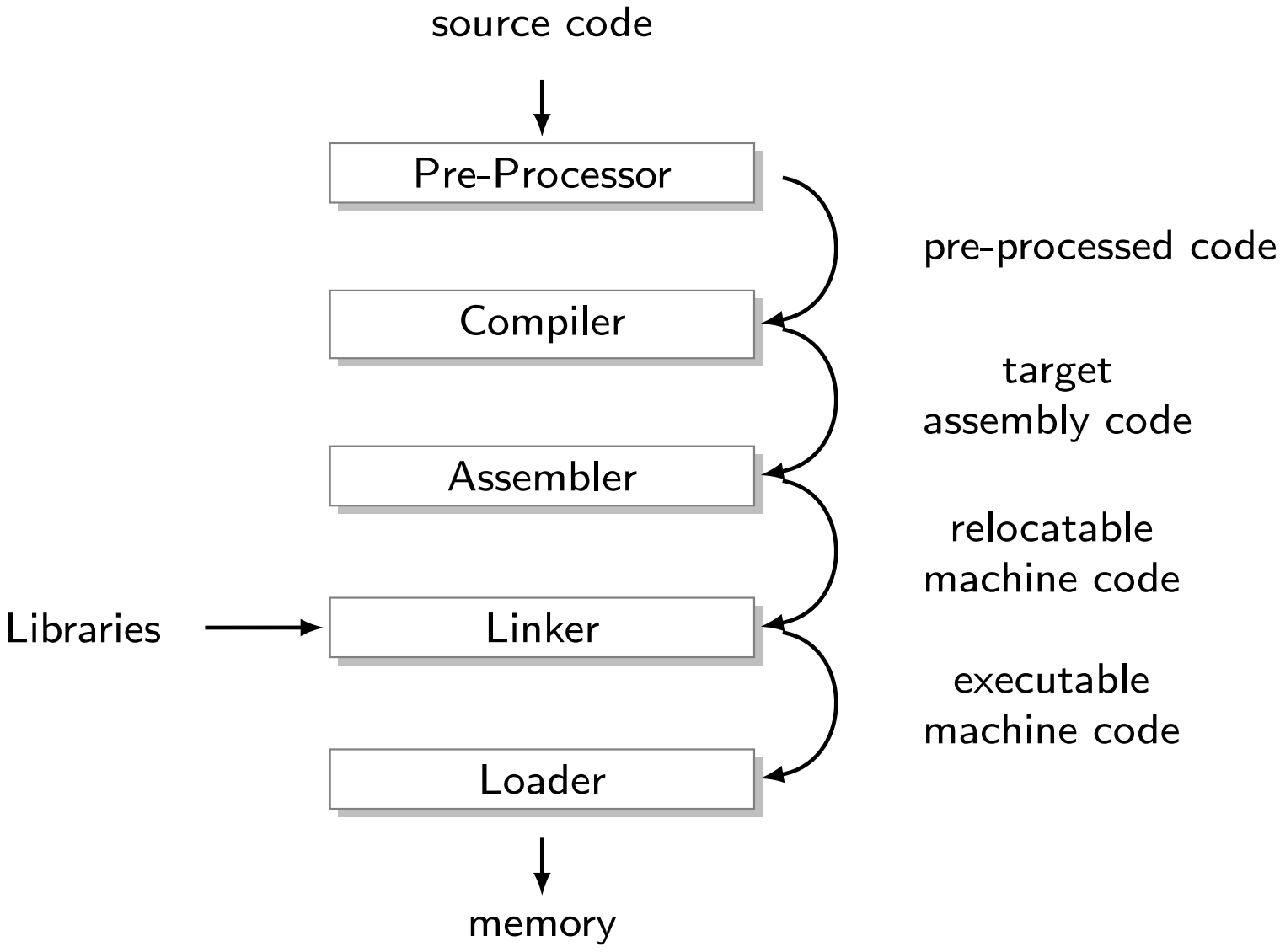
OpenSource ou Freeware

- GNU C/C++ (gcc, MinGW) (Windows, linux, MacOS, GNU Project)
- Clang/LLVM (clang) (Windows, linux, MacOS, LLVM Project)
- ...

https://en.wikipedia.org/wiki/List_of_compilers

Table of Contents

Une traduction en plusieurs étapes



Un exemple : td1

```c
#include <stdio.h>
void welcome(char const* nom) {
  printf("welcome %s\n", nom);
}
```

```c
#ifndef DEFAULT
#define DEFAULT "alice"
#endif

extern void welcome(char const*);

int main(int argc, char const* argv[]) {
  if      (argc == 1) { welcome(DEFAULT); }
  else if (argc == 2) { welcome(argv[1]); }
  else { return 1; }
  return 0;
}
```

```
  gcc welcome.c main_welcome.c
  ./a.out
  ./a.out alice
```

```
  gcc welcome.c main_welcome.c -DDEFAULT='"bob"'
  ./a.out
  ./a.out alice
```

Le pré-processeur

- Traitement des macros

```
1          gcc -E main_welcome.c
2          gcc -E main_welcome.c -DDEFAULT='"bob"'
```

- Inclusion des fichiers *header*

```
1          gcc -E welcome.c | wc -l
2          gcc -E welcome.c | head -n 10
```

Le compilateur : génération de code assembleur

```
1       gcc -S main_welcome.c
2       more main_welcome.s
```

L'assemblage: génération de code objet

```
1       as main_welcome.s -o main_welcome.o
2       file main_welcome.o
3       nm main_welcome.o
```

```
1       gcc -c welcome.c welcome.o
2       file welcome.o
3       nm welcome.o
```

L'édition de lien: génération de binaire

```
1       ld welcome.o main_welcome.o
```

```
1       ld welcome.o main_welcome.o -lc
2       file a.out
```

```
1       gcc -v welcome.o main_welcome.o
2       clang -v welcome.o main_welcome.o 2>&1 | grep ld
```

Table of Contents

Piloter son compilateur : les options

- Contrôle des inputs/outputs
  - fichiers source: paramètres positionnels
  - bibliothèques: `-l<libname>`
  - chemins d'accès
    - fichiers headers: `-I<path>`
    - bibliothèques: `-L<path>`
  - standard du language: `-std=c++11` par ex.
  - warnings: `-W<warning>` (ex: `-Wall`, `-Wall`
  - `--help`, `--version`, `-print-search-dirs`, ...
- Passer des options aux différentes étapes
  - pre-processeur: `-Wp,<option>`
  - assembleur: `-Wa,<option>`
  - linker: `-Wl,<option>`

Piloter l'optimiseur

- Liste des optimisations:
  - gcc/g++:

```
1          g++ --help=optimizers
```

  - clang/llvm:

```
1          clang -help
2          opt -help
```

- Optimisations globales: `-O<level>`
  - `-O0` : pas d'optimisations, pratique pour le debug
  - `-O3` : plus fort niveau d'optimisations
  - `-Os` : `-O2`, mais minimise la taille du code
  - `-Ofast`: `-O3` + `-ffastmath`

Piloter l'optimiseur : td2

```c
#define n 5
/* #define n 100 */

int main() {
  int a[n];
  for (int i = 0; i < n; ++i) { a[i] = i+1; }

  int s = 0;
  for (int i = 0; i < n; ++i) { s +=a [i]; }

  if ( s== (n*(n+1))/2 )       { return s; }
  return -1;
}
```

```
gcc array.c -o array-O0
./array-O0
echo $?
```

- avec `#define n 5`
  Comparer les binaires générés avec `-O0` et `-O1`
- avec `#define n 100`
  Comparer les binaires générés avec `-O0` et `-O3`

Debugging (1)

```c
#include <assert.h>

#define MAX_SIZE 512

int count_char(const char* str) {
  int c = 0;
  while ( (c < MAX_SIZE) && (str[c] != '\0')) { ++c; }
  return c;
}

int main(int argc, char** argv) {
  assert(argc == 2 && "compliant number of arguments");

  return count_char(argv[1]);
}
```

```
$ gcc debug.c -o debug
$ ./debug toto
$ echo $?
$ ./debug
```

Debugging (1)

- L'impact de `-DNDEBUG`

```
1  $ gcc debug.c -o debug-n -DNDEBUG
2  $ ./debug-n toto
3  $ echo $?
4  $ ./debug-n
```

Comparer les binaires générés avec et sans `-DNDEBUG`

- L'impact de `-g`

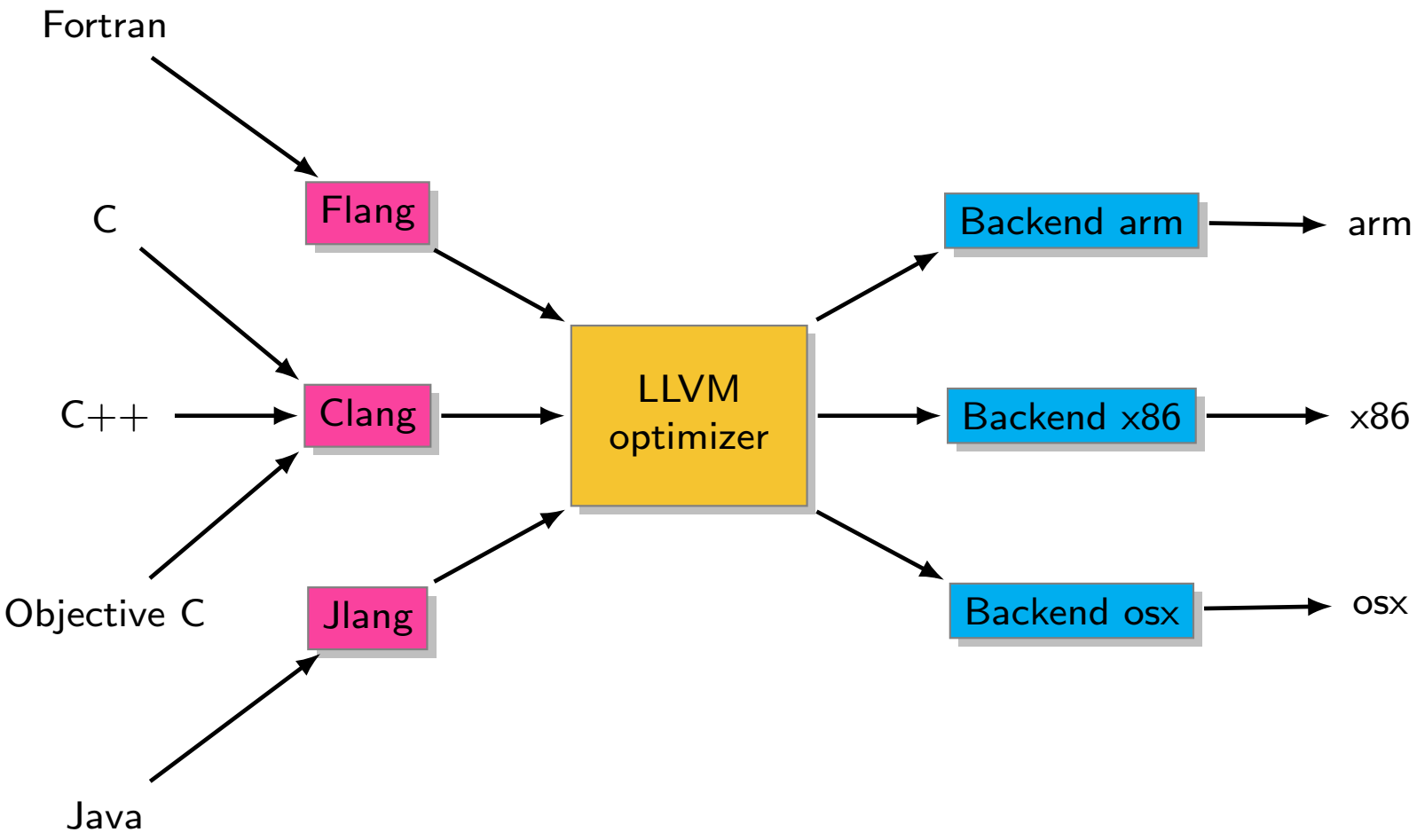
```
1  $ gcc debug.c -o debug-n -DNDEBUG -g
```

Comparer les binaires générés avec et sans `-g`

Table of Contents

Table of Contents

Clang/LLVM


Fortran
C
C++
Objective C
Java
Flang
Clang
Jlang
LLVM
optimizer
Backend arm
Backend x86
Backend osx
arm
x86
osx

LLVM : un optimiseur d'IR + un générateur de code assembleur

Table of Contents