



MI-GEN

Cviceni, majpetr@fit.cvut.cz (2017)



Projekt



- Lowering jednoduchého jazyka do LLVM bitcode
 - AST -> SSA
 - SSA v LLVM platí pouze pro registry, je možné používat proměnné na zásobníku
- Implementace optimalizací v rámci LLVM
 - Constant propagation,
 - Dead code elimination
 - Inlining



LLVM



- Low Level Virtual Machine
- Chris Lattner 2003, dnes vyvíjeno v rámci Apple, Inc.
- LLVM neobsahuje frontend pro žádný jazyk
 - CLANG je frontend pro C++ pro LLVM
- LLVM se specializuje na
 - Optimalizace nad bitcode
 - Několik jich vytvoříte v rámci projektu
 - Generování machine code
 - Tím se v rámci projektu zabývat nebudeme
- LLVM neobsahuje rozumný runtime, ten si musíme vytvořit sami
 - Ale náš runtime bude opravdu velmi primitivní



LLVM Debug vs Release

- LLVM se linkuje jako statická knihovna k
- Verze LLVM s debug symboly opravdu, ale opravdu dlouho linkuje
- Pro beznou práci není ideální, protože i na slušném počítači bude link time v řádu desítek vteřin až minut
- LLVM release verze je mnohem rychlejší, samozřejmě se v ní ale velmi, velmi těžce ladí
- Ideální je používat obě, a do debug verze přepnout jen když je opravdu třeba



CMAKE

- Crossplatform build system
- CMakeLists.txt obsahuje definici projektu (zavislosti, spustitelne soubory, knihovny, atd.)
- Cmake pak vygeneruje podle definice projektu make soubory pro dany backend a architekturu(make, ninja, nmake, ...)

Cd build
cmake .. -G "ninja"

- Zvoleny backend pak provede vlastni build
ninja

LLVM Instalace

- svn, g++, cmake, idealne ninja-build
- `svn co http://llvm.org/svn/llvm-project/llvm/tags/RELEASE_370/final/ llvm-src-370`
 - Clone svn repa LLVM, verze 3.7 se kterou budeme na cvicenich pracovat
- Debug konfigurace:
`cmake -G "Ninja" -DLLVM_OPTIMIZED_TABLEGEN=1 -
DLLVM_ENABLE_RTTI=1 -
DLLVM_TARGETS_TO_BUILD="X86;CppBackend" -
DCMAKE_BUILD_TYPE="Debug" --enable-debug-symbols --
with-oprofile ../llvm-src-370`

LLVM Instalace

- svn, g++, cmake, idealne ninja-build
- `svn co http://llvm.org/svn/llvm-project/llvm/tags/RELEASE_370/final/ llvm-src-370`
 - Clone svn repa LLVM, verze 3.7 se kterou budeme na cvicenich pracovat

- Debug konfigurace

```
cmake -G "Ninja" -DLLVM_ENABLE_R  
-DLLVM_TARGETS_TO_BUILD="x86,CppBackend" -  
-DCMAKE_BUILD_TYPE="Debug" --enable-debug-symbols --  
with-oprofile ../llvm-src-370
```

Release build nepotrebuje
debug symboly



cmake projekt

```
cmake_minimum_required(VERSION 2.8.8)

project(llvmtest)

set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -Werror)

#setup llvm
find_package(LLVM REQUIRED CONFIG)
message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION} in
${LLVM_DIR}")
add_definitions(${LLVM_DEFINITIONS})
include_directories("../llvm-src-370/include")

file(GLOB SRC "*.cpp" "*.h")
add_executable(${PROJECT_NAME} ${SRC})

llvm_map_components_to_libnames(LLVM_LIBS support core mcjit
native irreader linker ipo executionengine runtime dyld)

target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```


cmake projekt

Nazev projektu

cmake_minimum

project(llvmtest)

set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")

add_definitions(-g --std=c++11 -Wall -Werror)

#setup llvm

find_package(LLVM REQUIRED CONFIG)

message(STATUS "Found LLVM \${LLVM_PACKAGE_VERSION} in
\${LLVM_DIR}")

add_definitions(\${LLVM_DEFINITIONS})

include_directories("../llvm-src-370/include")

file(GLOB SRC "*.cpp" "*.h")

add_executable(\${PROJECT_NAME} \${SRC})

llvm_map_components_to_libnames(LLVM_LIBS support core mcjit
native irreader linker ipo executionengine runtime dyld)

target_link_libraries(\${PROJECT_NAME} \${LLVM_LIBS})

cmake projekt

Promenna v cmake,
odkaz pomoci
\${NAME}

```
cmake_minimum_required(VERSION 3.0)

project(llvmtest)

set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -Werror)

#setup llvm
find_package(LLVM REQUIRED CONFIG)
message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION} in
${LLVM_DIR}")
add_definitions(${LLVM_DEFINITIONS})
include_directories("../llvm-src-370/include")

file(GLOB SRC "*.cpp" "*.h")
add_executable(${PROJECT_NAME} ${SRC})

llvm_map_components_to_libnames(LLVM_LIBS support core mcjit
native irreader linker ipo executionengine runtime dyld)

target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```

cmake projekt

Promenna v cmake,
odkaz pomoci
\${NAME}

```
cmake_minimum_required(VERSION 3.0)
project(llvmtest)
set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -Werror)

#setup llvm
find_package(LLVM REQUIRED)
message(STATUS "Found LLVM ${LLVM_VERSION} in ${LLVM_DIR}")
add_definitions(${LLVM_DEFINITIONS})
include_directories("../llvm-src-370/include")

file(GLOB SRC "*.cpp" "*.h")
add_executable(${PROJECT_NAME} ${SRC})

llvm_map_components_to_libnames(LLVM_LIBS support core mcjit
native irreader linker ipo executionengine runtime dyld)

target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```

Tato promenna nam rika kde najdeme
LLVM

cmake projekt

```
cmake_minimum_required(VERSION 2.8.8)
project(llvmtest)
set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -Werror)
#setup llvm
find_package(LLVM REQUIRED)
message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION} in
${LLVM_DIR}")
add_definitions(${LLVM_DEFINITIONS})
include_directories("../llvm-src-370/include")

file(GLOB SRC "*.cpp" "*.h")
add_executable(${PROJECT_NAME} ${SRC})

llvm_map_components_to_libnames(LLVM_LIBS support core mcjit
native irreader linker ipo executionengine runtime dyld)

target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```

Argumenty pro prekladac,
pouzivaji GCC syntax, ale
cmake je prelozi podle
pouziteho backendu

cmake projekt

```
cmake_minimum_required(VERSION 2.8.8)
```

```
project(llvmtest)
```

```
set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
```

```
add_definitions(-g --std=c++11 -Wall -Werror)
```

```
#setup llvm
```

```
find_package(LLVM REQUIRED CONFIG)
```

```
message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION} in  
${LLVM_DIR}")
```

```
add_definitions(${LLVM_DEFINITIONS})
```

```
include_directories("../llvm-src-370/include")
```

```
file(GLOB SRC "*.cpp" "*.h")
```

```
add_executable(${PROJECT_NAME} ${SRC})
```

```
llvm_map_components_to_libnames(LLVM_LIBS support core mcjit  
native irreader linker ipo executionengine runtime dyld)
```

```
target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```

Vyhledani LLVM a
inicializace definic,
include adresaru,
atd. Cmake vetsinu
udela za nas v ramci
find_package

cmake projekt

```
cmake_minimum_required(VERSION 2.8.8)
project(llvmtest)
set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -Werror)

#setup llvm
find_package(LLVM REQUIRED COMPONENTS Core)
message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION}
${LLVM_DIR}")
add_definitions(${LLVM_DEFINITIONS})
include_directories("../llvm-src-370")

file(GLOB SRC "*.cpp" "*.h")
add_executable(${PROJECT_NAME} ${SRC})

llvm_map_components_to_libnames(LLVM_LIBS support core mcjit
native irreader linker ipo executionengine runtime dyld)

target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```

Nalezení všech zdrojových
souboru na dané cestě (.)
Vytvoření spustitelného
souboru

cmake projekt

```
cmake_minimum_required(VERSION 2.8.8)

project(llvmtest)

set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -Werror)

#setup llvm
find_package(LLVM REQUIRED CONFIG)
message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION} in
${LLVM_DIR}")
add_definitions(${LLVM_DEFINITIONS})
include_directories("../llvm-release-370/include")

file(GLOB SRC "*.cpp")
add_executable(${PROJECT_NAME} ${SRC})

llvm_map_components_to_libnames(LLVM_LIBS support core mcjit
native irreader linker ipo executionengine runtime dyld)

target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```

Definice component LLVM ktere
budeme potrebovat a jejich
namapovani na knihovny
(funkce llvm cmake)

cmake projekt

```
cmake_minimum_required(VERSION 2.8.8)
project(llvmtest)
set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -Werror)

#setup llvm
find_package(LLVM REQUIRED CONFIG)
message(STATUS "Found LLVM ${LLVM_PACKAGE_VERSION} in
${LLVM_DIR}")
add_definitions(${LLVM_DEFINITIONS})
include_directories("../llvm-src-370/include")

file(GLOB SRC "*.cpp" "*.h")
add_executable(${PROJECT_NAME} ${SRC})

llvm_map_components_to_libnames(LLVM_LIBS)
native irreader linker ipo executionengine runtimetarget

target_link_libraries(${PROJECT_NAME} ${LLVM_LIBS})
```

Prilinkovani
potrebných LLVM
knihoven



Ukazkový Projekt

- <https://bitbucket.org/mi-gen/test-1>

- test

 - CMakeLists.txt

 - main.cpp

- Build:

```
mkdir build  
cd build  
cmake .. -G "Ninja"  
ninja  
./test
```

Ukazkovy Projekt

➤ <https://bitbucket.org/mikolajdobrzynski/ninja>

➤ test

➤ **CMakeLists.txt**

➤ main.cpp

Zde je definice projektu

➤ Build:

```
mkdir build
cd build
cmake .. -G "Ninja"
ninja
./test
```

Ukazkový Projekt

➤ <https://bitbucket.org/mi-gen/test-1>

➤ test

➤ CMakeLists.txt

➤ **main.cpp**

Zdrojové soubory (tam, kde je
bude hledat project v
CMakeLists.txt)

➤ Build:

```
mkdir build  
cd build  
cmake .. -G "Ninja"  
ninja  
./test
```

Ukazkovy Projekt

➤ <https://bitbucket.org/mi-gen/test-1>

➤ test

➤ CMakeLists.txt

➤ main.cpp

Vytvoreni build adresare
(out of tree build)

➤ Build:

mkdir build

cd build

cmake .. -G "Ninja"

ninja

./test

Ukazkovy Projekt

➤ <https://bitbucket.org/mi-gen/test-1>

➤ test

➤ CMakeLists.txt

➤ main.cpp

Vytvoreni build adresare
(out of tree build)

➤ Build:

mkdir build

cd build

cmake .. -G "Ninja"

ninja

./test

Vytvoreni build souboru pro
zvoleny backend (Ninja)
.. Je odkaz na adresar s
CMakeLists.txt

Ukazkovy Projekt

➤ <https://bitbucket.org/mi-gen/test-1>

➤ test

➤ CMakeLists.txt

➤ main.cpp

Vytvoreni build adresare
(out of tree build)

➤ Build:

```
mkdir build  
cd build  
cmake .. -G "Ninja"  
ninja  
./test
```

Vytvoreni build souboru pro
zvoleny backend (Ninja)
.. Je odkaz na adresar s
CMakeLists.txt

Vlastni build (ninja, make, nmake,
atd.)



Zaver

- Za domaci ukol rozchodit LLVM
 - A to je pro dnešek vsechno
- 