MI-GEN

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

Projekt

- Lowering jednoducheho jazyka do LLVM bitcode
  - AST -> SSA
  - SSA v LLVM plati pouze pro registry, je mozne pouzivat promenne na zasobniku
- Implementace optimalizaci v ramci LLVM
  - Constant propagation,
  - Dead code elimination
  - Inlining

LLVM

- Low Level Virtual Machine
- Chris Lattner 2003, dnes vyvijeno v ramci Apple, Inc.
- LLVM neobsahuje frontend pro zadny jazyk
  - CLANG je frontend pro C++ pro LLVM
- LLVM se specializuje na
  - Optimalizace nad bitcode
    - Nekolik jich vytvorite v ramci projektu
  - Generovani machine code
    - Tim se v ramci projektu zabyvat nebudeme
- LLVM neobsahuje rozumny runtime, ten si musime vytvorit sami
  - Ale nas runtime bude opravdu velmi primitivni

LLVM Debug vs Release

- LLVM se linkuje jako staticka knihovna k
- Verze LLVM s debug symboly opravdu, ale opravdu dlouho linkuje
- Pro beznou praci neni idealni, protoze I na slusnem pocitaci bude link time v radu desitek vterin az minut
- LLVM release verze je mnohem rychlejsi, samozrejme se v ni ale velmi, velmi tezce ladi
- Idealni je pouzivat obe, a do debug verze prepnout jen kdyz je opravdu treba

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 konfigura

  cmake -G "Ninja" -
  DLLVM_ENABLE_R
  DLLVM_TARGETS_TO_BUILD
  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 runtimedyld)

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 runtimedyld)

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

cmake projekt

Promenna v cmake, odkaz pomoci ${NAME}

```
cmake_minim                                    3)

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 runtimedyld)

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

cmake projekt

Promenna v cmake, odkaz pomoci ${NAME}

```
cmake_minim                                    3)
project(llvmtest)
set(LLVM_DIR "../llvm-release-370/cmake/modules/CMakeFiles")
add_definitions(-g --std=c++11 -Wall -W    for)

#setup llvm
find_package(LLVM R
message(STATUS "Four
${LLVM_DIR}")
add_definitions(${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 runtimedyld)

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_packag
message(STA                                    KAGE_VERSION} in
${LLVM_DIR}"
add_definitio
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 runtimedyld)

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_PACKA
${LLVM_DIR}")
add_definitions(${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 runtimedyld)

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 CO
message(STATUS "Found LLVM ${LL
${LLVM_DIR}")
add_definitions(${LLVM_DEFINITION
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 runtimedyld)

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

Nalezeni vsech zdrojovych souboru na dane ceste (.)
Vytvoreni spustitelneho 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
${LLVM_DIR}")
add_definitions(${LLVM
include_directories("../l

file(GLOB SRC "*.cpp" "
add_executable(${PRO

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

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_LI
native irreader linker ipo executionengine run

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

Prilinkovani potrebnych LLVM knihoven

Ukazkovy 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/
- test
  - **CMakeLists.txt**
  - main.cpp

Zde je definice projektu

- Build:

```
mkdir build
cd build
cmake .. –G “Ninja”
ninja
./test
```

Ukazkovy Projekt

- https://bitbucket.org/mi-gen/test-1
- test
  - CMakeLists.t
  - **main.cpp**

Zdrojove 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
- Build:

mkdir build
cd build
cmake .. –G “Ninja”
ninja
./test

Vytvoreni build adresare (out of tree build)

Ukazkovy Projekt

- https://bitbucket.org/mi-gen/test-1
- test
  - CMakeLists.txt
  - main.cpp
- Build:

```
mkdir build
cd build
cmake .. –G “Ninja”
ninja
./test
```

Vytvoreni build adresare (out of tree build)

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
- Build:

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

Vytvoreni build adresare (out of tree build)

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 dnesek vsechno