Analizy, transformacje C++ za pomocą clanga

Piotr Padlewski

Przygotowanie prezentacji

Czym jest clang

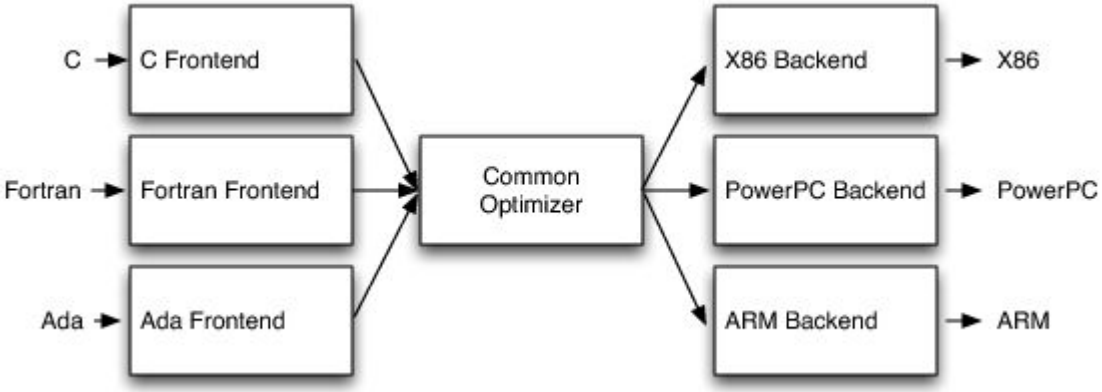
Clang (czytaj “K-lang” - ‘klæŋ’) to zestaw kompilatorów (tak na prawdę front-endów) języków C, C++, Objective-C. Clang został zbudowany na szczycie LLVMa.

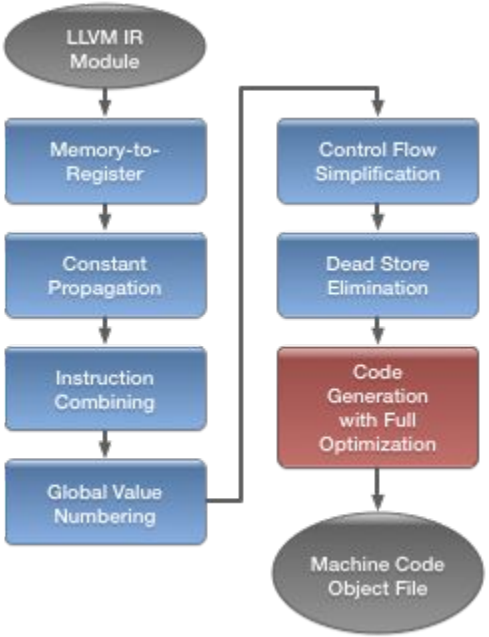
Czym jest LLVM

LLVM - Low Level Virtual Machine - architektura do kompilacji wielu różnych języków, tak zwany middle-end.

LLVM IR (Intermediate representation) to kod niskiego poziomu użyty do reprezentacji wielu języków programowania.

Modularność clanga





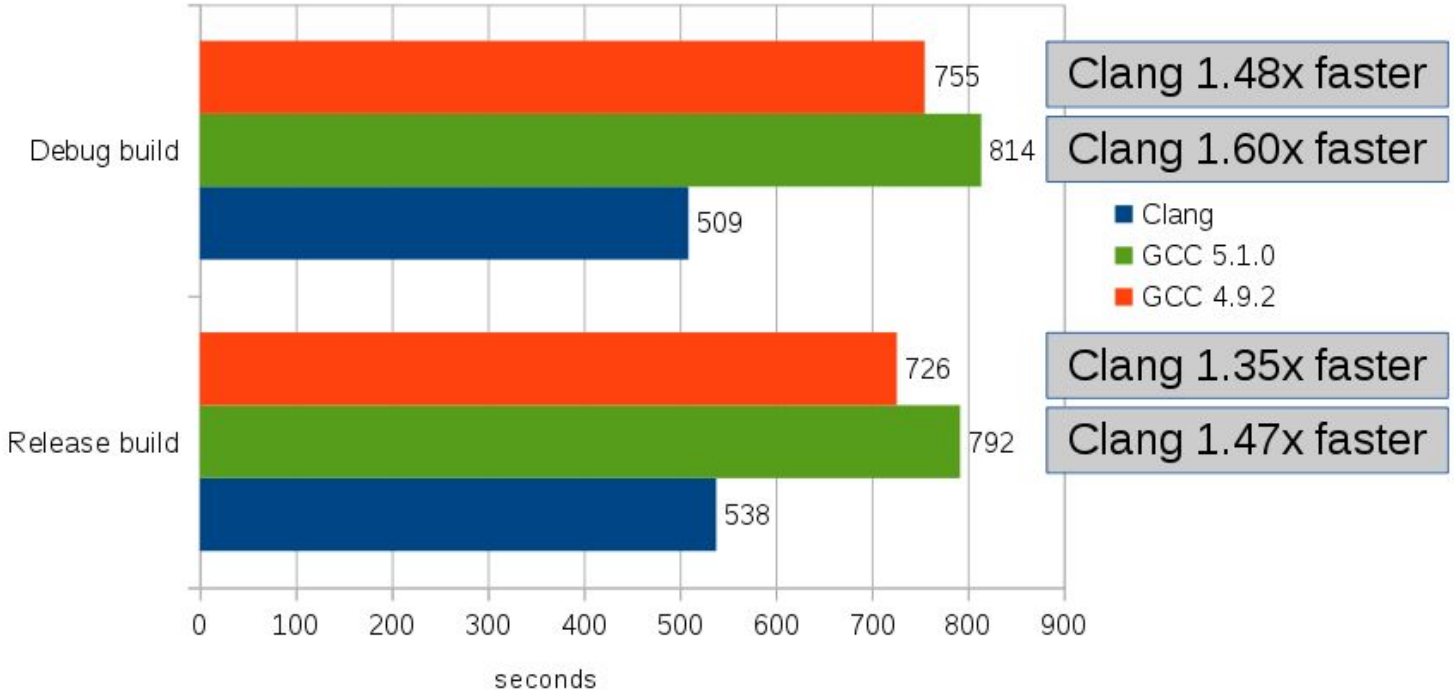
Inne cechy clanga i LLVMa

Jedną z głównych cech clanga jak i LLVM jest to że są to dobrze zaprojektowane i napisane projekty.
Główny powód jest niewielki wiek projektów (2003 LLVM, 2005 Clang).
Clang jak i LLVM jest napisany w C++11.

Clang vs gcc

- Szybsza kompilacja
- Lepsze komunikaty błędów
- Więcej użytecznych warningów
- Lepsze wsparcie dla sanitizerów

Clang vs gcc



Z moich doświadczeń:

17min vs 34min
clang-3.8 gcc-4.8

Clang vs gcc - Lepsze komunikaty błędów

```
struct A {};
void foo(A *a);
int main() {
    A abcdeef;
    fooa(abcdeef);
}
```

```
ppadlewski@SSD1:~ $ clang++ test.cc -std=c++11
test.cc:7:5: error: use of undeclared identifier 'fooa'; did you mean 'foo'?
    fooa(abcdeef);
    ^~~~
    foo
test.cc:2:6: note: 'foo' declared here
void foo(A *a);
     ^
test.cc:7:10: error: no viable conversion from 'A' to 'A *'
    fooa(abcdeef);
         ^~~~~~~
test.cc:2:13: note: passing argument to parameter 'a' here
void foo(A *a);
            ^
2 errors generated.
ppadlewski@SSD1:~ $
```

```
ppadlewski@SSD1:~ $ g++ -std=c++11 test.cc -Wall
test.cc: In function 'int main()':
test.cc:7:17: error: 'fooa' was not declared in this scope
     fooa(abcdeef);
                 ^
ppadlewski@SSD1:~ $
```

Clang vs gcc - Lepsze komunikaty błędów

```
CalcECPrim.h:33:19: warning: private field 'auditPeriodBeginTS_' is not used [-Wunused-private-field]
    const int32_t auditPeriodBeginTS_;
NodeIDCounterWriter.h:25:10: warning: 'commit' overrides a member function but is not marked 'override' [-
Winconsistent-missing-override]
    void commit();
         ^
StatsWriter.h:43:10: note: overridden virtual function is here
    void commit()
         ^
OfflinePIDRestore.cc:34:26: warning: binding reference member 'twister_' to stack allocated parameter 'twister' [-
Wdangling-field]
                twister_(twister)
                         ^~~~~~~
OfflinePIDRestore.cc:78:19: note: reference member declared here
    TwisterType & twister_;
```

gcc vs Clang

- better performance ~ 3-6%
- gcc-5.1 powoli dogania

Z wnętrza clanga

Jak jest testowany clang?

Każdego dnia:

- skompiluj wszechświat i zobacz czy działa
- .. na 7 różnych sposobów
- i nie zapomnij skompilować sam siebie

Co ciekawe nie używane są testy jednostkowe.

but Clang is not only a compiler

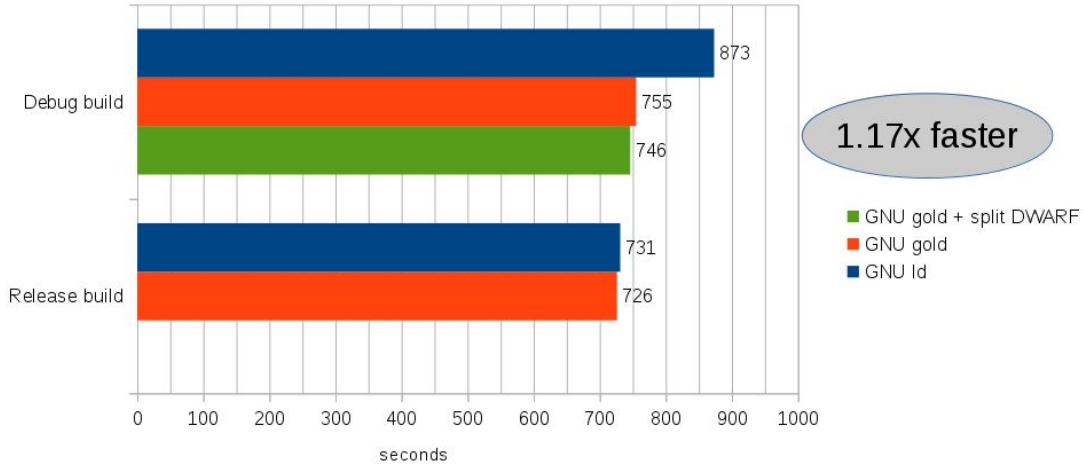
Clang to tak na prawdę biblioteki, na podstawie których zostało zbudowanych wiele narzędzi takich jak:

- clang-format (LibFormat)
- clang-tidy (LibTooling)

i wiele innych...

Tip1: What is GOLD?

Gold to szybszy linker.



Przy rekompilacji clanga różnica między bfd a gold była rzędu 50%.
(4 min vs 2 min)

/usr/bin/ld -> /usr/bin/ld.gold

clang-format

clang-format to narzędzie do formatowania kodu C++ które:

- (jako jedyne) … działa
- wiele możliwości konfiguracji
- kilka różnych styli
- integracja z vim i emacs

po co fomatować kod narzędziem?

- Zmarnowany czas podczas review
- narzędzie vs style guide
- refaktoryzacja, ale o tym później

opcje clang-format

- ~ 65 konfigurowalnych opcji (sic!)
- sortowanie includów
- automatyczne wykrywanie niektórych opcji
- niektóre opcje są na prawdę zaskakujące

clang-format minusy

- Niektóre przydatne rzeczy nie są wspierane:
  - ustawianie nazw typedefów w jednej lini
  - operator ## w makrach
  - formatowanie argumentów makr
- pragmatyzm comunity clang-format
  - ciężko jest wrzucić coś na upstream
- Użytkownicy (niekonsekwencja stylu)

clang-format inne plusy

- nie wymaga bycia kompatybilnym z clangiem
- można wyłączać formatowanie na części kodu

git-clang-format i clang-format-diff

Istnieją także narzędzia do formatowania tylko kodu dotkniętego przez jakąś zmianę:

- clang-format-diff
- gitowa nakładka git-clang-format

Tip2: What is ccache?

Ccache to narzędzie do cachowania kompilacji.

Kiedy używać?

Jeśli zmieniamy branche lub czyścimy builda.

scan_build

“nakładka” na make. Kompiluje + odpala wiele analiz statycznych. Czasami występują false-positivy.

Dużym plusem jest generowanie raportu html.

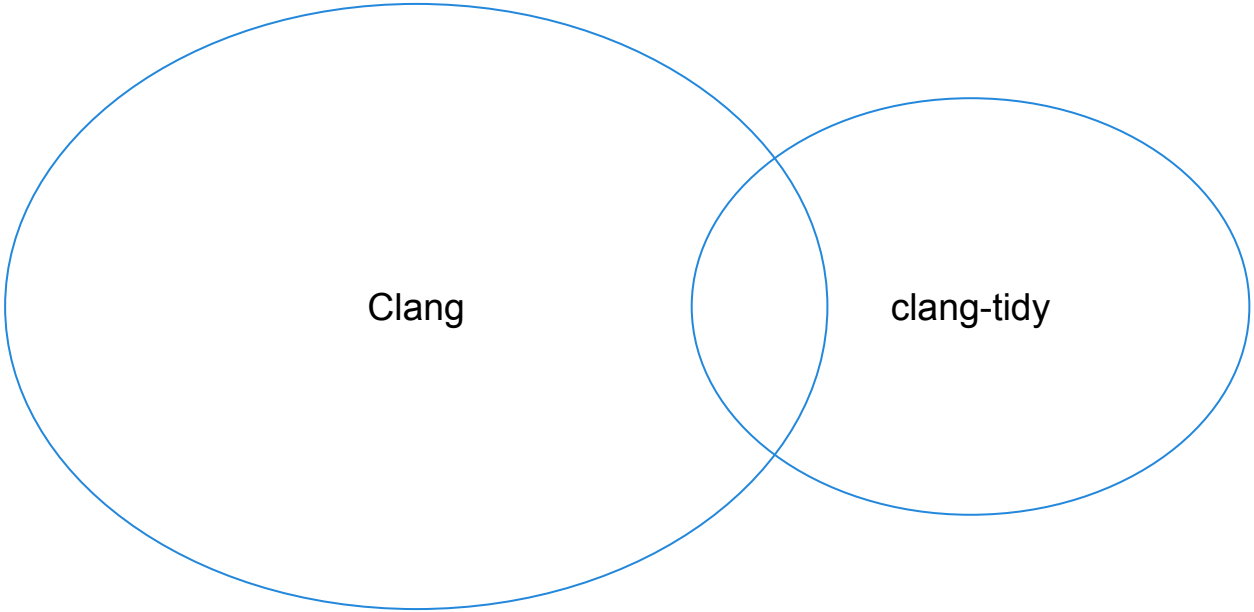
clang-tidy

clang-tidy to coś w rodzaju lintera opartego na clangu. Jednak spośród innych linterów dostępnych na rynku wyróżnia go:

- banalnie proste pisanie nowych checków
- transformowanie kodu

clang-tidy-3.8 posiada ~80 checków

po co osobny tool?



- Nie zwiększać czasu kompilacji
- wrzucać checki które nie wszystkim pasują

clang-tidy (clang-modernize)

- przerabianie pętli
- dopisywanie override
- dopisywanie default
- przerabianie konstruktorów
- dodawanie auto - da się zrobić znacznie więcej
- NULL|0 -> nullptr
- auto_ptr -> unique_ptr
- modernize-make-unique
- modernize-shrink-to-fit

clang-tidy: misc-*

- misc-inaccurate-erase
- misc-inefficient-algorithm
- misc-swapped-arguments
- misc-uniqueptr-reset-release
- misc-static-assert: assert -> static_assert
- misc-noexcept-move-constructor
- misc-move-const-arg

clang-tidy: readability-*

- container-size-empty
- else-after-return
- identifier-naming
- inconsistent-declaration-parameter-name
- redundant-smartptr-get
- redundant-string-cstr
- simplify-boolean-expr

clang-tidy: llvm-*

- llvm-header-guard
- llvm-include-order
- llvm-namespace-comment

clang-tidy: google-*

- google-global-names-in-headers
- google-runtime-member-string-references
- google-explicit-constructor
- google-build-using-namespace

cert-* i cppcoreguidelines-*

Grupa mniej przydatnych checków o losowych nazwach (w przypadku cert) i małej użyteczności.

clang-analyzer-*

Bardzo dużo must-have checków. Większość z nich działa także przy normalnym odpaleniu clanga.

Niestety słaba dokumentacja.

Po nazwach można się domyślić że duża częsć z nich stara się znajdować bugi.

minusy clang-tidy

- za dużo checków (just kidding)
- problemy z formatowaniem (np else return)
- gryzące się ze sobą checki
- multiple run checki
- nakładanie kilka razy tej samej zmiany (.h)
- trochę słaba dokumentacja

plusy clang-tidy

- oszczędność czasu
- pilnowanie programistów
- łatwość pisaina checków
  - clang-query
- polityka dodawania nowych checków

clang-tidy: koncert życzeń

- use-after-move (radykalny lub nie)
- redundant-type (lexical_cast itd)
- niechciana konwersja typów
- typedef -> auto
- nieefektywne używanie seta

clang-tidy: rezultaty

Dla projektu mającego ok 80 KLOC zamienił 1100 lini, co uważam za bardzo dobry resultat.

run-clang-tidy

Istnieje także prosta nakładka na clang-tidy, która uruchamia clang-tidy w wielu procesach dla wielu plików.

you complete me - IDE w konsoli

YouCompleteMe to plugin do vima, który zmienia prosty (...) edytor tekstu w uber IDE.

Oczywiście tak jak wszystko co dobre, YCM został zbudowany przy użyciu clanga.

Tip3: What is ninja?

Ninja to build system.

- Jego wielkim plusem jest cachowanie.
- Jest ortogonalny z ccache.
- Wsparcie w cmake

clang-rename

Małe narzędzie do mini refaktoryzacji kodu poprzez zmianę nazw.

clang-check

Małe frontendowe narzędzie do:

- analizy syntaktycznej
- poprawiania błędów kompilacji
- dumpowania ast

Tip4: Moduły

Wszyscy ekscytują się modułami w c++17.

Mało osób jednak wie że duża cześć modułów jest zaimplementowana w clangu.

Wystarczy użyć flagi -fmodules i liczyć na to że nasz kod jest odpowiednio modułowy.

Jeśli nie jest to użyj modularize.

Źródełka

http://blogs.s-osg.org/an-introduction-to-accelerating-your-build-with-clang/

http://www.aosabook.org/en/llvm.html