Яндекс

Как помочь и как помешать компилятору

Андрей Олейников,
разработчик, Беспилотные автомобили Яндекса
andreyol@yandex-team.ru

Введение

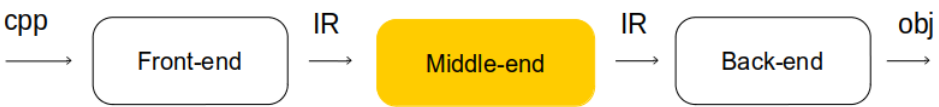
- Компиляторные оптимизации
- Возможность на них повлиять

Предупреждения

- Clang/LLVM (6.0)
- x64 (i7-8750H CPU)
- Нестандартные расширения
- Синтетические примеры

https://github.com/duke-gh/CompilerHintsExamples

Clang/LLVM



План доклада

- Inline
- Loop unrolling
- Instruction combining
- Branching
- LTO

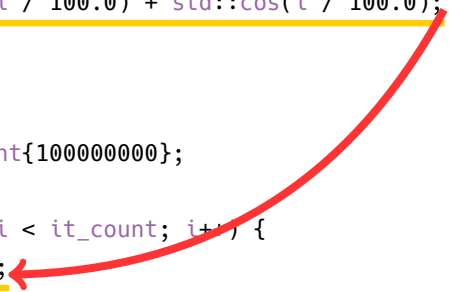
Inline

Inline

```
double calc(int i) {
  return std::sin(i / 100.0) + std::cos(i / 100.0);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += calc(i);
  }
  return res;
}
```

Inline

```
double calc(int i) {
  return std::sin(i / 100.0) + std::cos(i / 100.0);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += calc(i);
  }
  return res;
}
```



Псевдокод после оптимизации

```
double calc(int i) {
  return std::sin(i / 100.0) + std::cos(i / 100.0);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += std::sin(i / 100.0) + std::cos(i / 100.0);
  }
  return res;
}
```

Inline в Clang

- Passes: AlwaysInlinerPass, InlinerPass
- inline
- __attribute__((always_inline))
- __attribute__((noinline))

Inline пример 1

```
double calc_value(int i, int branch) {
  switch (i) {...}
...
  switch (i) {...}
  double di = static_cast<double>(i);
  if (branch) {
    return i + di * di + di * di * di + di * di * di * 0.2 + i % 2;
  } else {
    return i * 0.1 + di * di / 2.0 + di * di * di * 0.3
        + di * di * di * 0.4 + i / 2.0;
  }
}
```

Inline пример 1

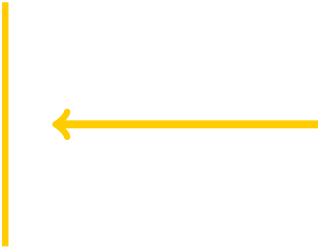
```
double calc_value(int i, int branch) {
  switch (i) {...}
...
  switch (i) {...}
  double di = static_cast<double>(i);
  if (branch) {
    return i + di * di + di * di * di + di * di * di * 0.2 + i % 2;
  } else {
    return i * 0.1 + di * di / 2.0 + di * di * di * 0.3
        + di * di * di * 0.4 + i / 2.0;
  }
}
```

Inline пример 1

```
double calc_value(int i, int branch) {
  switch (i) {...}
...
  switch (i) {...}
  double di = static_cast<double>(i);
  if (branch) {
    return i + di * di + di * di * di + di * di * di * 0.2 + i % 2;
  } else {
    return i * 0.1 + di * di / 2.0 + di * di * di * 0.3
        + di * di * di * 0.4 + i / 2.0;
  }
}
```

```
double get_first_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_second_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_value(i, 0) + get_second_value(i, 0);
  }
  return res;
}
```

```
double get_first_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_second_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_value(i, 0) + get_second_value(i, 0);
  }
  return res;
}
```



```
double get_first_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_second_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_value(i, 0) + get_second_value(i, 0);
  }
  return res;
}
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:87:12: remark: _Z15get_first_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
           ^
calc.cpp:87:37: remark: _Z16get_second_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:87:12: remark: _Z15get_first_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
           ^
calc.cpp:87:37: remark: _Z16get_second_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:87:12: remark: _Z15get_first_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
           ^
calc.cpp:87:37: remark: _Z16get_second_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:87:12: remark: _Z15get_first_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
           ^
calc.cpp:87:37: remark: _Z16get_second_valueii inlined into
      _Z7get_resv with cost=0 (threshold=337) -[Rpass=inline]
    res += get_first_value(i, 0)  + get_second_value(i, 0);
```

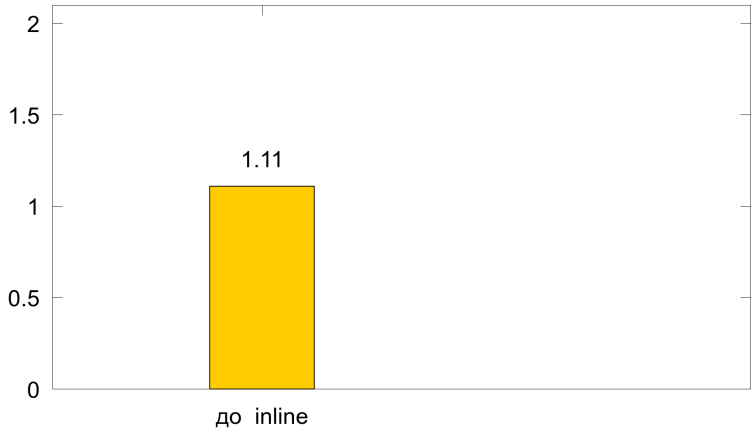
```
$ clang++ -O2 -Rpass=inline -Rpass-missed=inline -c calc.cpp
calc.cpp:76:10: remark: _Z10calc_valueii not inlined into
    _Z15get_first_valueii because too costly to inline
    (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);
         ^
calc.cpp:80:10: remark: _Z10calc_valueii not inlined into
     _Z16get_second_valueii because too costly to inline
     (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);

...
```

```
$ clang++ -O2 -Rpass=inline -Rpass-missed=inline -c calc.cpp
calc.cpp:76:10: remark: _Z10calc_valueii not inlined into
    _Z15get_first_valueii because too costly to inline
    (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);
         ^
calc.cpp:80:10: remark: _Z10calc_valueii not inlined into
     _Z16get_second_valueii because too costly to inline
     (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);

...
```

```
$ clang++ -O2 -Rpass=inline -Rpass-missed=inline -c calc.cpp
calc.cpp:76:10: remark: _Z10calc_valueii not inlined into
    _Z15get_first_valueii because too costly to inline
    (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);
         ^
calc.cpp:80:10: remark: _Z10calc_valueii not inlined into
     _Z16get_second_valueii because too costly to inline
     (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);

...
```

```
$ clang++ -O2 -Rpass=inline -Rpass-missed=inline -c calc.cpp
calc.cpp:76:10: remark: _Z10calc_valueii not inlined into
    _Z15get_first_valueii because too costly to inline
    (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);
         ^
calc.cpp:80:10: remark: _Z10calc_valueii not inlined into
     _Z16get_second_valueii because too costly to inline
     (cost=320, threshold=225) -[Rpass-missed=inline]
  return calc_value(i, branch);

...
```

Время работы get_res, сек:

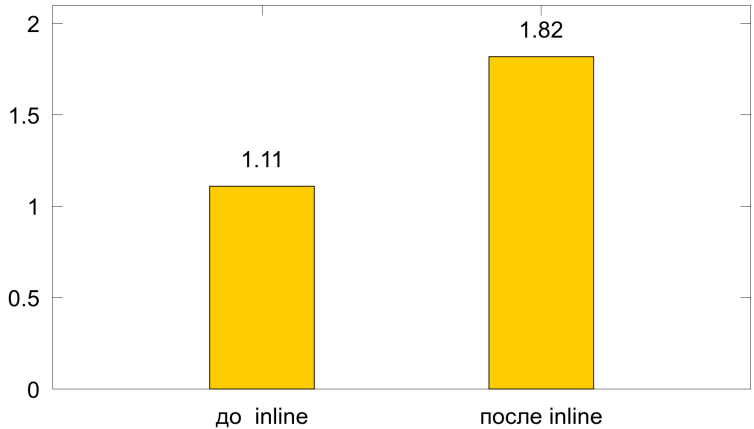


```
inline double calc_value(int i, int branch) {
...
}
```

```
// llvm/lib/Analysis/InlineCost.cpp:916
    if (Callee.hasFnAttribute(Attribute::InlineHint))
      Threshold = MaxIfValid(Threshold, Params.HintThreshold);
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:76:10: remark: _Z10calc_valueii inlined into
    _Z15get_first_valueii with cost=320 (threshold=325)
    -[Rpass=inline]
  return calc_value(i, branch);
         ^
calc.cpp:80:10: remark: _Z10calc_valueii inlined into
    _Z16get_second_valueii with cost=320 (threshold=325)
    -[Rpass=inline]
  return calc_value(i, branch);
         ^
```

Время работы get_res, сек:



```
double get_first_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_second_value(int i, int branch) {
  return calc_value(i, branch);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_value(i, 0) + get_second_value(i, 0);
  }
  return res;
}
```

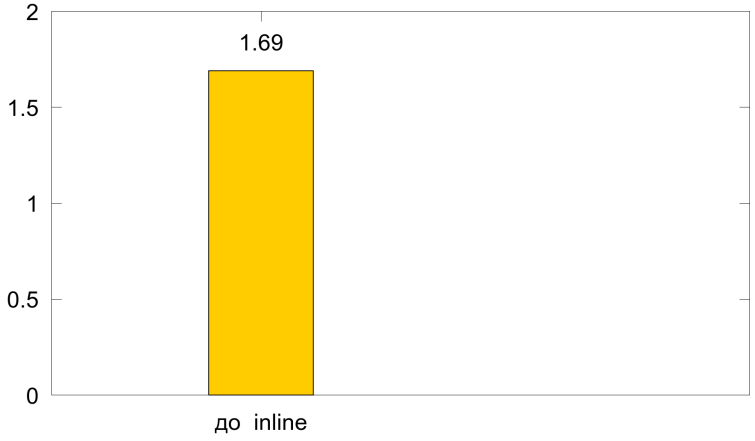
Inline пример 2

```
double calc_value(int i, int branch) {
  switch (i) {...}
...
  switch (i) {...}
  double di = static_cast<double>(i);
  if (branch) {
    return i + di * di + di * di * di + di * di * di * 0.2 + i % 2;
  } else {
    return i * 0.1 + di * di / 2.0 + di * di * di * 0.3 + di * di * di * 0.4 + i
     / 2.0;
  }
}
```

```
double get_first_value(int i) {
  return calc_value(i, 0);
}

double get_second_value(int i) {
  return calc_value(i, 1);
}

double get_res() {
  const int it_count{100000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_value(i) + get_second_value(i);
  }
  return res;
}
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:89:12: remark: _Z15get_first_valuei inlined into
      _Z7get_resv with cost=5 (threshold=337) -[Rpass=inline]
    res += get_first_value(i) + get_second_value(i);
           ^
calc.cpp:89:33: remark: _Z16get_second_valuei inlined into
      _Z7get_resv with cost=5 (threshold=337) -[Rpass=inline]
    res += get_first_value(i) + get_second_value(i);
```

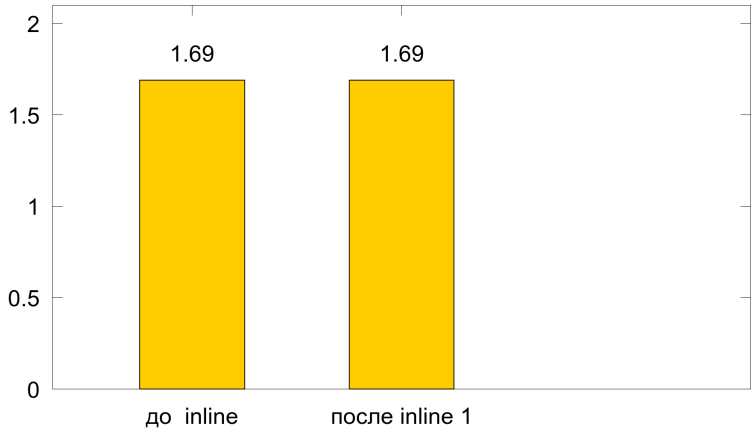
Время работы get_res, сек:



```
inline double calc_value(int i, int branch) {
...
}
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:78:10: remark: _Z10calc_valueii inlined into
     _Z15get_first_valuei with cost=265 (threshold=325) -[Rpass=
    inline]
  return calc_value(i, 0);
         ^
calc.cpp:82:10: remark: _Z10calc_valueii inlined into
     _Z16get_second_valuei with cost=255 (threshold=325) -[Rpass=
    inline]
  return calc_value(i, 1);
         ^
```

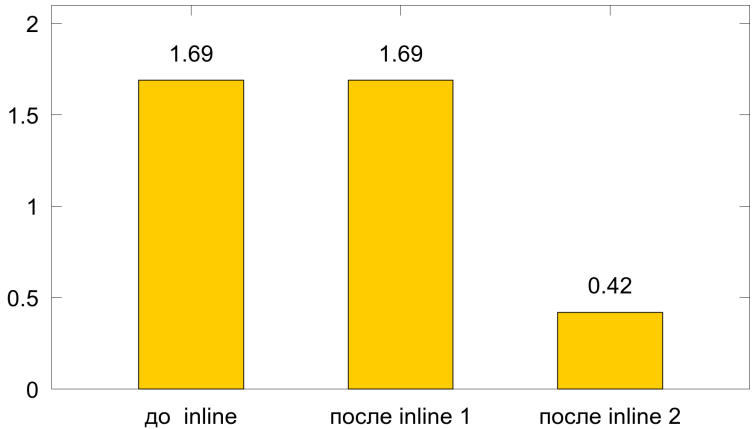
Время работы get_res, сек:



```
inline double get_first_value(int i) { ... }

inline double get_second_value(int i) { ... }
```

```
$ clang++ -O2 -Rpass=inline -c calc.cpp
calc.cpp:78:10: remark: _Z10calc_valueii inlined into
    _Z15get_first_valuei with cost=265 (threshold=325)
  return calc_value(i, 0);
         ^
...
calc.cpp:89:12: remark: _Z15get_first_valuei inlined into
      _Z7get_resv with cost=270 (threshold=325) -[Rpass=inline]
    res += get_first_value(i) + get_second_value(i);
           ^
...
```

Время работы get_res, сек:



```
double calc_value(int i, int branch) {
  switch (i) {...}
...
  switch (i) {...}
  double di = static_cast<double>(i);
  if (branch) {
    return i + di * di + di * di * di + di * di * di * 0.2 + i % 2;
  } else {
    return i * 0.1 + di * di / 2.0 + di * di * di * 0.3 + di * di * di * 0.4 + i
     / 2.0;
  }
}
```

```
__attribute__((always_inline)) double get_first_value(int i) { ... }

__attribute__((noinline)) double get_second_value(int i) { ... }
```

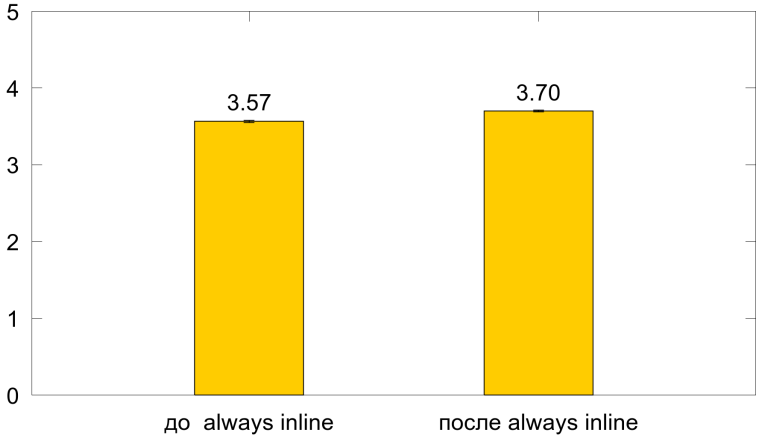
```
$ clang++ -O2 -Rpass=inline -Rpass-missed=inline -c calc.cpp
...
calc.cpp:89:12: remark: _Z15get_first_valuei inlined into
      _Z7get_resv with cost=always -[Rpass=inline]
    res += get_first_value(i) + get_second_value(i);
           ^
...
calc.cpp:89:33: remark: _Z16get_second_valuei not inlined into
      _Z7get_resv because it should never be inlined
      (cost=never) -[Rpass-missed=inline]
    res += get_first_value(i) + get_second_value(i);
```

```
double get_res() {
  const int it_count{1000000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_light_value(i) + get_second_light_value(i + 1);
    if (i < 10) {
      res += get_first_heavy_value(i, res);
      res += get_second_heavy_value(i, res);
    }
    res += get_second_light_value(i) + get_first_light_value(i - 1);
  }
  return res;
}
```

```
double get_res() {
  const int it_count{1000000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_light_value(i) + get_second_light_value(i + 1);
    if (i < 10) {
      res += get_first_heavy_value(i, res);
      res += get_second_heavy_value(i, res);
    }
    res += get_second_light_value(i) + get_first_light_value(i - 1);
  }
  return res;
}
```

```
double get_res() {
  const int it_count{1000000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += get_first_light_value(i) + get_second_light_value(i + 1);
    if (i < 10) {
      res += get_first_heavy_value(i, res);
      res += get_second_heavy_value(i, res);
    }
    res += get_second_light_value(i) + get_first_light_value(i - 1);
  }
  return res;
}
```

Время работы get_res, сек:



Итого, inline

- Позволяет делать подсказки только у вызываемой функции.
- Можно облегчить компиятору работу указав явно как поступать.
- Потенциально руками можно обогнать эвристики.

Loop unroll

Loop unroll

```
double calc(int i) {
  return std::sin(i);
}

double get_res(int it_count) {
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += calc(i);
  }
  return res;
}
```

```
// before
for (int i = 0; i < it_count; i++) {
    res += calc(i);
}

// after
int i = 0;
for (; it_count - i > 3; i += 4) {
    res += calc(i);
    res += calc(i + 1);
    res += calc(i + 2);
    res += calc(i + 3);
}
for (; i < it_count; i++) {
    res += calc(i);
}
```

Разворачивание циклов в Clang

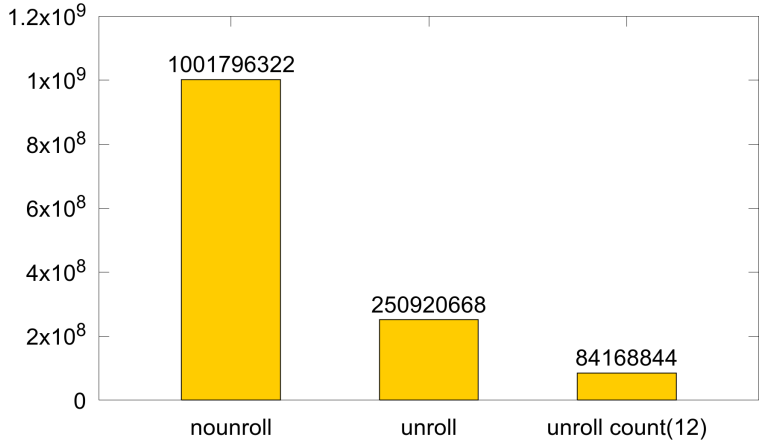
- Pass: LoopUnrollPass
- #pragma unroll
- #pragma nounroll
- #pragma clang loop unroll(full)
- #pragma clang loop unroll_count(12)

```
// llvm/lib/Transforms/Scalar/LoopUnrollPass.cpp:960
  if (HasUnrollDisablePragma(L))
    return LoopUnrollResult::Unmodified;
...

// llvm/lib/Transforms/Scalar/LoopUnrollPass.cpp:747
  if (ExplicitUnroll && TripCount != 0) {
    // If the loop has an unrolling pragma, we want to be more aggressive with
    // unrolling limits. Set thresholds to at least the PragmaThreshold value
    // which is larger than the default limits.
    UP.Threshold = std::max<unsigned>(UP.Threshold, PragmaUnrollThreshold);
    UP.PartialThreshold =
        std::max<unsigned>(UP.PartialThreshold, PragmaUnrollThreshold);
  }
```

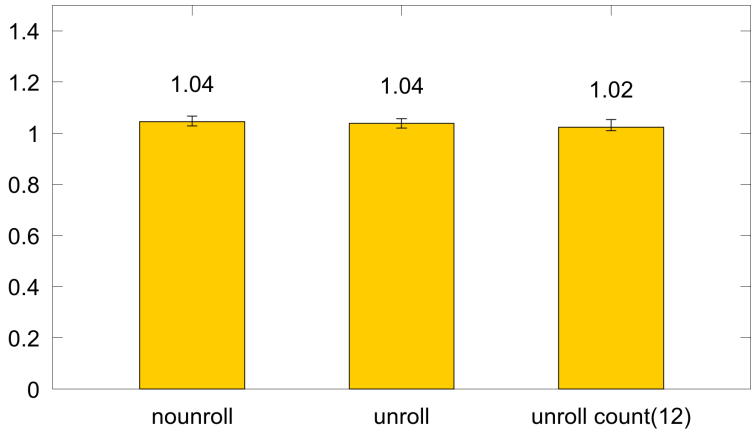
```
double calc(int i) {
  return std::sin(i);
}

double get_res_unroll_12(int it_count) {
  double res{0.0};
#pragma clang loop unroll_count(12)
  for (int i = 0; i < it_count; i++) {
    res += calc(i);
  }
  return res;
}
```

```cpp
double calc(int i) {
  return std::sin(i);
}

double get_res_unroll_12(int it_count) {
  double res{0.0};
#pragma clang loop unroll_count(12)
  for (int i = 0; i < it_count; i++) {
    res += calc(i);
  }
  return res;
}
```

Количество условных переходов:



Время работы get_res, сек:



Итого, разворачивание циклов

- Можно делать подсказки у конкретного цикла.
- Не является обязующей директивой.
- Потенциально руками можно обогнать эвристики.

Instruction combining

Instruction combining

```
int get_res(int a, int b, int c) {
  return a * b + a * c + a + a + a + c - a;
}
```

.

Instruction combining: результаты

Было:
$ab + ac + a + a + a + c - a$
Стало:
$(b + 2 + c)a + c$

Instruction combining в Clang

- Passes: InstCombinePass, ReassociatePass
- -ffast-math
  - -fno-honor-infinities
  - -fno-honor-nans
  - -fno-math-errno
  - -ffinite-math
  - -fassociative-math
  - -freciprocal-math
  - -fno-signed-zeros
  - -fno-trapping-math
  - -ffp-contract=fast

```
float get_res(float a, float b, float c, float d, float e) {
  return a * d + b * e + c * e;
}

int main() {
  float a = 500000000.0f;
  float b = -500000000.0f;
  float c = 1.0f;
  std::cout << get_res(a, b, c, 1.0f, 1.0f) << std::endl;
  return 0;
}
```

```
$ clang++ -O2 main.cpp
$ ./a.out
1


$ clang++ -ffast-math -O2 main.cpp
$ ./a.out
0
```

```
$ clang++ -O2 main.cpp
$ ./a.out
1
```

```
$ clang++ -ffast-math -O2 main.cpp
$ ./a.out
0
```

```
$ clang++ -O2 main.cpp
$ ./a.out
1


$ clang++ -ffast-math -O2 main.cpp
$ ./a.out
0
```

```
// a = 500000000.0f;
// b = -500000000.0f;
// c, d, e = 1.0f;

float get_res(float a, float b, float c, float d, float e) {
  return a * d + b * e + c * e;
}

float get_res_opt(float a, float b, float c, float d, float e) {
  return a * d + (b + c) * e;
}
```

Итого, упрощение выражений

- Ключ для сборки - влияет на всю единицу трансляции.
- Говорит компилятору быть менее консервативным.

Branching

Branching

```
void function(bool should_fail) {
  if (should_fail) {
    ...
    // Some long, rarely used error handling
    ...
  } else {
    ...
    // Main execution path
    ...
  }
}
```

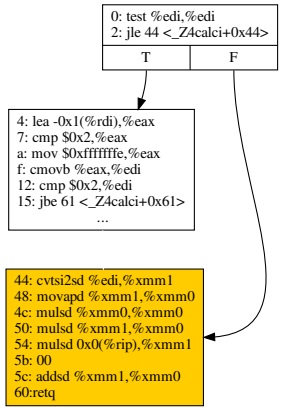
Branching в Clang

- __builtin_expect(long exp, long c)
- [[likely]] [[unlikely]] (с C++ 20)

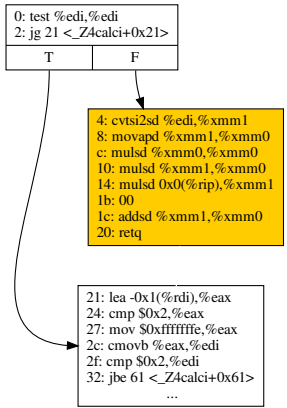
Branching пример

```
double calc(int i) {
  if (__builtin_expect(i>0, 0)) {
    switch (i) { ... }
    switch (i) { ... }
    double di = static_cast<double>(i);
    return di * di + di * 0.1;
  } else {
    double di = static_cast<double>(i);
    return di * di * di + di * 0.3;
  }
}
```

Ассемблер без подсказок



Ассемблер с подсказками

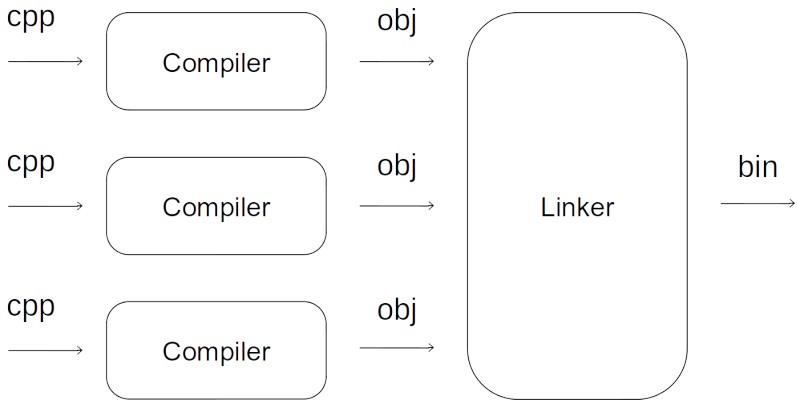


Итого, подсказки при ветвлении

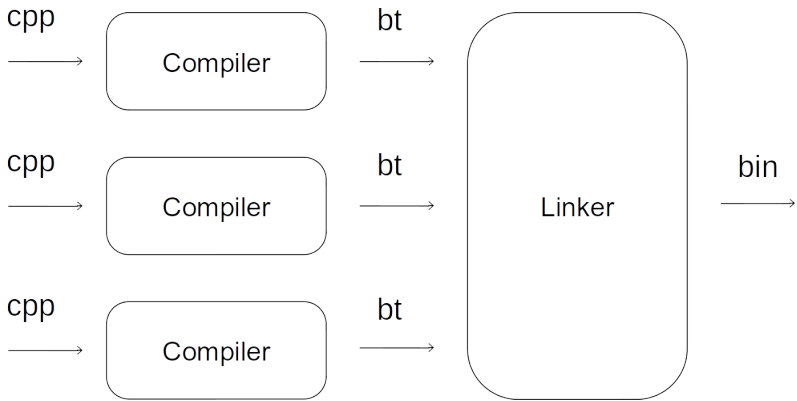
- Можно делать подсказки у каждого отдельного условного оператора.
- Компилятор не знает на каких данных будет запускаться программа.

Link-time optimization

Сборка


cpp
Compiler
obj
cpp
Compiler
obj
Linker
bin
cpp
Compiler
obj

Сборка с LTO



```cpp
// main.cpp
#include <iostream>
#include <chrono>
#include "source2.h"

int main() {
  double res{0.0};
  const auto start = std::chrono::high_resolution_clock::now().
    time_since_epoch().count();
  res = get_res();
  const auto end = std::chrono::high_resolution_clock::now().
    time_since_epoch().count();
  std::cout << "res = " << res << std::endl;
  ...
  return 0;
}
```

```cpp
// main.cpp
#include <iostream>
#include <chrono>
#include "source2.h"

int main() {
  double res{0.0};
  const auto start = std::chrono::high_resolution_clock::now().
    time_since_epoch().count();
  res = get_res();
  const auto end = std::chrono::high_resolution_clock::now().
    time_since_epoch().count();
  std::cout << "res = " << res << std::endl;
  ...
  return 0;
}
```

```cpp
// source2.cpp
#include "source1.h"

double get_res() {
  const int it_count{1000000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += calc_value(i, 0);
  }
  return res;
}
```

```cpp
// source2.cpp
#include "source1.h"

double get_res() {
  const int it_count{1000000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += calc_value(i, 0);
  }
  return res;
}
```

```cpp
// source1.cpp
int calc_value(int i, int branch) {
  int original_i = i;
  switch (i) { ... }
  switch (i) { ... }
  ...
  if (branch) {
    return i;
  } else {
    return original_i;
  }
}
```

```
$ clang++ –O2 main.cpp source1.cpp source2.cpp
```

```
$ clang++ –O2 –flto –c main.cpp source1.cpp source2.cpp
$ clang++ –O2 –flto main.o source1.o source2.o

$ llvm–dis–6.0 source1.o –o source1.ll
```

```
$ clang++ -O2 -flto -c main.cpp source1.cpp source2.cpp
$ clang++ -O2 -flto main.o source1.o source2.o

$ llvm-dis-6.0 source1.o -o source1.ll
```

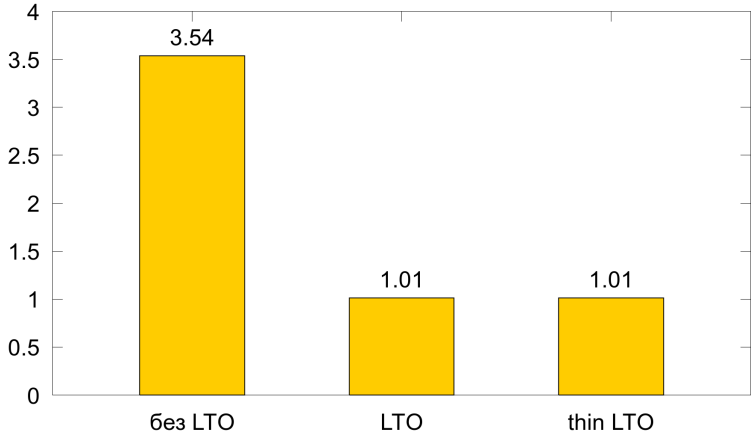
```
$ clang++ -O2 -flto -c main.cpp source1.cpp source2.cpp
$ clang++ -O2 -flto main.o source1.o source2.o

$ llvm-dis-6.0 source1.o -o source1.ll
```

```
$ clang++ -O2 -flto -c main.cpp source1.cpp source2.cpp
$ clang++ -O2 -flto main.o source1.o source2.o

$ llvm-dis-6.0 source1.o -o source1.ll
```

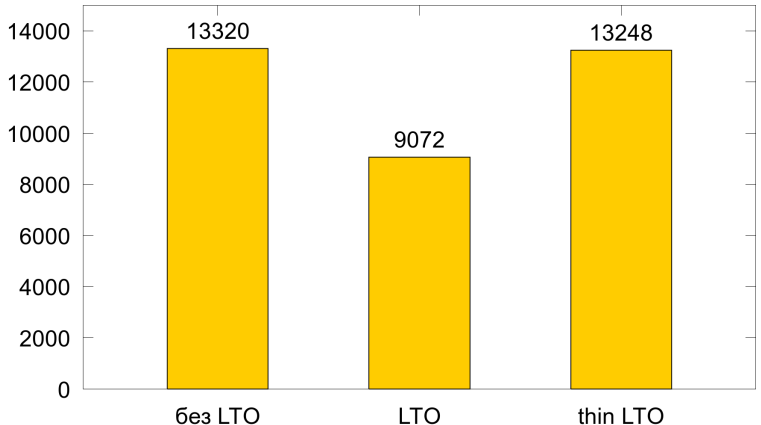
```
$ clang++ -O2 -flto=thin -c main.cpp source1.cpp source2.cpp
$ clang++ -O2 -flto=thin main.o source1.o source2.o
```

```
$ clang++ -O2 -flto=thin -c main.cpp source1.cpp source2.cpp
$ clang++ -O2 -flto=thin main.o source1.o source2.o
```

Время выполнения get_res, сек:



Размер бинарника, байт:



```cpp
// source2.cpp
#include "source1.h"

double get_res() {
  const int it_count{1000000000};
  double res{0.0};
  for (int i = 0; i < it_count; i++) {
    res += calc_value(i, 0);
  }
  return res;
}
```

```
// source1.cpp
int calc_value(int i, int branch) {
  int original_i = i;
  switch (i) { ... }
  switch (i) { ... }
  ...
  if (branch) {
    return i;
  } else {
    return original_i;
  }
}
```

Итого, оптимизации на этапе линковки

- Требуют изменения процесса сборки проекта.
- Ограничивают возможные выбор линкера.
- Увеличивают время сборки (особенно инкрементальной).
- Дают компилятору больше контекста для оптимизаций.

Заключение

- LTO, если не жалко скорость сборки.
- Подсказки компилятору, если есть время и желание проверять результат.
- Стоит изучать свои инструменты

Спасибо за внимание!

Андрей Олейников
 andreyol@yandex-team.ru