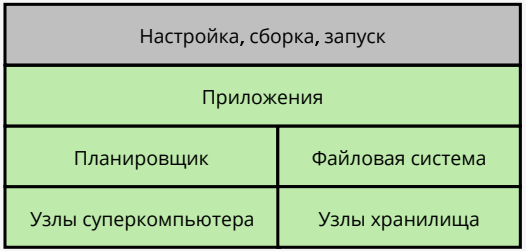
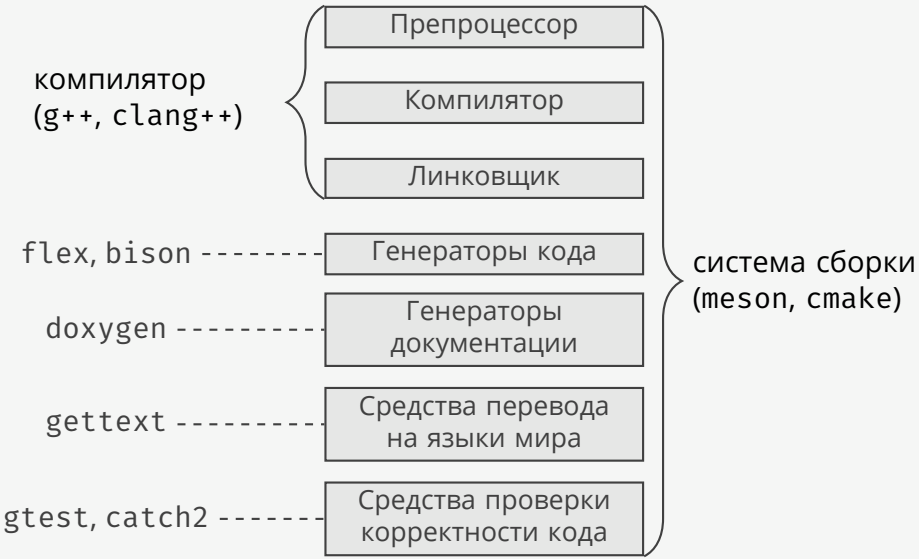
Инструменты разработчика

2022

План



- Системы сборки.
- Отладка.
- Измерение времени работы.
- Контейнеры приложений.


компилятор
(g++, clang++)
Препроцессор
Компилятор
Линковщик
flex, bison
Генераторы кода
doxygen
Генераторы документации
gettext
Средства перевода на языки мира
gtest, catch2
Средства проверки корректности кода
система сборки
(meson, cmake)

Этапы сборки

Единица компиляции в C++ — файл.

```
$ ls
group.cc main.cc user.cc
$ g++ -c group.cc -o group.o          # препроцессор + компиляция
$ g++ -c user.cc -o user.o            # препроцессор + компиляция
$ g++ -c main.cc -o main.o            # препроцессор + компиляция
$ g++ main.o group.o user.o -o myprog # линковка
```

Флаги сборки

```
$ g++ -O3 ...                  # максимальная оптимизация
$ g++ -O3 -march=native ...    # оптимизация под текущую платформу
$ g++ -fsanitize=address ...   # проверка ошибок работы с памятью
$ g++ -flto ...                # оптимизация во время линковки
$ g++ -g ...                   # отладочные символы
```

Сколько всего флагов у компилятора g++?

```
$ man g++ | col -b | grep -Eo '\\s+\\-[a-zA-Z0-9\\-]+' |
sed 's/\\s*//g' | sort -u | wc -l
2580
```

Какой прирост производительности дают различные флаги?

```
using namespace std::chrono;
auto t0 = high_resolution_clock::now();
size_t n = 1<<23; // 8 млн.
std::valarray<double> x(2.0, n), y(3.0, n);
std::valarray<double> z = x + y;
auto t1 = high_resolution_clock::now();
std::cout << duration_cast<milliseconds>(t1-t0).count() << "мс\n";
```

-O0	340мс
-O3	150мс
-O3 -march=native	145мс

Как передать флаги компиляции в систему сборки?

```
$ export CXX=clang++              # во все последующие процессы
$ env CXX=clang++ meson . build  # только в следующий процесс
```

`CXX`	компилятор C++
`CXXFLAGS`	флаги компилятора C++
`CPLUS_INCLUDE_PATH`	путь к заголовочным файлам C++
`CC`	компилятор C
`CFLAGS`	флаги компилятора C
`C_INCLUDE_PATH`	путь к заголовочным файлам C
`LDFLAGS`	флаги линковщика
`LD_LIBRARY_PATH`	путь к библиотекам
`PKG_CONFIG_PATH`	путь к файлам `pkg-config`

Зависимости

OpenCL.pc:

```
Name: OpenCL
Description: Open Computing Language Client Driver Loader
Version: 2.2
Libs: -L/usr/lib64 -lOpenCL
Cflags: -I/usr/include
```

В терминале:

```
$ pkg-config --cflags 'OpenCL >= 1.2'

$ pkg-config --libs 'OpenCL >= 1.2'
-lOpenCL
$ g++ $(pkg-config --cflags 'OpenCL >= 1.2') main.cc \
      $(pkg-config --libs 'OpenCL >= 1.2') -o myprog
```

meson.build:

```
OpenCL = dependency('OpenCL', version: '>=1.2')
```

Оптимизации под платформу

```
// версия 1
uint16_t byte_swap(uint16_t n) {
    return ((n & 0xff00)>>8) | ((n & 0x00ff)<<8);
}
// версия 2
uint16_t byte_swap(uint16_t n) {
    return __builtin_bswap16(n);
}
// версия 3
uint16_t byte_swap(uint16_t n) {
    #if defined(HAVE_BSWAP16)    // как определить HAVE_BSWAP16?
    return __builtin_bswap16(n);
    #else
    return ((n & 0xff00)>>8) | ((n & 0x00ff)<<8);
    #endif
}
```

Отладка

Сборка с отладочными символами:

```
g++ -g -O0 main.cc -o main.o                # ок: без оптимизации
g++ -g -O3 main.cc -o main.o                # неточные данные
g++ -g -O3 -march=native main.cc -o main.o # неточные данные
```

Запуск программы в режиме отладки:

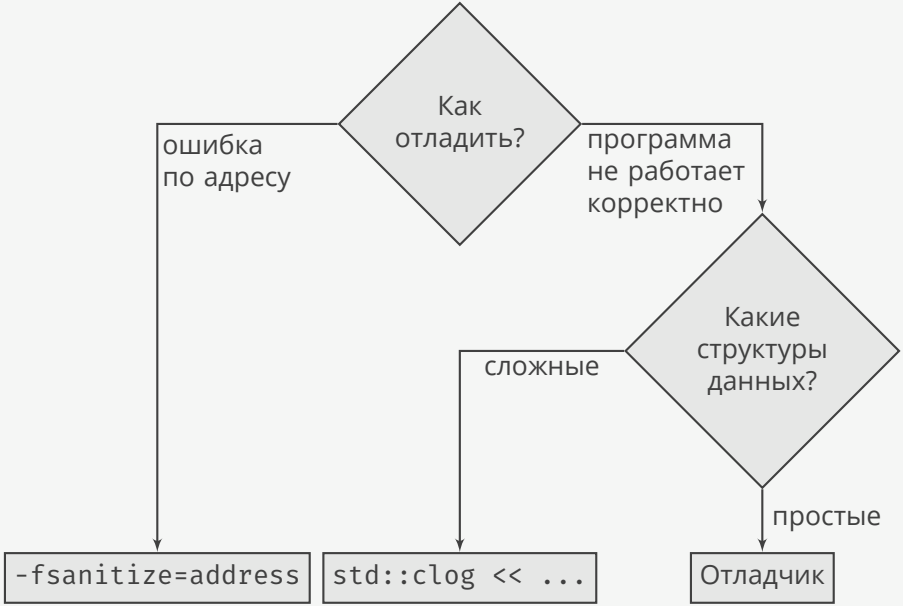
```
$ gdb ./myprog
Reading symbols from ./myprog...done.
(gdb) run
[Inferior 1 (process 3737) exited normally]
$ gdb -p НОМЕРПРОЦЕССА    # отладить уже запущенный процесс
```

Задание точки останова:

```
$ gdb ./myprog
Reading symbols from ./myprog...done.
(gdb) break main.cc:11
Breakpoint 1 at 0x400b2e: file ../main.cc, line 11.
(gdb) run
Breakpoint 1, main () at ../main.cc:11
11              std::string name;
(gdb) print name
$1 = ""
(gdb) backtrace
#0 main () at ../main.cc:11
```

Многопоточная программа:

```
$ gdb -p 26000
...
(gdb) info threads
  Id   Target Id             Frame
* 1    LWP 26140 "telegram-deskto" 0x00007f252234b559 in poll ()
...
  8    LWP 26162 "MTP::internal::" 0x00007f252234b559 in poll ()
(gdb) thread 8
[Switching to thread 8 (LWP 26162)]
#0 0x00007f252234b559 in poll () from /lib64/libc.so.6
(gdb) bt
#0 0x00007f252234b559 in poll () at /lib64/libc.so.6
#1 0x00007f2522e32b06 in g_main_context_iterate.isra
#2 0x00007f2522e32c30 in g_main_context_iteration
#3 0x000000000227a50f in ()
#4 0x0000000000000000 in ()
```


Как
отладить?
ошибка
по адресу
программа
не работает
корректно
Какие
структуры
данных?
сложные
простые
-fsanitize=address
std::clog << ...
Отладчик

Оптимизация с помощью профилирования

```
# сборка с профилингом
$ g++ -fprofile-generate main.cc -o main.o
$ g++ -fprofile-generate main.o -o myprog
# запуск тестов
...
# использование информации после профилинга
$ g++ -fprofile-use main.cc -o main.o
$ g++ -fprofile-use main.o -o myprog
```

Тоже самое в Meson build:

```
$ meson configure -Db_pgo=generate
$ meson configure -Db_pgo=use
```

Порядок веток кода

Исходный код на C++:

```
int a = 10, b = 10, d;
if (a > b) {
    d = 30;
} else {
    d = 10;
}
```

Сгенерированный код на ассемблере:

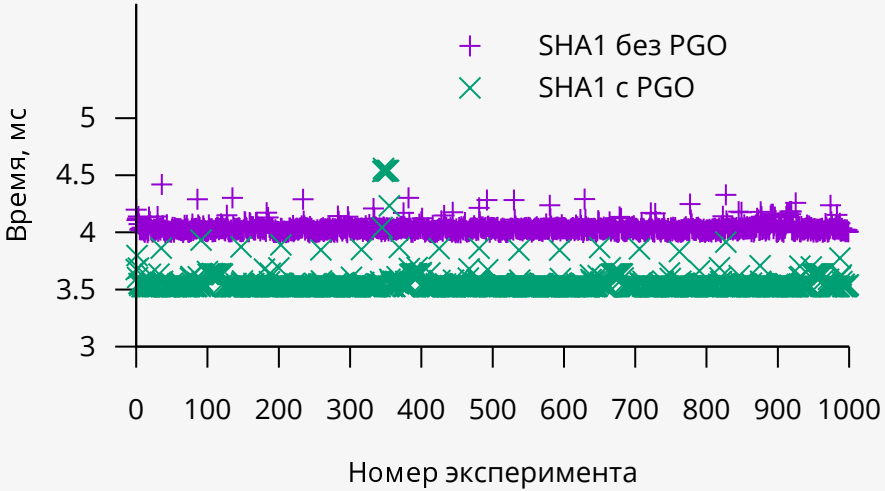
```
    movl    $10, -4(%rbp)
    movl    $10, -8(%rbp)
    movl    -4(%rbp), %eax
    cmpl    -8(%rbp), %eax
    jle .L2
    movl    $30, -12(%rbp)
    jmp .L3
.L2:
    movl    $10, -12(%rbp)
.L3:
    movl    $0, %eax
```

Порядок веток кода

```
void add_new_user(User user) {
    if (!(user.id() >= min_user_id)) {
        throw std::invalid_argument("bad uid"); // редкая ветка
    }
    if (!(user.group_id() >= min_group_id)) {
        throw std::invalid_argument("bad gid"); // редкая ветка
    }
    if (!user.has_valid_name()) {
        throw std::invalid_argument("bad name"); // редкая ветка
    }
    ...
}
```

Порядок веток кода

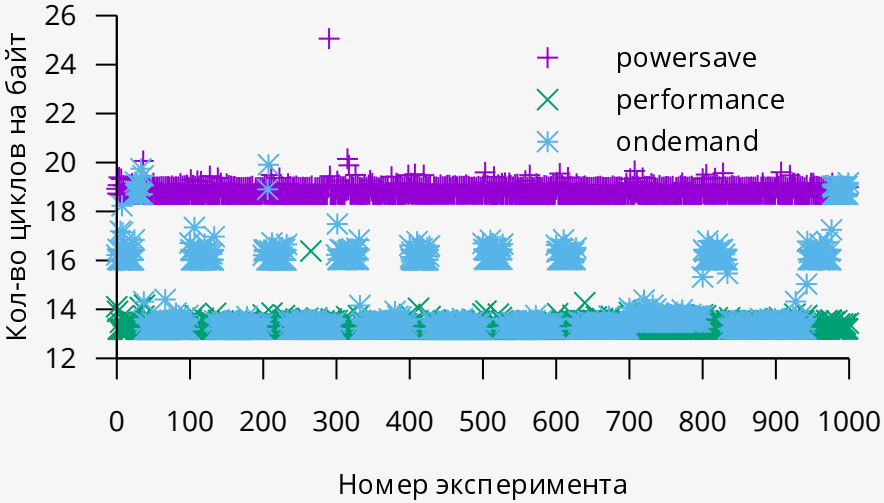
```
#define LIKELY(x) __builtin_expect((x),1)
#define UNLIKELY(x) __builtin_expect((x),0)
void add_new_user(User user) {
    if (UNLIKELY(!(user.id() >= min_user_id))) {
        ...
    }
    if (UNLIKELY(!(user.group_id() >= min_group_id))) {
        ...
    }
    if (UNLIKELY(!user.has_valid_name())) {
        ...
    }
    ...
}
```


SHA1 без PGO
SHA1 с PGO
Время, мс
5
4.5
4
3.5
3
0
100
200
300
400
500
600
700
800
900
1000
Номер эксперимента

Циклы процессора

Для быстрой функции:

```
inline uint64_t cycles() {        // TSC — Time Stamp Counter
    uint32_t high, low;
    asm volatile("lfence\n"       // барьер
                 "rdtsc"          // количество циклов
                 :
                 "=d"(high),      // считать из регистра edx
                 "=a"(low)        // считать из регистра eax
    );
    return ((uint64_t)high << 32) | low;
}
```




```
$ sudo cpupower frequency-set -g ondemand # режим работы процессора
```

Календарное время

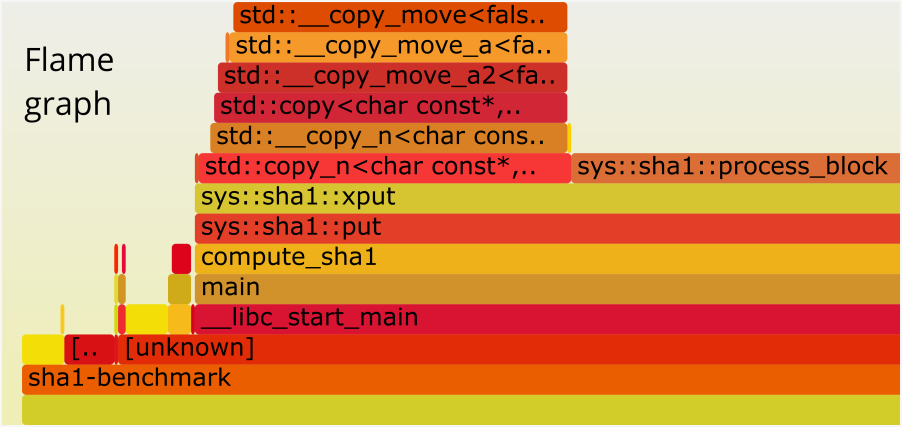
Измерение календарного (реального) времени:

```
using namespace std::chrono;
auto t0 = high_resolution_clock::now();
...
auto t1 = high_resolution_clock::now();
std::cout << duration_cast<milliseconds>(t1-t0).count() << "мс\n";
```

Распределение времени

```
Samples: 620  of event 'cycles:upp', Event count (approx.): 795163312
  Children      Self  Command          Shared Object          Symbol
+   96,57%     0,00%  sha1-benchmark   [unknown]              [.] 0x41fd89415541f689
+   96,57%     0,00%  sha1-benchmark   libc-2.27.so           [.] __libc_start_main
+   96,57%     0,00%  sha1-benchmark   sha1-benchmark         [.] main
+   96,57%     0,00%  sha1-benchmark   sha1-benchmark         [.] compute_sha1
+   96,57%     0,00%  sha1-benchmark   sha1-benchmark         [.] sys::sha1::put
+   96,40%     0,35%  sha1-benchmark   libunistdx.so.0.4.13   [.] sys::sha1::xput
+   74,85%    73,37%  sha1-benchmark   libunistdx.so.0.4.13   [.] sys::sha1::process_block
+   20,86%     0,47%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::copy_n<char const*,
+   20,04%     0,17%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::copy<char const*, ur
+   19,86%     0,00%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_n<char const*
+   19,69%     0,17%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_move_a2<false
+   19,51%    19,51%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_move<false, 1
+   19,51%     0,00%  sha1-benchmark   libunistdx.so.0.4.13   [.] std::__copy_move_a<false,
+    1,13%     0,52%  sha1-benchmark   libunistdx.so.0.4.13   [.] sys::to_host_format<unsig
```

```
$ perf record -F 199 -g ./myprog    # 199 Гц + стек вызовов
$ perf report                       # интерактивная таблица
```




```
$ perf record -F 199 -g ./myprog
$ cp ~/github/FlameGraph/*.pl .   # копируем скрипты
$ perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > g.svg
```

еще графики

Инструмент	Накладные расходы	Портируемость	Автом.	Ед. измерения
`rdtsc`	низкие	нет	нет	циклы
`std::chrono`	средние	да	нет	микросекунды
`perf`	высокие	да	да	проценты

Окружение

- Singularity
- Docker
- Vagrant

Рецепт Singularity

```
Bootstrap: yum
OSVersion: 28
MirrorURL: https://...
Include: dnf

%post
dnf install -y gcc-c++ meson gtest-devel git
git clone https://... .
meson . build
cd build
ninja install
dnf erase -y gcc-c++ meson gtest-devel git
dnf clean all
rm -rf /var/cache/*

%runscript
/path/to/your/app
```

Singularity и Docker

```
$ singularity build my-python docker://python:latest
$ ./my-python --version
```

Singularity и Docker

```
Bootstrap: docker
From: ubuntu:16.04

%post
apt-get -y update
apt-get -y install fortune cowsay lolcat

%environment
export LC_ALL=C
export PATH=/usr/games:$PATH

%runscript
fortune | cowsay | lolcat
```

Singularity и OpenGL

```
Bootstrap: yum
OSVersion: 28
MirrorURL: https://...
Include: dnf

%post
dnf --refresh -y install VirtualGL hostname mesa-dri-drivers

%runscript
vglrun glxspheres64
```

По типу программ:

- Docker: сервисы на несколько узлов и т.п.
- Singularity: пакетная обработка данных, высокопроизводительные вычисления, сборка кода.
- Vagrant: сборка кода, сервисы на несколько узлов, самый переносимый вариант.

По назначению:

- Docker: тестирование, развертка.
- Singularity: сборка, тестирование, развертка.
- Vagrant: сборка, тестирование.

Ссылки

- Flame Graph.
- Singularity User Guide.