



Uniwersytet
Wrocławski

Wsparcie ze strony kompilatora

Zbigniew Koza

Wydział Fizyki i Astronomii

Opcje kompilatora (tylko gcc)

- Optymalizacja kodu
- Debugowanie kodu
- Profilowanie kodu
(ogólnie: „Instrumentacja”)

Optymalizacja

- -O0 Wyłącza optymalizację
- -O1 Włącza podstawowe optymalizacje
- -O2 Włącza większość optymalizacji
- -O3 Włącza prawie wszystkie optymalizacje, kompilacja może trwać długo
- -Os Stara się uzyskać jak najkrótszy kod
- -march=native Kompiluje na daną maszynę

Optymalizacja

– opcje domyśłe w cmake

CMAKE_BUILD_TYPE =	DEBUG	-g
	MINSIZEREL	-Os -DNDEBUG
	RELEASE	-O3 -DNDEBUG
	RELWITHDEBINFO	-O2 -g -DNDEBUG

- -DNDEBUG wyłącza asercje
- -O3 w praktyce wyklucza debugowanie

ccmake

Page 1 of 1

CMAKE_BUILD_TYPE
CMAKE_INSTALL_PREFIX
SHORT_FLOAT_TYPE

Release
/usr/local
OFF

CMAKE_BUILD_TYPE: Choose the type of build, options are: None Debug Release RelWithDebInfo MinSizeRel
Keys: [enter] Edit an entry [d] Delete an entryCMake Version 3.22.1
[l] Show log output [c] Configure
[h] Help [q] Quit without generating
[t] Toggle advanced mode (currently off)

Uwaga!

- W większości przypadków różnice między -02 i -03 czy włączeniem lub nie opcji -march=native są minimalne lub niezauważalne
- Potrzeba czasu na eksperymentowanie

Debugowanie

Podstawowa opcja

- > g++ -g
 - Nie działa z bardzo agresywną optymalizacją (np. -O3)
 - Umożliwia śledzenie programu debugerem
 - Ułatwia analizę raportów generowanych przez inne narzędzia służące do odpluskwiania programów

coredump

1

```
> ./a.out 1 0
4782
Błąd w obliczeniach zmiennoprzecinkowych (zrzut pamięci)
```

2

```
Thu 2022-01-20 23:17:13 CET 4769 1000 1000 SIGFPE present /home/zkoza/Pulpit/Dydaktyka/zcpp/w13-instrumentation/cpp/a.out
Thu 2022-01-20 23:17:37 CET 4782 1000 1000 SIGFPE present /home/zkoza/Pulpit/Dydaktyka/zcpp/w13-instrumentation/cpp/a.out
```

3

```
> coredumpctl gdb 4782
```

4

```
Core was generated by './a.out 1 0'.
Program terminated with signal SIGFPE, Arithmetic exception.
#0  0x000055e05fb183c3 in main (argc=3, argv=0x7ffe6a25ecf8) at gdb.cpp:13
13      std::cout << "wynik: " << x / y << "\n";
(gdb) print x
$1 = 1
(gdb) print y
$2 = 0
(gdb) q
```

```
// compile with -g
#include <iostream>
#include <unistd.h>

int main(int argc, const char* argv[])
{
    std::cout << getpid() << "\n";
    if (argc >= 3)
    {
        int x = std::stoi(argv[1]);
        int y = std::stoi(argv[2]);
        std::cout << "wynik: " << x / y << "\n";
    }

    int i = 1;
    while(i)
    {
        continue;
    }
    std::cout << "Koniec\n";
}

// coredumpctl list
// coredumpctl info pid
// coredumpctl gdb pid
// objdump ./a.out -S | vim -
```

Process id

Bum!

Debugowanie żywego procesu

```
// compile with -g
#include <iostream>
#include <unistd.h>

int main(int argc, const char* argv[])
{
    std::cout << getpid() << "\n";
    if (argc >= 3)
    {
        int x = std::stoi(argv[1]);
        int y = std::stoi(argv[2]);
        std::cout << "wynik: " << x / y << "\n";
    }

    int i = 1;
    while(i)
    {
        continue;
    }
    std::cout << "Koniec\n";
}

// coredumpctl list
// coredumpctl info pid
// coredumpctl gdb pid
// objdump ./a.out -S | vim -
```

Process id

Bum!

```
> ./a.out 1 2
5352
wynik: 0
```

Otwieramy drugą konsolę

```
> sudo gdb -p 5352
[sudo] hasło użytkownika zkoza:
GNU gdb (GDB) 11.1
main (argc=3, argv=0x7ffd50c3e408) at gdb.cpp:19
19         continue;
(gdb) list
14     }
15
16     int i = 1;
17     while(i)
18     {
19         continue;
20     }
21     std::cout << "Koniec\n";
22 }
23
(gdb) set variable i = 0
(gdb) next
17     while(i)
(gdb) next
21     std::cout << "Koniec\n";
```

Code instrumentation

Code instrumentation

- Technika, która nakazuje kompilatorowi dodać do kodu programu wykonywalnego dodatkowy kod diagnostyczny
- Kod zwykle działa wolniej, ale realizuje wyznaczone mu dodatkowe cele
- Często wynik działania tego dodatkowego kodu automatycznie umieszczany jest w dodatkowych plikach wynikowych podczas kończenia pracy programu

PGO (*profile-guided optimization*)

- > g++ -O3 -march=native \
-fprofile-generate
- > ./a.out ← Generuje plik *.gcda
- > g++ -O3 -march=native \
-fprofile-use
- > ./a.out

Zwykła kompilacja:

```
Time: 0.457513 sec (with attributes:)  
Time: 0.600341 sec (without attributes)  
Time: 0.261399 sec (std::cos)
```



```
Time: 0.516903 sec (with attributes:)  
Time: 0.517345 sec (without attributes)  
Time: 0.255938 sec (std::cos)
```


Kod programu: cos.cpp

Analiza pokrycia

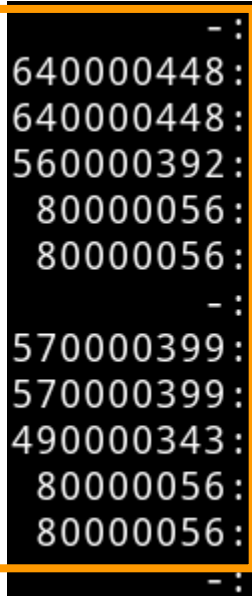

- Czy wszystkie instrukcje mojego programu są wykonywane (jeśli nie, to albo mam błąd, albo mogę uprościć program)
- **Czy testy obejmują cały program?**
- Ile razy wykonywane są określone instrukcje w programie
 - Przydatne przy analizie wąskich gardeł
 - Ważne przy określaniu, które gałęzie wyrażeń warunkowych powinny mieć wyższy priorytet

Analiza pokrycia

- > g++ --coverage cos.cpp
- > ./a.out
- > gcov z.cpp



```
File 'z.cpp'  
Lines executed:100.00% of 54  
Creating 'z.cpp.gcov'
```



```
-:      7:namespace with_attributes {  
640000448:    8:constexpr double pow(double x, long long n) noexcept {  
640000448:    9:      if (n > 0) [[likely]]  
560000392:   10:        return x * pow(x, n - 1);  
 80000056:   11:      else [[unlikely]]  
 80000056:   12:        return 1;  
-:      13:}  
570000399:   14:constexpr long long fact(long long n) noexcept {  
570000399:   15:      if (n > 1) [[likely]]  
490000343:   16:        return n * fact(n - 1);  
 80000056:   17:      else [[unlikely]]  
 80000056:   18:        return 1;  
-:      19:}
```

Profiler

- > g++ -pg
- > ./a.out
- > gprof a.out



```
> gprof ./a.out
```

```
Flat profile:
```

```
Each sample counts as 0.01 seconds.
```

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
25.04	2.03	2.03	80000056	0.00	0.00	with_attributes::pow(double, long long)
21.34	3.77	1.73	79999992	0.00	0.00	no_attributes::pow(double, long long)
10.61	4.63	0.86	79999992	0.00	0.00	no_attributes::fact(long long)
10.61	5.49	0.86	80000056	0.00	0.00	with_attributes::fact(long long)
7.28	6.08	0.59	96154	0.00	0.00	std::mersenne_twister_engine<unsigned lo
7ul, 2636928640ul, 15ul, 4022730752ul, 18ul, 1812433253ul>::_M_gen_rand()						
4.93	6.48	0.40	10000007	0.00	0.00	with_attributes::cos(double)

Profiler

- Jakie funkcje zajęły jaki procent czasu działania programu
- Ile razy były wywoływane i z jakich funkcji

```
index % time    self  children    called    name
-----
[1]    99.9      0.00    8.12          3/3    <spontaneous>
      0.01      8.11          3/3    auto main::{lambda(auto:1, auto:2)#1}::operator()<
) noexcept, char const*) const [2]
      0.00      0.00      8/10000007    with_attributes::cos(double) [3]
      0.00      0.00      4/7          std::setprecision(int) [41]
      0.00      0.00      1/2          std::initializer_list<double>::begin() const [57]
      0.00      0.00      1/1          std::initializer_list<double>::end() const [61]
-----
[2]    99.9      0.01      8.11          3/3    main [1]
      0.01      8.11          3          auto main::{lambda(auto:1, auto:2)#1}::operator()<doub
except, char const*) const [2]
      0.40      2.90 9999999/10000007    with_attributes::cos(double) [3]
      0.24      2.59 9999999/9999999    no_attributes::cos(double) [4]
      0.11      1.87 29999997/29999997    gen_random() [6]
```

Profiler

- Używaj wyłącznie z włączonymi flagami optymalizacji
- Profiler g++ słabo radzi sobie z programami wielowątkowymi
- Alternatywy (nie wymagają instrumentacji kodu):
 - valgrind
 - perf
 - oprofile

perf

- Dwie wersje programu różniące się definicją tablicy

perf stat -e cache-misses,cache-references,LLC-loads,LLC-stores,L1-dcache-load-misses,L1-dcache-prefetch-misses,L1-dcache-store-misses ./a.out 100000000 0

```
Performance counter stats for './a.out 100000000 1':
```

4 210	cache-misses:u	#	11,882 % of all cache refs	(57,36%)
35 431	cache-references:u			(57,30%)
26 351	LLC-loads:u			(57,49%)
2 687	LLC-stores:u			(57,49%)
54 490	L1-dcache-load-misses:u			(57,81%)
625	L1-dcache-prefetch-misses:u			(57,87%)
235	L1-dcache-store-misses:u			(57,68%)

```
0,188834274 seconds time elapsed
```

```
0,528938000 seconds user
```

```
0,000000000 seconds sys
```

perf stat -e cache-misses,cache-references,LLC-loads,LLC-stores,L1-dcache-load-misses,L1-dcache-prefetch-misses,L1-dcache-store-misses ./a.out 100000000 1

```
Performance counter stats for './a.out 1000000000 0':
```

15 031	cache-misses:u	#	0,060 % of all cache refs	(56,93%)
25 040 244	cache-references:u			(57,04%)
16 731 415	LLC-loads:u			(57,03%)
8 236 694	LLC-stores:u			(57,68%)
26 885 503	L1-dcache-load-misses:u			(58,52%)
14 409 306	L1-dcache-prefetch-misses:u			(57,85%)
25 748	L1-dcache-store-misses:u			(57,48%)

```
0,215386523 seconds time elapsed
```

```
0,617845000 seconds user
```

```
0,009982000 seconds sys
```

Sanitizers

- Potężne narzędzie służące do wychwytywania błędów w programie
- G++ dodaje do kompilowanego programu dodatkowe instrukcje mające wychwycić różnego typu błędy
- Istnieje kilka sanitizerów; nie mogą one działać jednocześnie

Sanitizery

- Podstawowe:

-fsanitize=address

-fsanitize=thread

-fsanitize=undefined

AddressSanitizer (ASAN)

- <https://github.com/google/sanitizers/wiki/AddressSanitizer>

```
> g++ -g -fsanitize=address asan1.cpp
```

```
#include <iostream>
int main()
{
    int tab[10];
    for (int i = 0; i <= 10; i++)
        tab[i] = i;
}
```



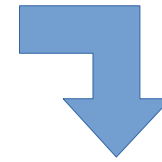
```
> ./a.out
=====
==22399==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fff75618708 at pc 0x55b8075e62cc
WRITE of size 4 at 0x7fff75618708 thread T0
#0 0x55b8075e62cc in main /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/asan1.cpp:6
#1 0x7f83e6d12151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)
#2 0x55b8075e610d in _start (/home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/a.out+0x110d)

Address 0x7fff75618708 is located in stack of thread T0 at offset 88 in frame
#0 0x55b8075e61e8 in main /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/asan1.cpp:3

This frame has 1 object(s):
[48, 88) 'tab' (line 4) <== Memory access at offset 88 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapco
(longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/c
Shadow bytes around the buggy address:
```

AddressSanitizer (ASAN)

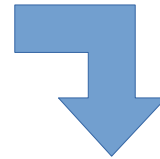
```
#include <iostream>
int main()
{
    int * tab = new int[10];
    tab[0] = 0;
    delete [] tab;
    std::cout << tab[0];
}
```



```
> ./a.out
=====
==22686==ERROR: AddressSanitizer: heap-use-after-free on address 0x604000000010 at pc 0x56170383d1b0
READ of size 4 at 0x604000000010 thread T0
#0 0x56170383d299 in main /home/zkoza/Pulpit/Dydaktyka/P0-2020/W-compiler/cpp/asan2.cpp:7
#1 0x7fbf0b9ce151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)
#2 0x56170383d12d in _start (/home/zkoza/Pulpit/Dydaktyka/P0-2020/W-compiler/cpp/a.out+0x112d)
0x604000000010 is located 0 bytes inside of 40-byte region [0x604000000010,0x604000000038)
freed by thread T0 here:
#0 0x7fbf0bf60c09 in operator delete[](void*) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:100
#1 0x56170383d262 in main /home/zkoza/Pulpit/Dydaktyka/P0-2020/W-compiler/cpp/asan2.cpp:6
#2 0x7fbf0b9ce151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)
previously allocated by thread T0 here:
#0 0x7fbf0bf600c1 in operator new[](unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:100
#1 0x56170383d20a in main /home/zkoza/Pulpit/Dydaktyka/P0-2020/W-compiler/cpp/asan2.cpp:4
#2 0x7fbf0b9ce151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)
```

AddressSanitizer (ASAN)

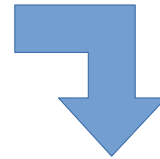
```
#include <iostream>
int main()
{
    int * tab = nullptr;
    std::cout << tab[0];
}
```



```
> ./a.out
AddressSanitizer:DEADLYSIGNAL
=====
==22794==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x55d93b29f214 bp 0x7f6760971151)
==22794==The signal is caused by a READ memory access.
==22794==Hint: address points to the zero page.
#0 0x55d93b29f214 in main /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/asan3.cpp:5
#1 0x7f6760971151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)
#2 0x55d93b29f0fd in _start (/home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/a.out+0x10fd)
```


AddressSanitizer (ASAN)

```
#include <iostream>
int main()
{
    int * tab = new int[10];
    std::cout << tab[0];
}
```



```
> ./a.out
-1094795586
=====
==22936==ERROR: LeakSanitizer: detected memory leaks

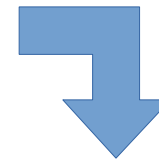
Direct leak of 40 byte(s) in 1 object(s) allocated from:
#0 0x7fa08ce7c0c1 in operator new[](unsigned long) /build/gcc/src/gcc/libsanitizer/asan/asan_new_delete.cpp:102
#1 0x557d5c78f1ea in main /home/zkoza/Pulpit/Dydaktyka/P0-2020/W-compiler/cpp/asan4.cpp:4
#2 0x7fa08c8ea151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)
```

ThreadSanitizer (TSAN)

- <https://github.com/google/sanitizers/wiki#threadsanitizer>

```
> g++ -g -fsanitize=thread tsan1.cpp -fopenmp
```

```
int main()
{
    int sum = 0;
    #pragma omp parallel for
    for (int i = 0; i < 2; i++)
        sum += i;
}
```



```
> ./a.out
```

```
=====
```

```
WARNING: ThreadSanitizer: data race (pid=23238)
```

```
Write of size 4 at 0x7ffc0d826790 by thread T1:
```

```
#0 main._omp_fn.0 /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/tsan1.cpp:6
```

```
#1 gomp_thread_start /build/gcc/src/gcc/libgomp/team.c:123 (libgomp.so.1+0x1a3ed)
```

```
Previous write of size 4 at 0x7ffc0d826790 by main thread:
```

```
#0 main._omp_fn.0 /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/tsan1.cpp:6
```

```
#1 GOMP_parallel /build/gcc/src/gcc/libgomp/parallel.c:171 (libgomp.so.1+0x126b5)
```

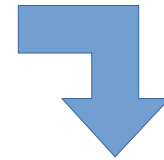
```
#2 __libc_start_main <null> (libc.so.6+0x28151)
```

UndefinedBehavior Sanitizer (UBSAN)

- <https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html>

```
> g++ -g -fsanitize=undefined ubsan1.cpp
```

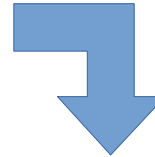
```
#include <limits>
int main()
{
    int i = std::numeric_limits<int>::max();
    i++;
}
```



```
> ./a.out
ubsan1.cpp:5:4: runtime error: signed integer overflow: 2147483647 + 1 cannot be represented in type 'int'
#0 0x56102468616b in main /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/ubsan1.cpp:5
#1 0x7f23c999f151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)
#2 0x56102468606d in _start (/home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/a.out+0x106d)
```

UndefinedBehavior Sanitizer (UBSAN)

```
int f() {}  
  
int main()  
{  
    f();  
}
```



```
ubsan2.cpp:1:5: runtime error: execution reached the end of a value-returning function without returning a value  
#0 0x563ef9f80148 in f() /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/ubsan2.cpp:1  
#1 0x563ef9f80151 in main /home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/ubsan2.cpp:5  
#2 0x7f268a837151 in __libc_start_main (/usr/lib/libc.so.6+0x28151)  
#3 0x563ef9f8006d in start (/home/zkoza/Pulpit/Dydaktyka/PO-2020/W-compiler/cpp/a.out+0x106d)
```

Sanitizery

- Nowoczesny sposób diagnostyki programów w C++
- Zwykle wiążą się z odczuwalnym zmniejszeniem prędkości badanych programów, a także ze znacznym spowolnieniem kompilacji

Wniosek

- Znaj swój kompilator