Delivering Sample-based PGO for PlayStation®4

(and the impact on optimized debugging)

Greg Bedwell ○ Andrea Di Biagio ○ Robert Lougher

Sony Interactive Entertainment

A typical PS4 game…

- Is written in C++

- Needs to run at:
  - 30 fps **(~33.3 ms**/frame) or 60 fps (**~16.6 ms**/frame)
  - **Every cycle counts**!

- Needs to be as debuggable as possible
  - with full optimization (-O0 is **not an option**)

- Has tight project deadlines
  - Developers don’t often have much time to try out new compiler features

Profile Guided Optimization

- 2 Primary methods:
  - Instrumentation-based
    - Front-end (`-fprofile-instrument=clang`)
    - IR (`-fprofile-instrument=llvm`)
  - Sample-based
    - See https://research.google.com/pubs/pub45290.html

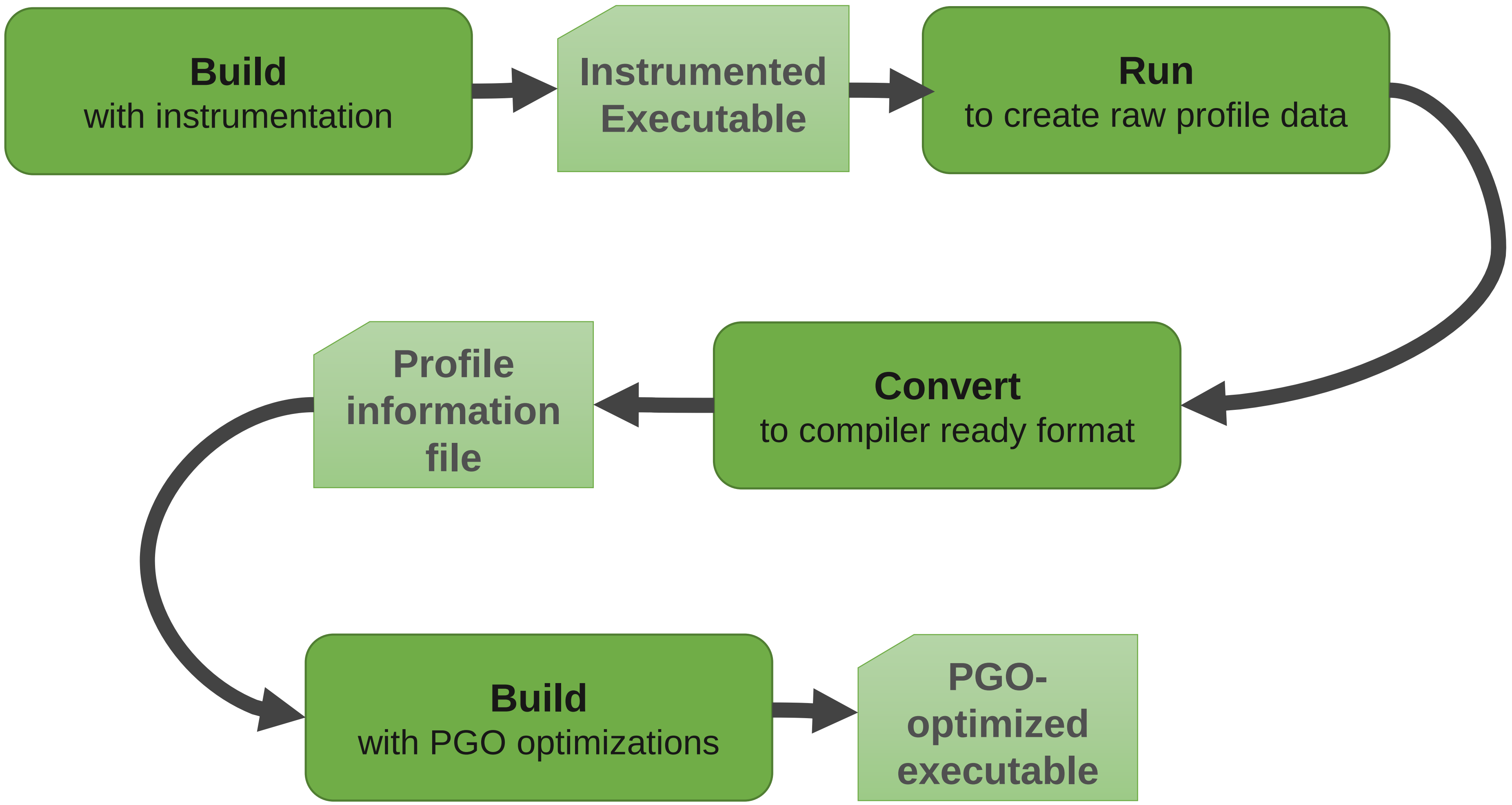
      AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications
      Dehao Chen, David Xinliang Li, Tipp Moseley
      CGO 2016 Proceedings of the 2016 International Symposium on Code Generation and Optimization, ACM, New York, NY, USA, pp. 12-23

Profile Guided Optimization

- 2 Primary methods:
  - Instrumentation-based
    - ~~Front-end (`-fprofile-instrument=clang`)~~
    - IR (`-fprofile-instrument=llvm`)
  - Sample-based
    - See https://research.google.com/pubs/pub45290.html

      AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications
      Dehao Chen, David Xinliang Li, Tipp Moseley
      CGO 2016 Proceedings of the 2016 International Symposium on Code Generation and Optimization, ACM, New York, NY, USA, pp. 12-23

Instrumentation-based PGO



Instrumentation-based PGO

```
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

Instrumentation-based PGO

```
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

Instrumentation-based PGO

```c
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

```
$ clang ex1.c -S -O2

func:                                    # @func
  .cfi_startproc
# BB#0:                                  # %entry
  testl %edi, %edi
  je  .LBB1_1
# BB#2:                                  # %if.then
  jmp handle_error@PLT         # TAILCALL
.LBB1_1:                                 # %return
  movl  %esi, %eax
  retq
.Lfunc_end1:
```

Instrumentation-based PGO

```c
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

```
$ clang ex1.c -S -O2 -fprofile-instr-generate

func:                                   # @func
  .cfi_startproc
# BB#0:                                 # %entry
  movl  %esi, %eax
  testl %edi, %edi
  je   .LBB1_2
# BB#1:                                 # %if.then
  pushq %rbp
.Ltmp5:
  .cfi_def_cfa_offset 16
.Ltmp6:
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp7:
  .cfi_def_cfa_register %rbp
  incq  .L__profc_ex1.c_func+8(%rip)
  callq handle_error@PLT
  popq  %rbp
.LBB1_2:                                # %return
  incq  .L__profc_ex1.c_func(%rip)
  retq
.Lfunc_end1:
```

Instrumentation-based PGO

```
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

```
$ clang -O2                                   \
  -fprofile-instr-generate=p.profraw          \
  ex1.c ex2.c -o ex.elf

$ llvm-profdata merge p.profraw               \
  -text -output p.txt

ex1.c:func
# Func Hash:
22759827559
# Num Counters:
2
# Counter Values:
1000000000
10000000
```

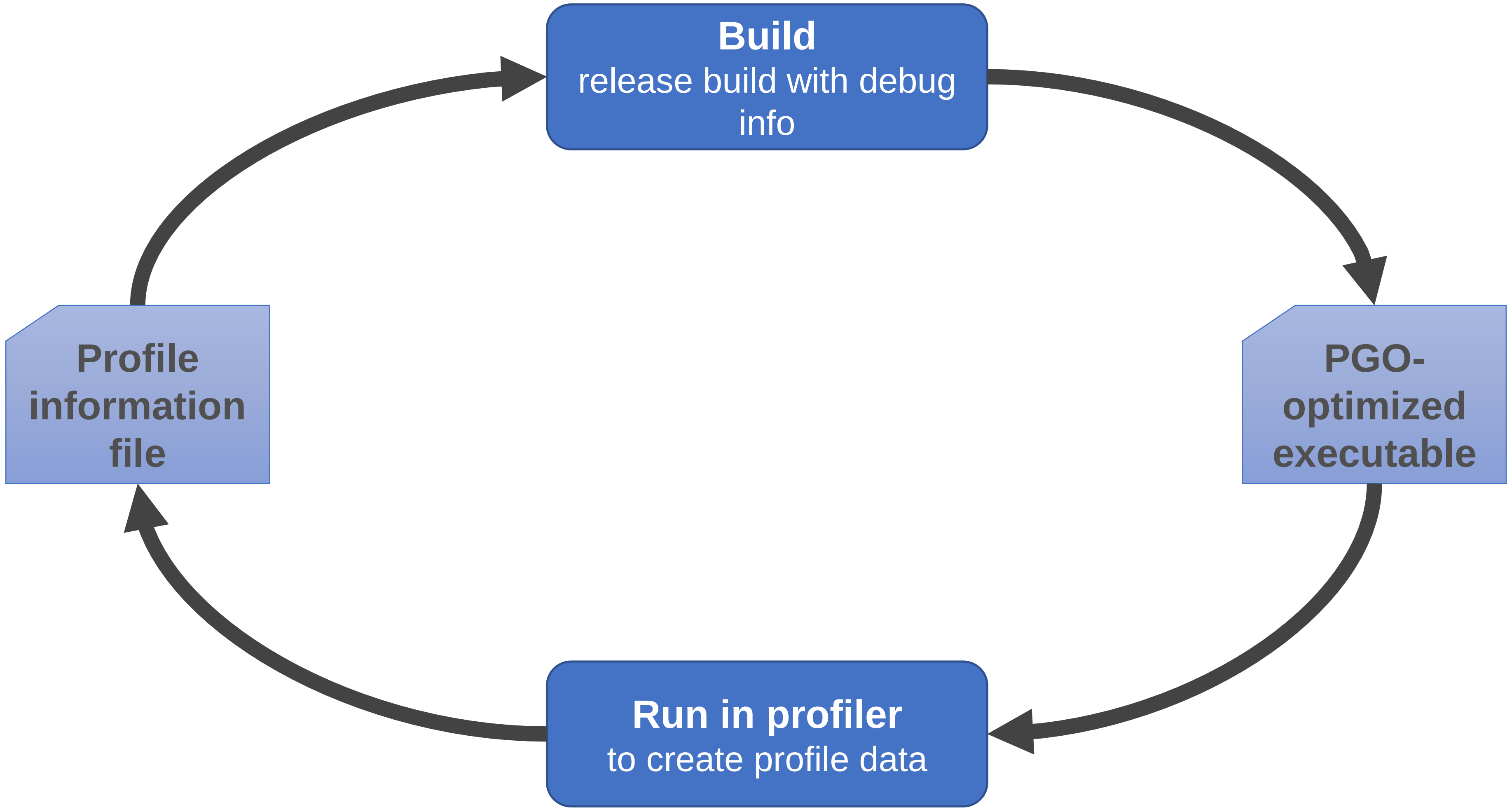
Instrumentation-based PGO

```
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

```
$ llvm-profdata merge p.profraw                  \
   -output p.profdata

$ clang -O2                                       \
  -fprofile-instr-use=p.profdata -S ex1.c
```

```
func:                                       # @func
  .cfi_startproc
# BB#0:                                     # %entry
  testl %edi, %edi
  jne .LBB1_2
# BB#1:                                     # %return
  movl  %esi, %eax
  retq
.LBB1_2:                                    # %if.then
  jmp handle_error@PLT            # TAILCALL
.Lfunc_end1:
```

Sample-based PGO



Sample-based PGO

```
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

Sample-based PGO

```c
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

```
$ clang -S -O2 -gmlt ex1.c

func:                                          # @func
.Lfunc_begin1:
          .loc       1 4 0                     # ex1.c:4:0
          .cfi_startproc
# BB#0:                                        # %entry
          .loc       1 5 0 prologue_end        # ex1.c:5:0
          testl      %edi, %edi
          je         .LBB1_1
# BB#2:                                        # %if.then
          .loc       1 6 0                     # ex1.c:6:0
          jmp        handle_error@PLT          # TAILCALL
.LBB1_1:                                       # %return
          .loc       1 9 0                     # ex1.c:9:0
          movl       %esi, %eax
          retq
.Ltmp7:
.Lfunc_end1:
```

Sample-based PGO

```
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

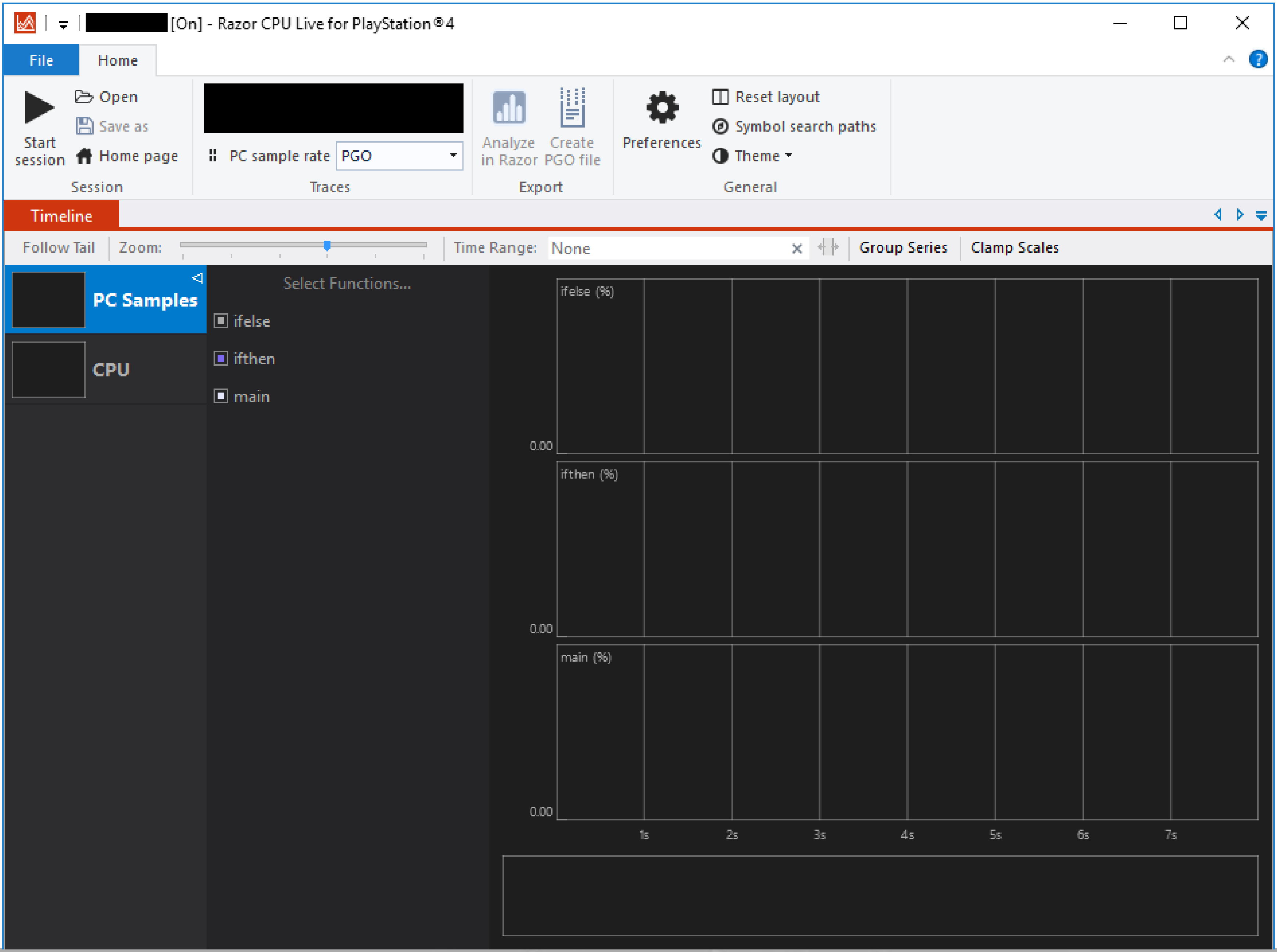
```
llvm-profdata show -sample p.ps4prof

Function: func: 6552476, 1089549, 3 sampled
lines
Samples collected in the function's body {
  1: 1089549
  2: 10879, calls: handle_error:10879
  5: 1087372
}
No inlined callsites in this function
```

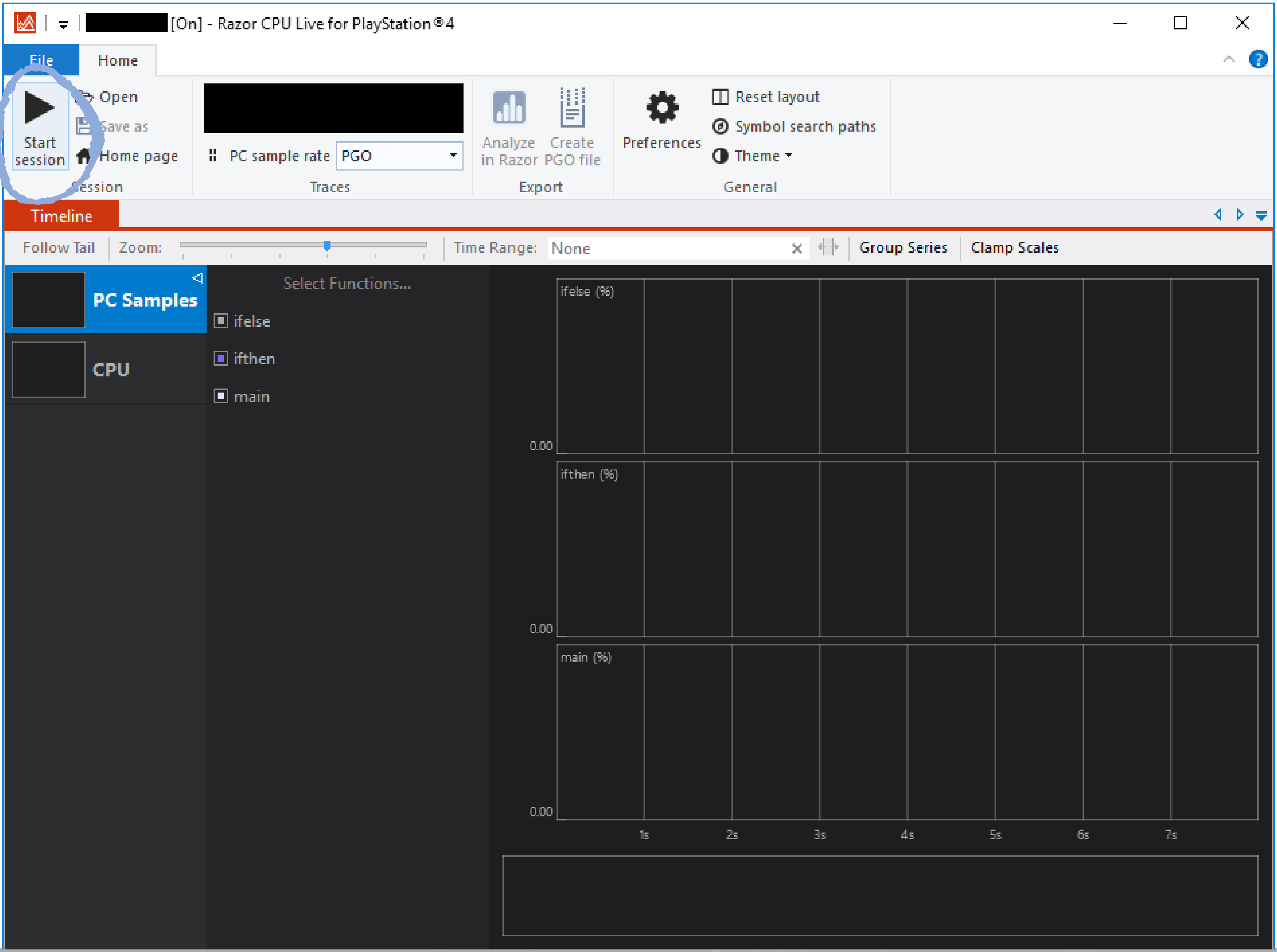
Sample-based PGO

```c
extern int handle_error(int);

__attribute__((__noinline__))
static int func(int error, int val) {
  if (error)
    return handle_error(error);

  return val;
}

int main() {
  int r = 0;
  for (int i = 0; i < 1000000000; ++i) {
    r += func((i % 100) == 0, i);
  }
  return r;
}
```

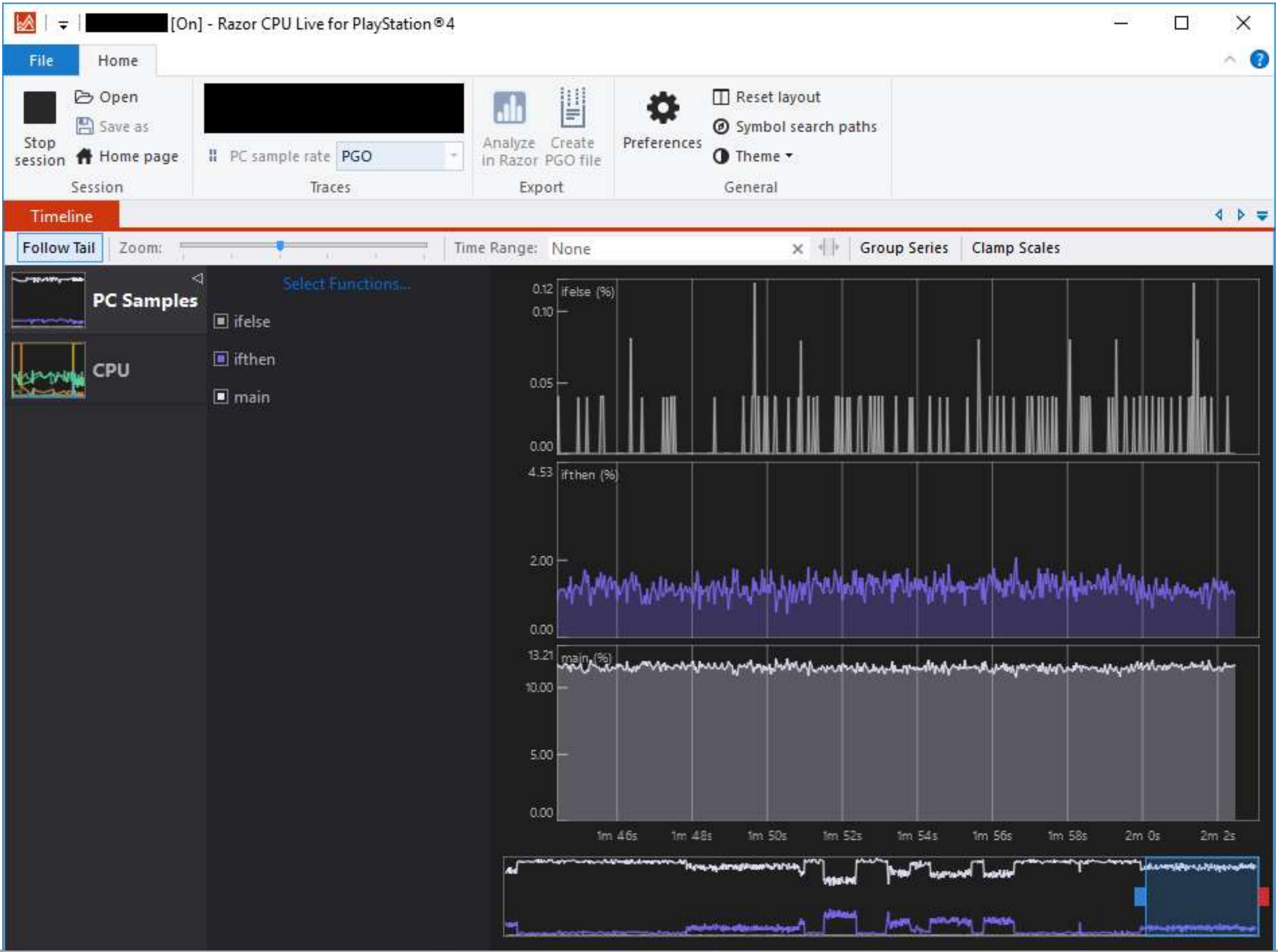
```
$ clang -S -O2 -gmlt ex1.c                              \
  -fprofile-sample-use=p.ps4prof

func:                                          # @func
.Lfunc_begin1:
           .loc       1 4 0                    # ex1.c:4:0
           .cfi_startproc
# BB#0:                                        # %entry
           .loc       1 5 0 prologue_end       # ex1.c:5:0
           testl      %edi, %edi
           jne        .LBB1_2
# BB#1:                                        # %return
           .loc       1 9 0                    # ex1.c:9:0
           movl       %esi, %eax
           retq
.LBB1_2:                                       # %if.then
           .loc       1 6 0                    # ex1.c:6:0
           jmp        handle_error@PLT         # TAILCALL
.Ltmp7:
.Lfunc_end1:
```

Sample-based PGO Workflow


[redacted] [On] - Razor CPU Live for PlayStation®4
File
Home
Start session
Open
Save as
Home page
Session
[redacted]
PC sample rate
PGO
Traces
Analyze in Razor
Create PGO file
Export
Preferences
Reset layout
Symbol search paths
Theme
General
Timeline
Follow Tail
Zoom:
Time Range:
None
Group Series
Clamp Scales
PC Samples
CPU
Select Functions...
ifelse
ifthen
main
ifelse (%)
ifthen (%)
main (%)
0.00
1s
2s
3s
4s
5s
6s
7s

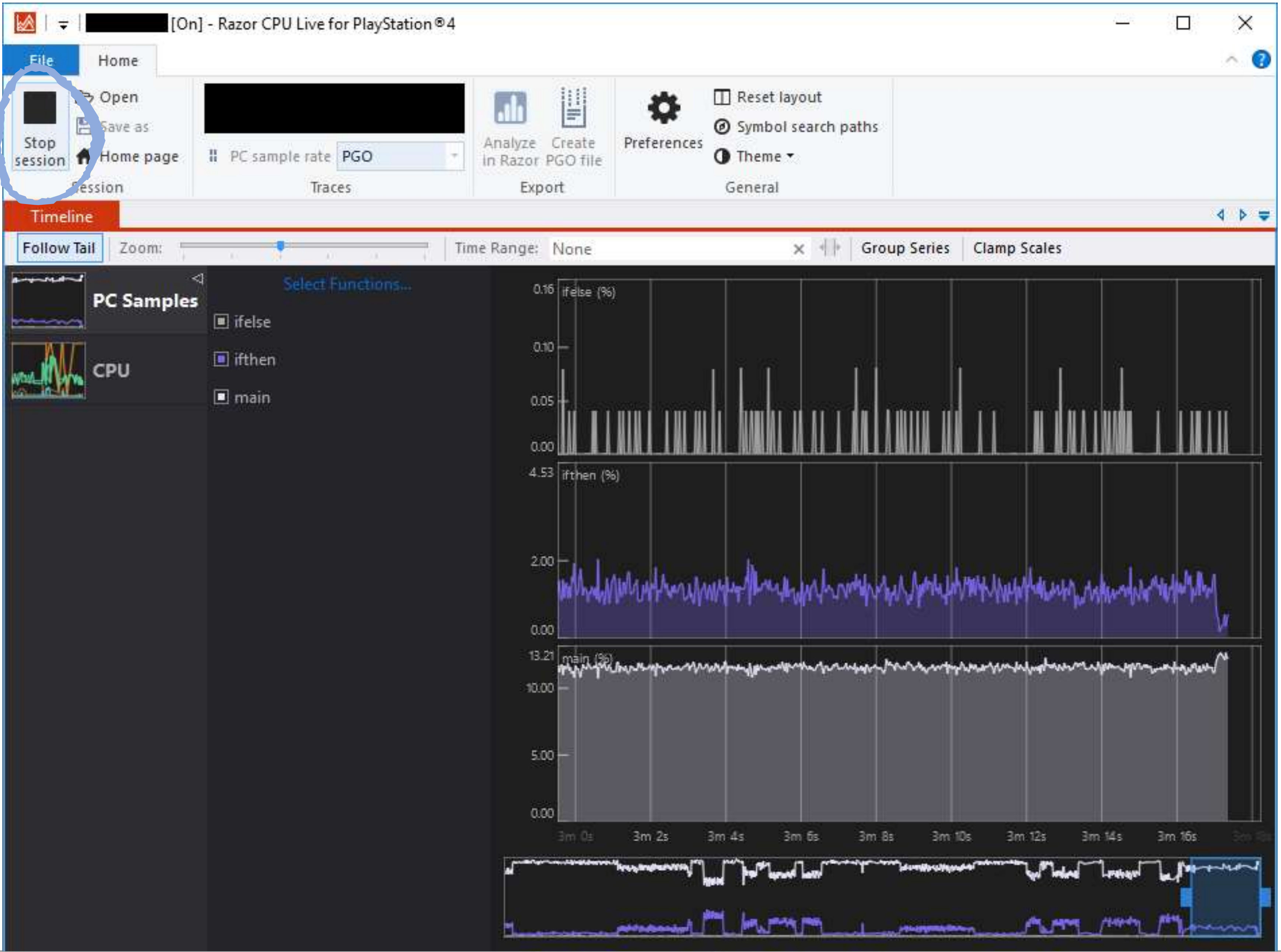


[redacted] [On] - Razor CPU Live for PlayStation®4



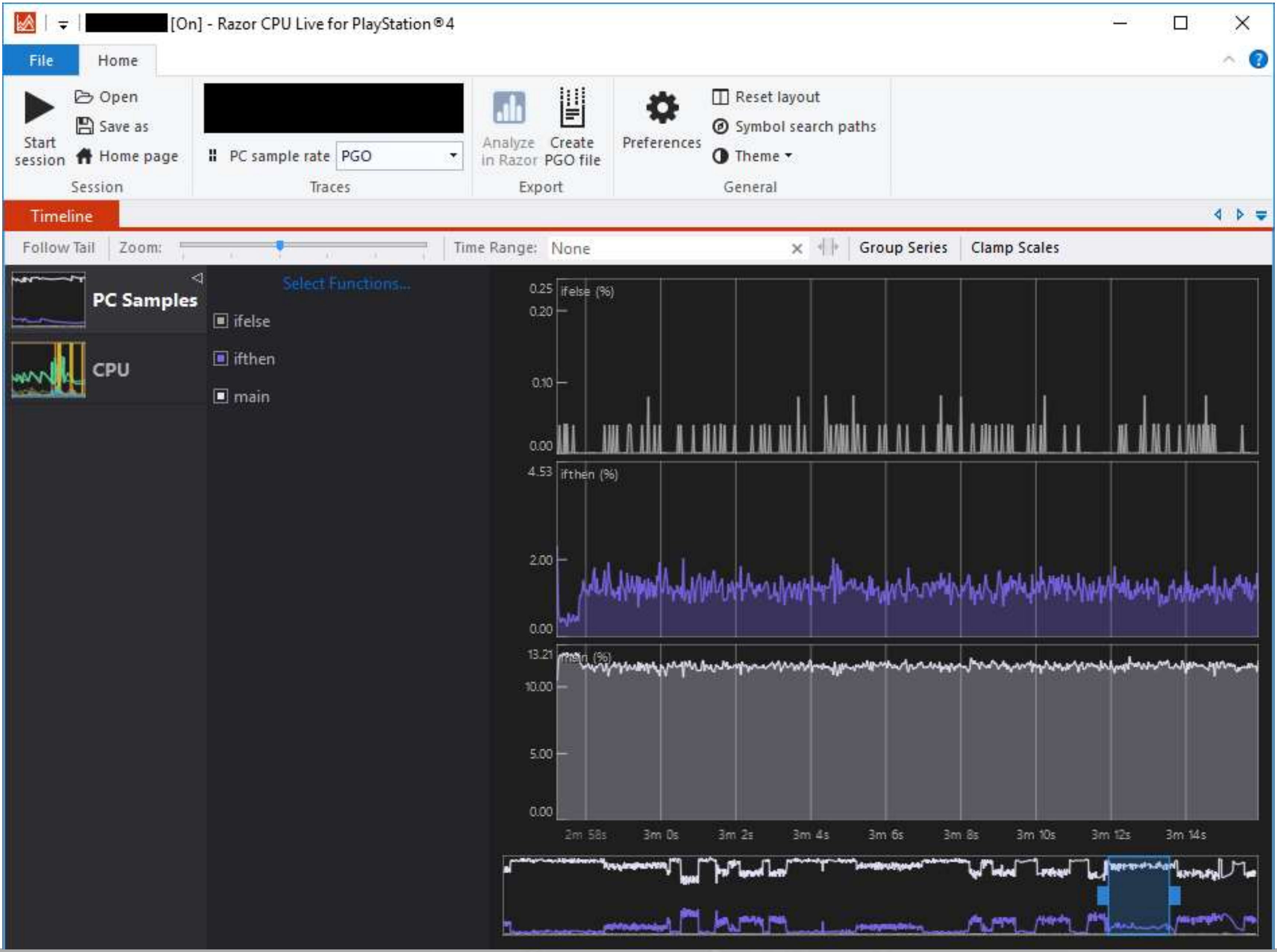
[redacted] [On] - Razor CPU Live for PlayStation®4
File
Home
Stop session
Open
Save as
Home page
Session
[redacted]
PC sample rate PGO
Traces
Analyze in Razor
Create PGO file
Export
Preferences
Reset layout
Symbol search paths
Theme
General
Timeline
Follow Tail
Zoom:
Time Range: None
Group Series
Clamp Scales
PC Samples
CPU
Select Functions...
ifelse
ifthen
main
ifelse (%)
ifthen (%)
main (%)
0.12
0.10
0.05
0.00
4.53
2.00
0.00
13.21
10.00
5.00
0.00
1m 46s
1m 48s
1m 50s
1m 52s
1m 54s
1m 56s
1m 58s
2m 0s
2m 2s

SHARE
OPTIONS



PlayStation.

sn systems

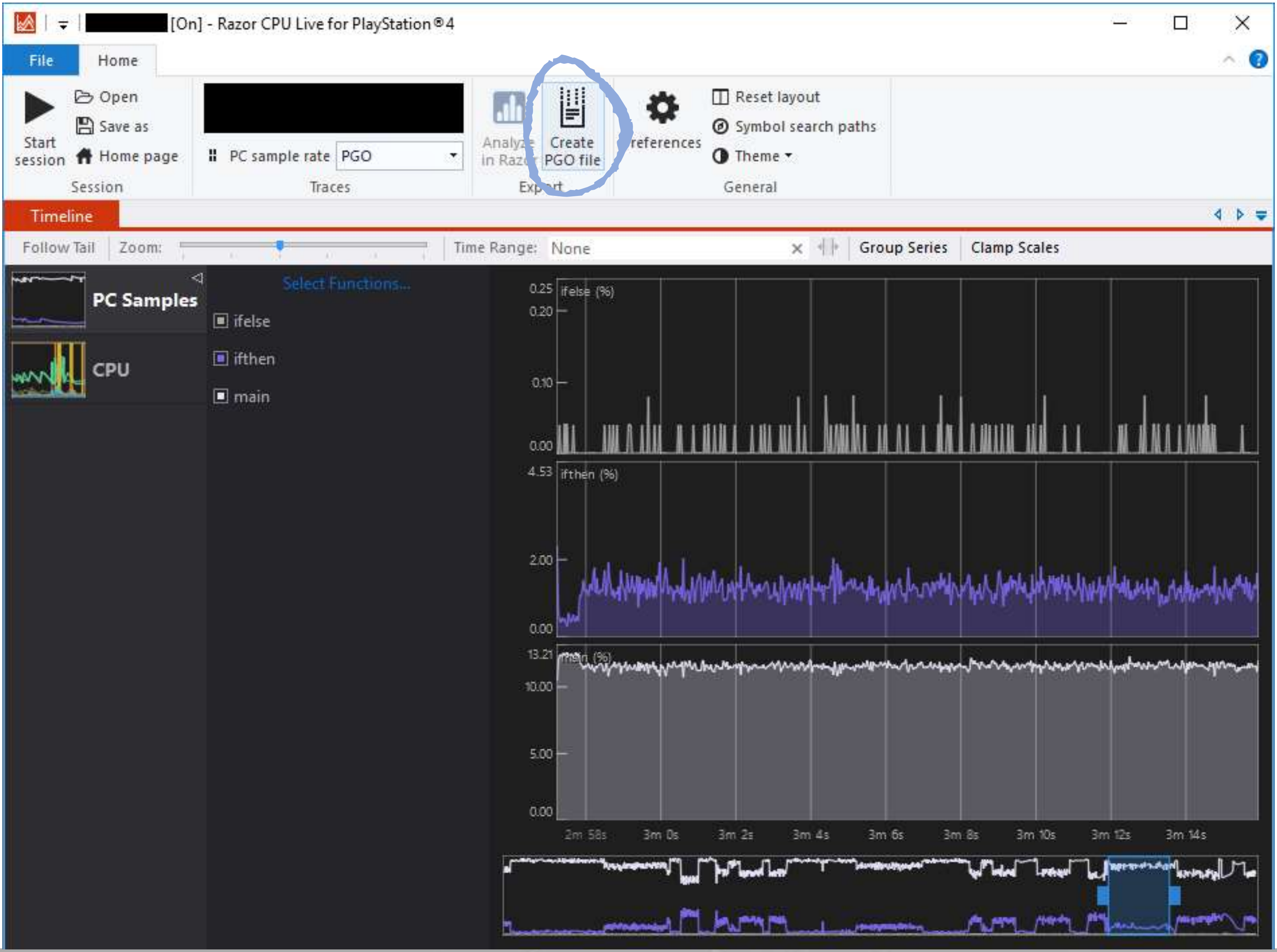


[redacted] [On] - Razor CPU Live for PlayStation®4

[redacted]

PC sample rate: PGO

Time Range: None

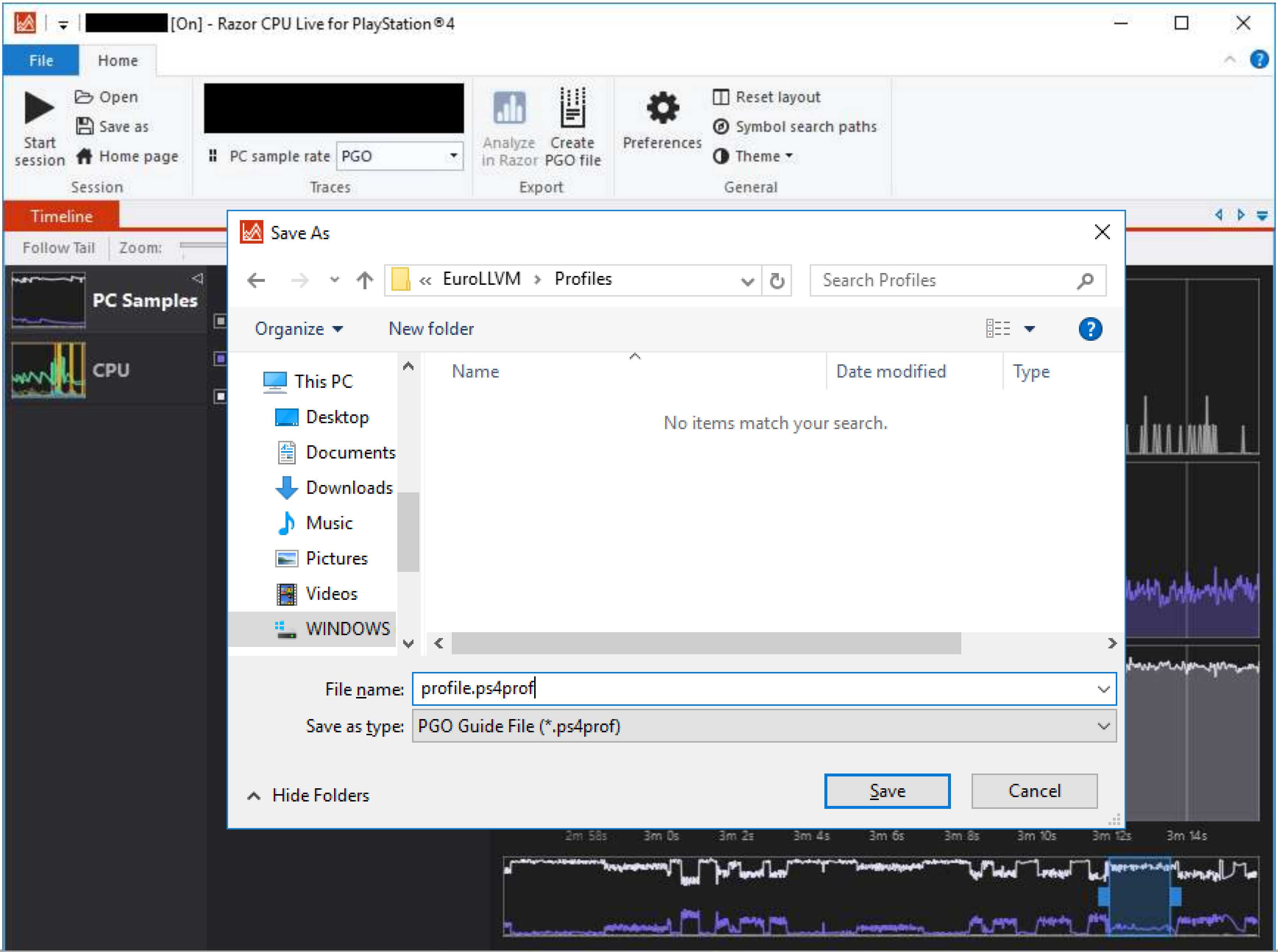


[redacted] [On] - Razor CPU Live for PlayStation®4

[redacted]

PC sample rate: PGO

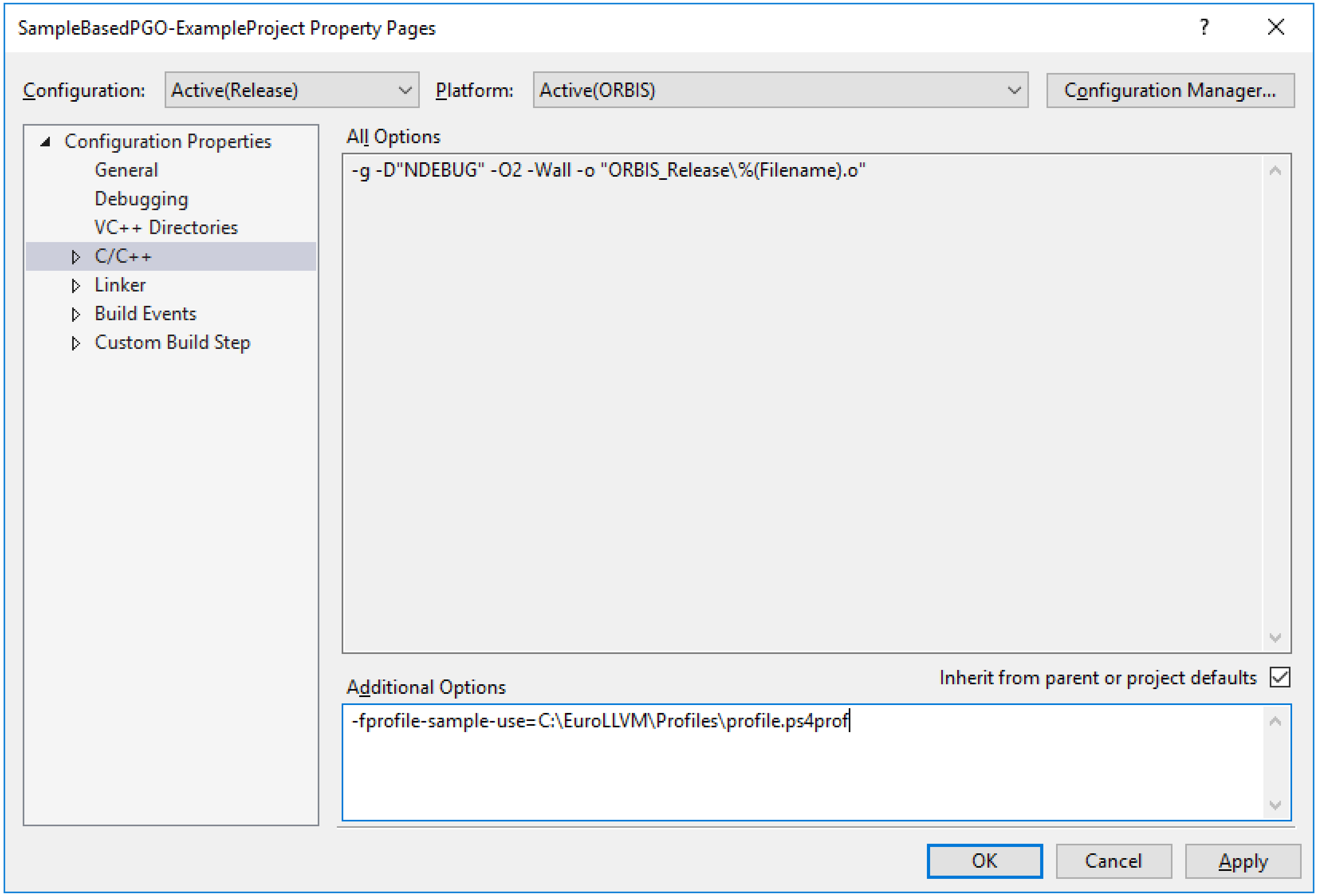
Time Range: None



PC sample rate: PGO

File name: profile.ps4prof

Save as type: PGO Guide File (*.ps4prof)

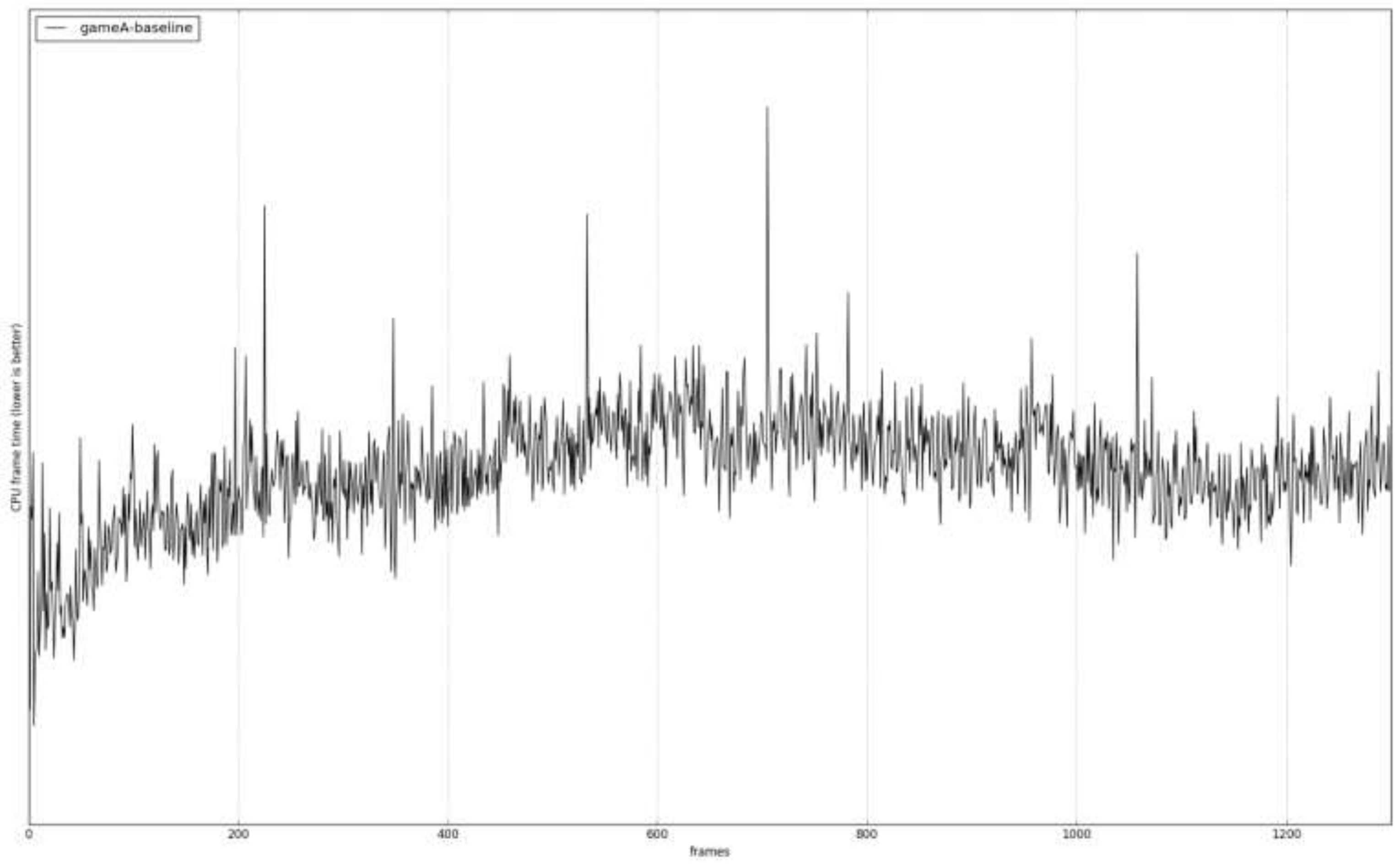


```
-g -D"NDEBUG" -O2 -Wall -o "ORBIS_Release\%(Filename).o"
```

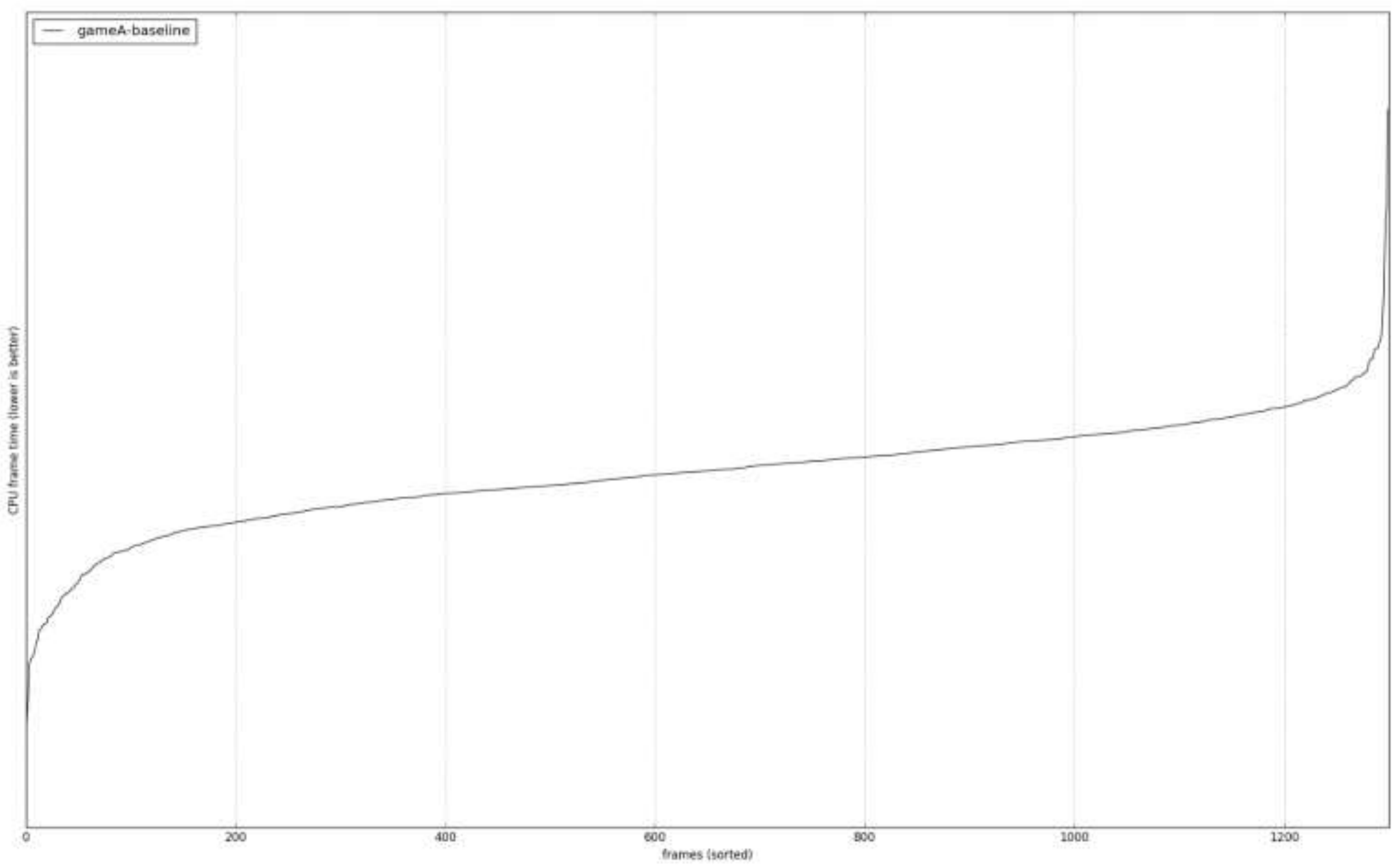
```
-fprofile-sample-use=C:\EuroLLVM\Profiles\profile.ps4prof
```

Initial Results

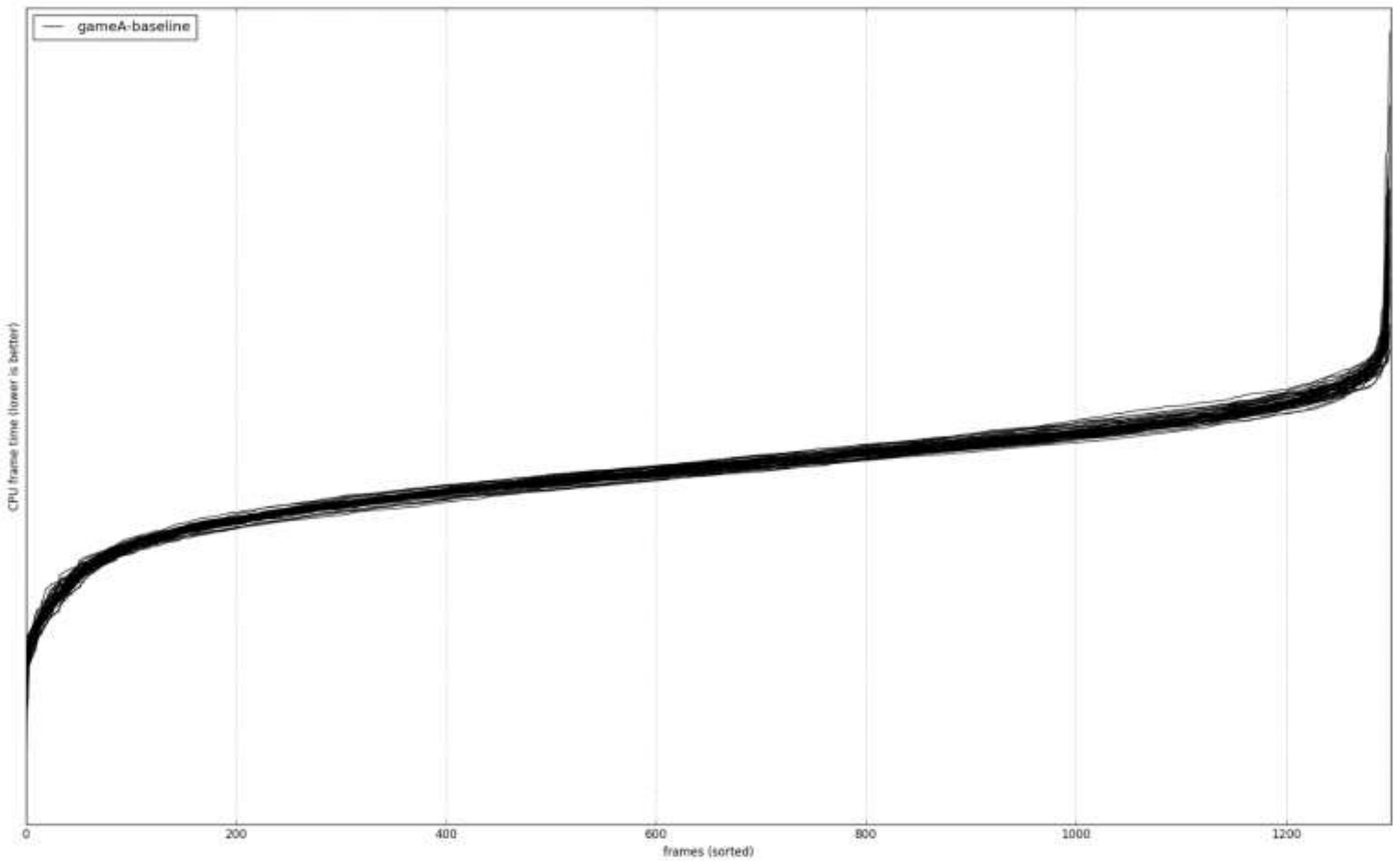
Frame time (Game A)



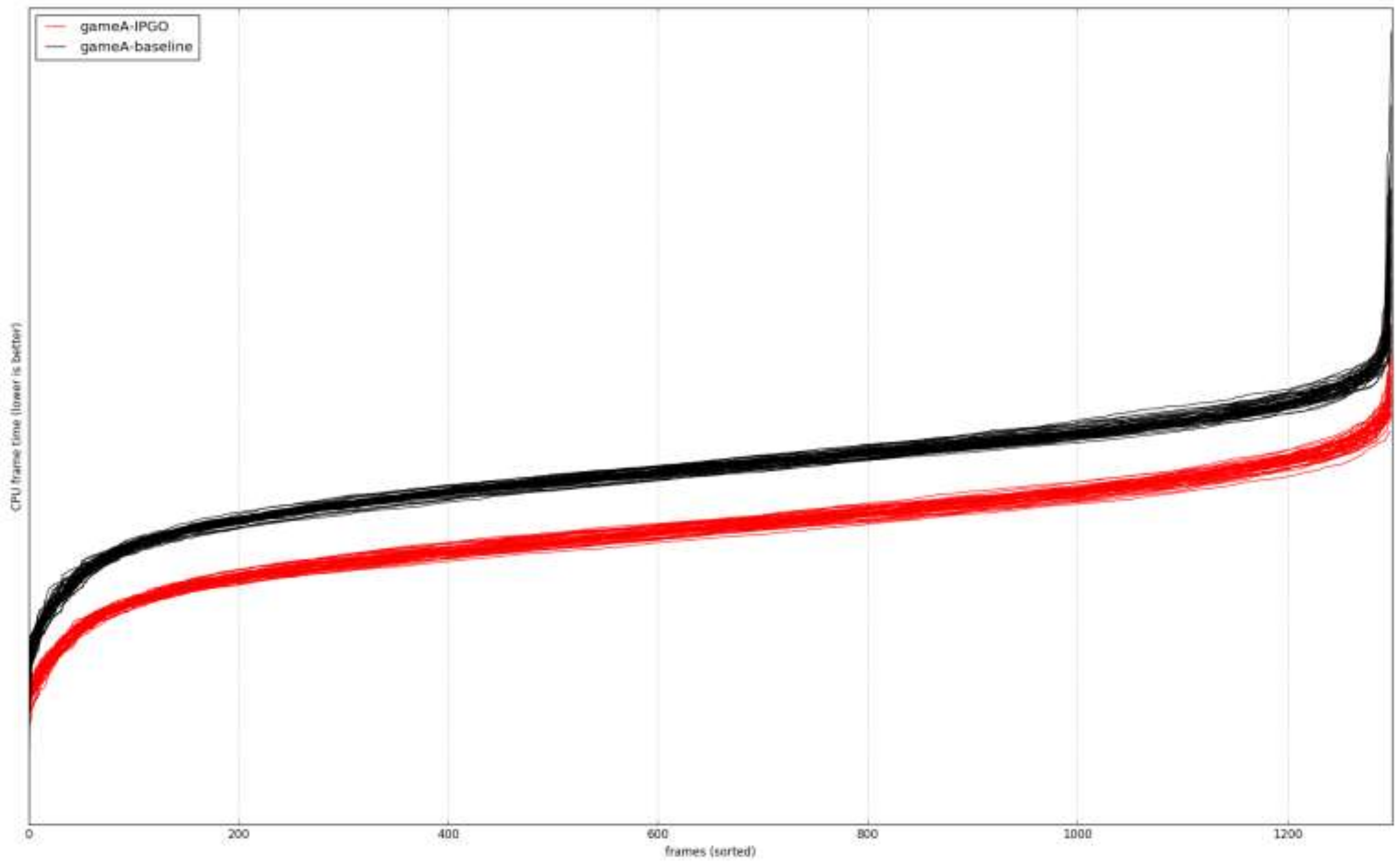
Frame time (Game A, sorted)



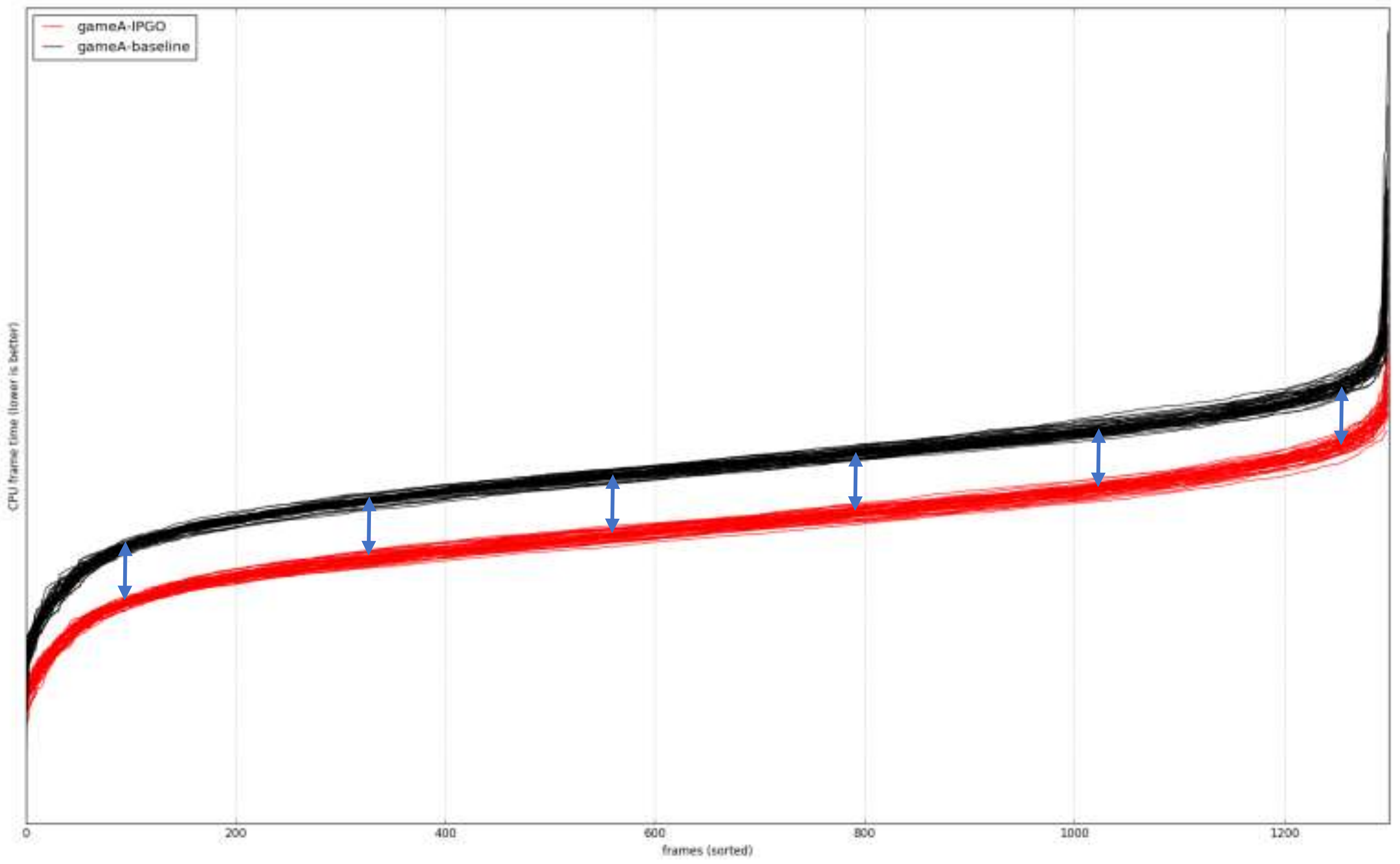
Frame time (Game A, sorted)



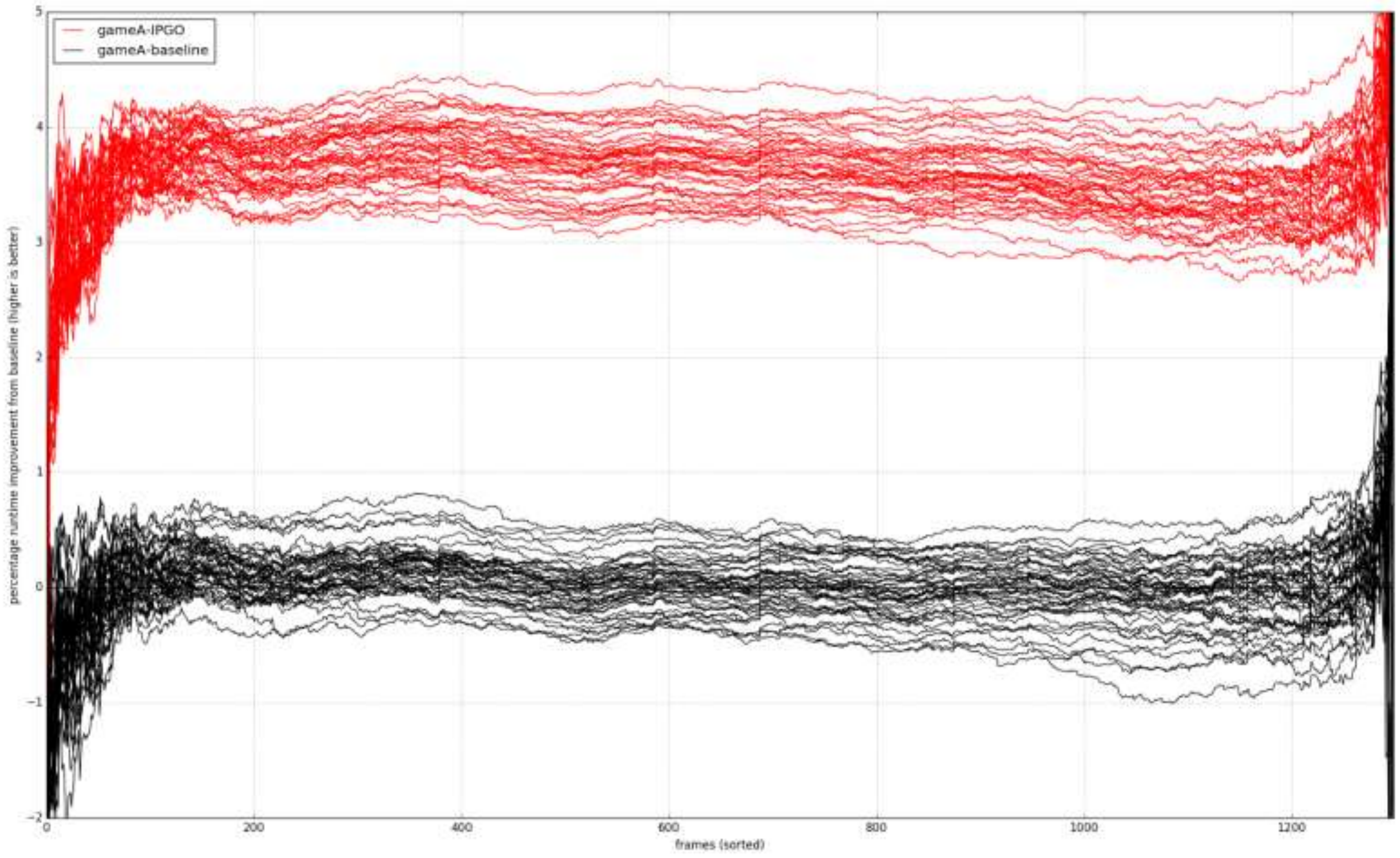
Frame time (Game A, sorted)



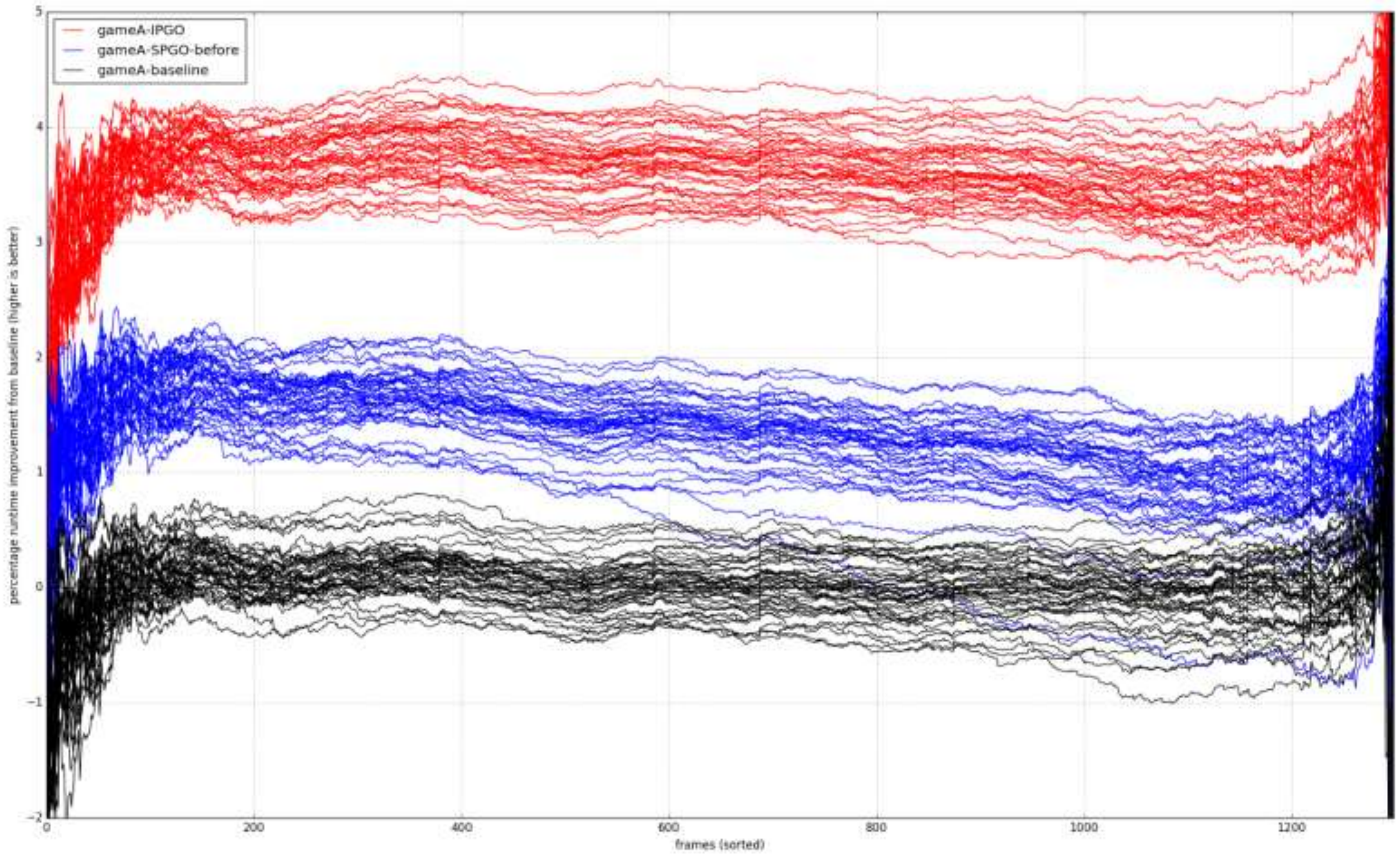
Frame time (Game A, sorted)



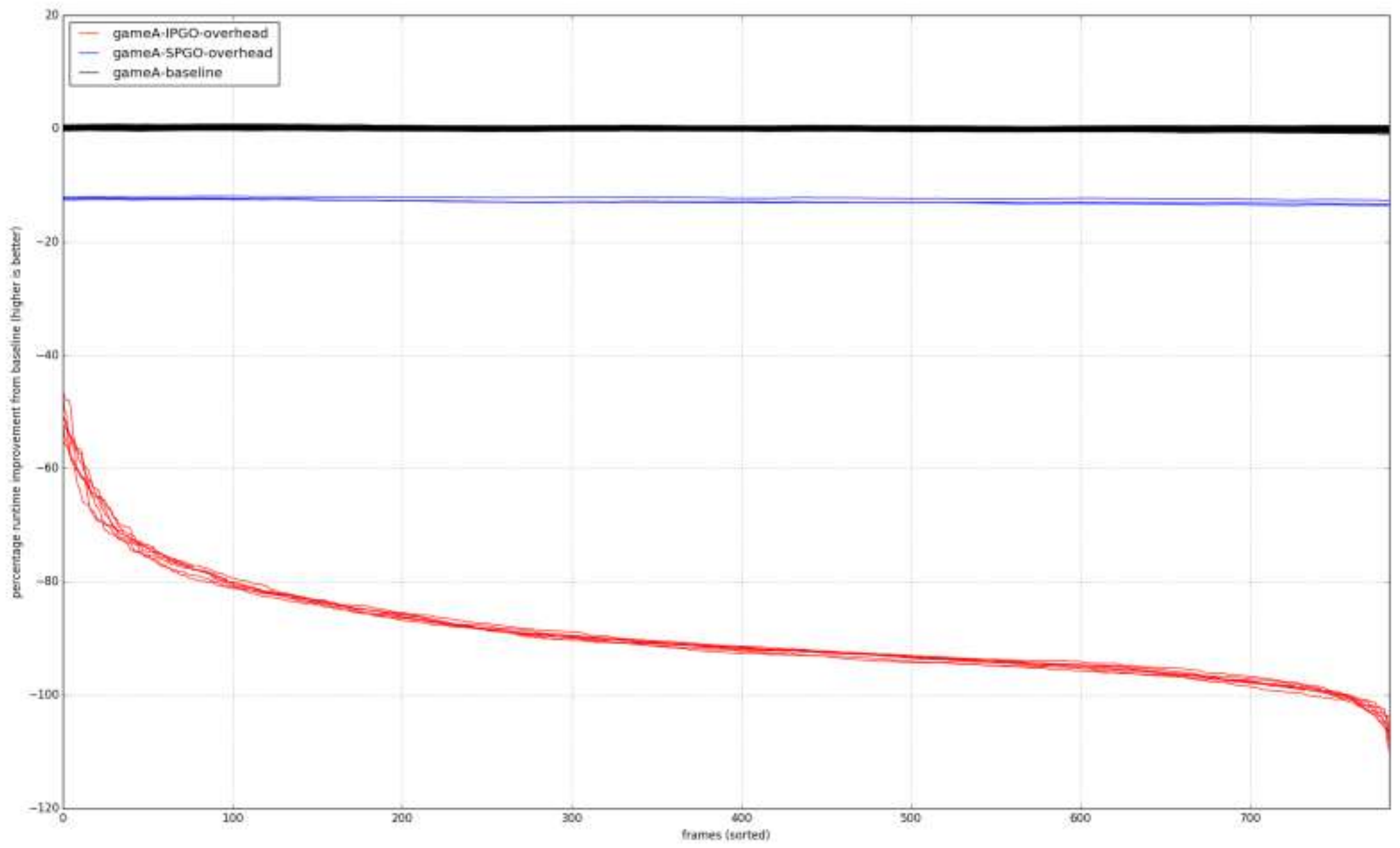
Frame time improvement (Game A)



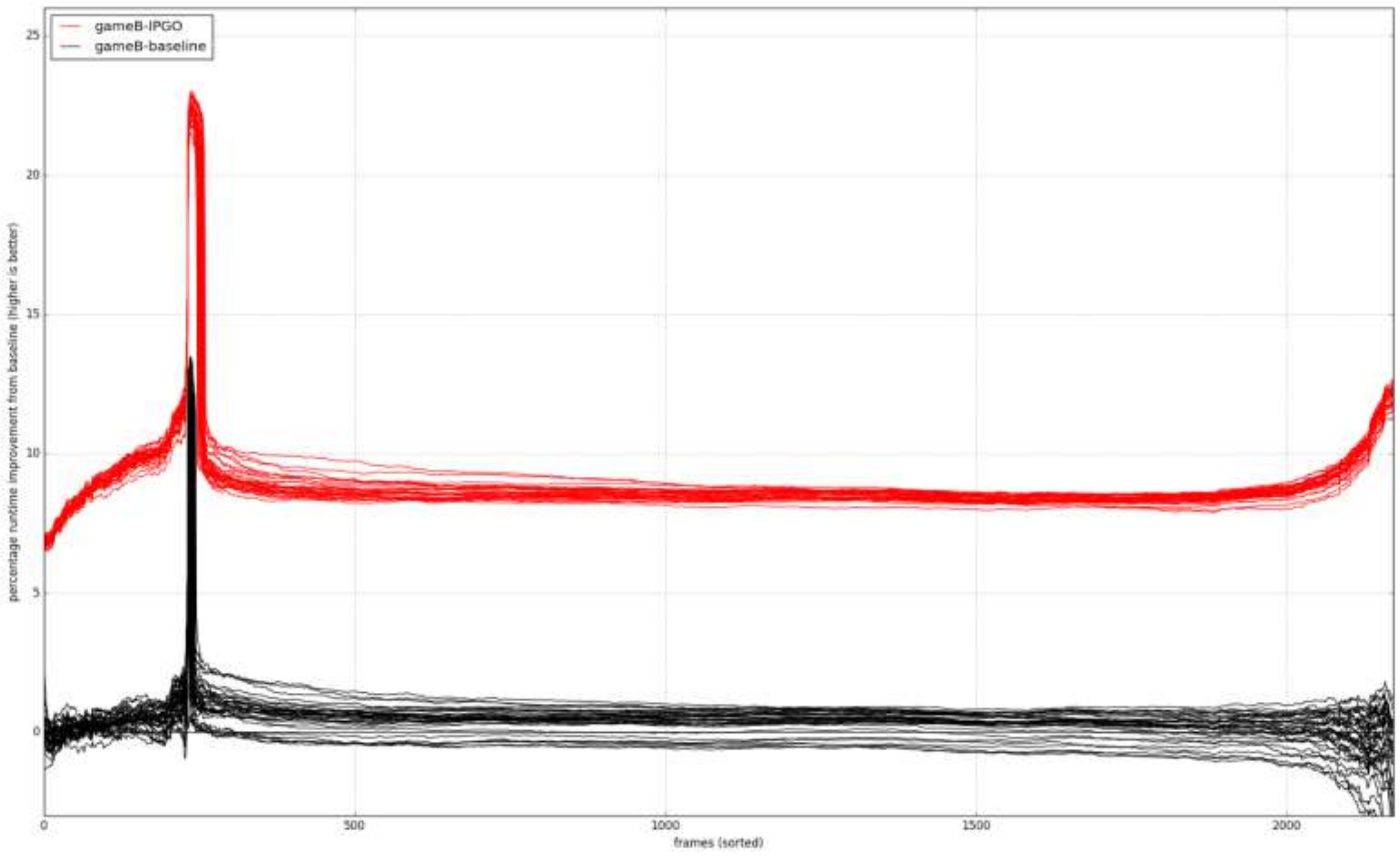
Frame time improvement (Game A)



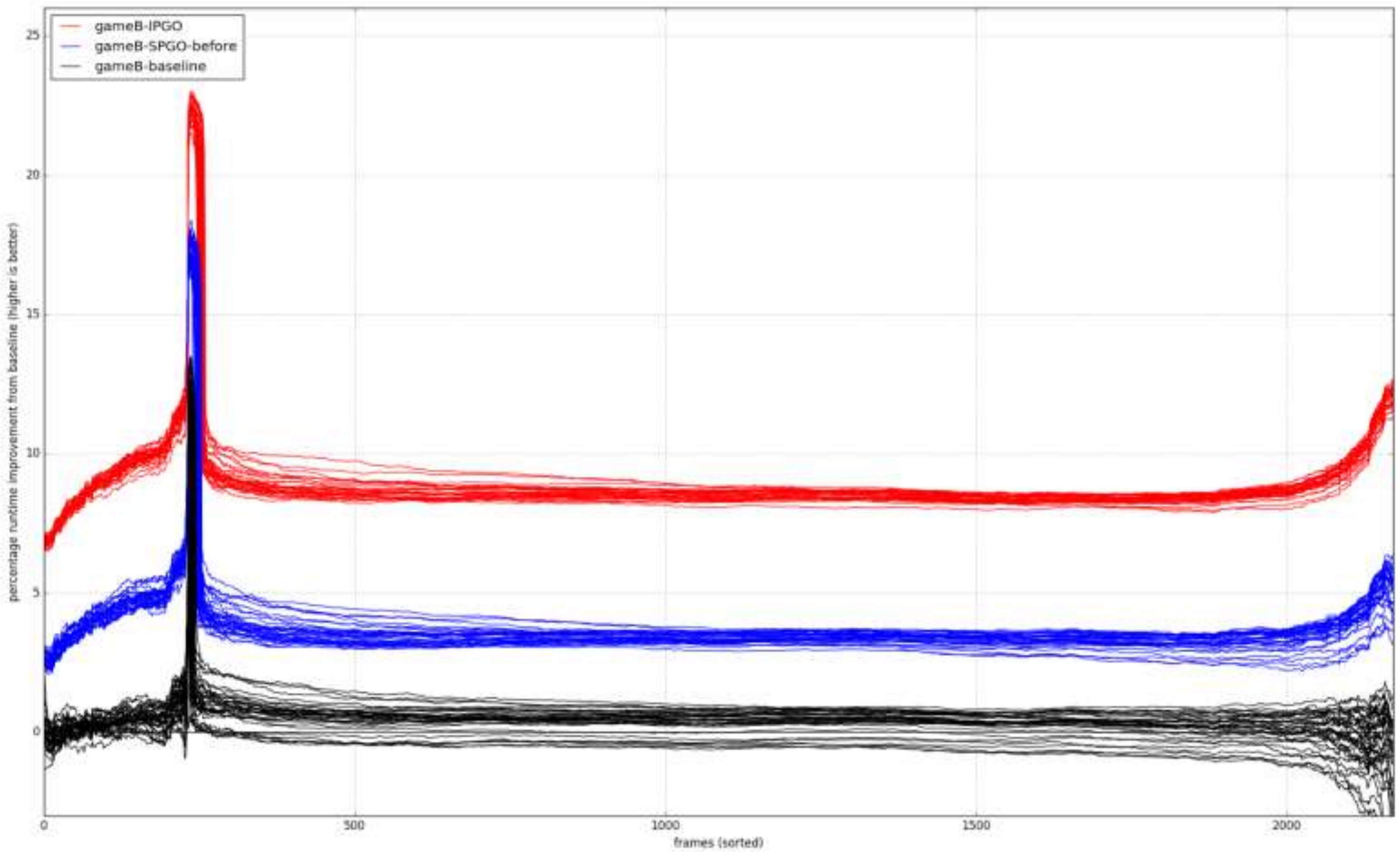
Profile generation overhead (Game A)



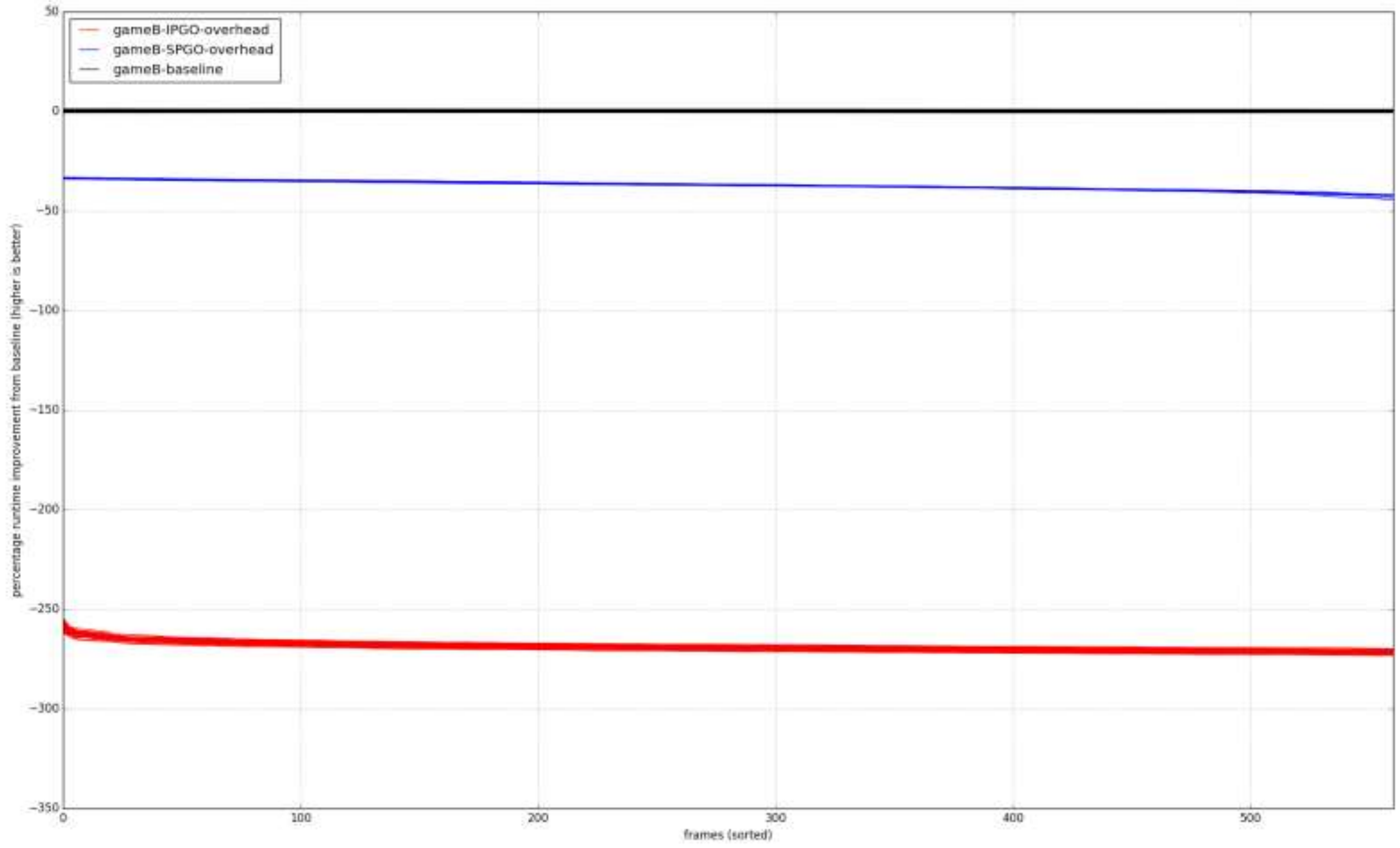
Frame time improvement (Game B)



Frame time improvement (Game B)



Profile generation overhead (Game B)



Summary of initial results

- Sample-based PGO is clear win in terms of runtime intrusion when generating the profile data

- Instrumentation-based PGO is a clear win in terms of overall runtime speed improvement when applying the profile data

What is being done differently with sample-generated profile data compared to instrumentation-generated data?

Basic assumption

- Instrumentation-based PGO has ‘perfect’ information
- If Sample-based PGO is making a different decision, assume ‘imperfect’ information
- Possibilities:
  - Not enough coverage in the raw profile data
  - Poor mapping of instructions to source lines/basic blocks
  - Destructive optimizations

Let’s start looking at the differences!

Example 1

Commoning

```c
__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

   if (condition)
     * p = uncommon(1);
   else
     * p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
}
  return x;
}
```

```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

   if (condition)
     * p = uncommon(1);
   else
     * p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
}
  return x;
}

```

-O2 -g

	Instruction
my_hot_function:	pushq %rbp
	movq %rsp, %rbp
	pushq %rbx
	pushq %rax
	movq %rsi, %rbx
	testl %edi, %edi
	je **.LBB2_2**
	movl $1, %edi
	callq uncommon@PLT
	jmp **.LBB2_3**
.LBB2_2:	movl $2, %edi
	callq common@PLT
	andl $32767, %eax
.LBB2_3:	movl %eax, (%rbx)
	addq $8, %rsp
	popq %rbx
	popq %rbp
	retq

-O2 -g
-fprofile-sample-use

```
                      Instruction
my_hot_function:      pushq  %rbp
                      movq  %rsp, %rbp
                      pushq  %rbx
                      pushq  %rax
                      movq  %rsi, %rbx
                      testl  %edi, %edi
                      je  .LBB2_2
                      movl  $1, %edi
                      callq  uncommon@PLT
                      jmp  .LBB2_3
.LBB2_2:              movl  $2, %edi
                      callq  common@PLT
                      andl  $32767, %eax
.LBB2_3:              movl  %eax, (%rbx)
                      addq  $8, %rsp
                      popq  %rbx
                      popq  %rbp
                      retq
```

Heat Map		Instruction
🟥	**my_hot_function:**	pushq %rbp
🟥		movq %rsp, %rbp
🟥		pushq %rbx
🟥		pushq %rax
🟥		movq %rsi, %rbx
🟥		testl %edi, %edi
🟥		je **.LBB2_2**
▨		movl $1, %edi
▨		callq uncommon@PLT
▨		jmp **.LBB2_3**
🟥	**.LBB2_2:**	movl $2, %edi
🟥		callq common@PLT
🟥		andl $32767, %eax
🟥	**.LBB2_3:**	movl %eax, (%rbx)
🟥		addq $8, %rsp
🟥		popq %rbx
🟥		popq %rbp
🟥		retq

-O2 -g

```
Function: main: 2027237, 0, 6 sampled lines
Samples collected in the function's body {
  0: 0
  1: 0
  2.1: 39959
  2.3: 39933
  3: 43747, calls: my_hot_function:43747
  5: 0
}
No inlined callsites in this function
Function: my_hot_function: 1472441, 44530, 5 sampled lines
Samples collected in the function's body {
  0: 45081
  2: 45728
  3: 46922, calls: uncommon:48
  5: 46871, calls: common:46654
  6: 47252
}
No inlined callsites in this function
Function: common: 188824, 47206, 1 sampled lines
Samples collected in the function's body {
  0: 47206
```

```
10  void my_hot_function(int condition, int *p) {
11
12    if (condition)
13      *p = uncommon(1);
14    else
15      *p = common(2) & 0x7fff;
16  }
```

```
void my_hot_function(int condition, int *p) {   45081

  if (condition)                                45728
    *p = uncommon(1);                           46922, calls: uncommon:48
  else
    *p = common(2) & 0x7fff;                    46871, calls: common:46654
}                                               47252
```

```
void my_hot_function(int condition, int *p) {    45081

  if (condition)                                  45728
    *p = uncommon(1);                             46922, calls: uncommon:48
  else
    *p = common(2) & 0x7fff;                      46871, calls: common:46654
}                                                 47252
```

```
void my_hot_function(int condition, int *p) {   45081

  if (condition)                                 45728
    *p = uncommon(1);                            46922, calls: uncommon:48
  else
    *p = common(2) & 0x7fff;                     46871, calls: common:46654
}                                                47252
```

Heat Map		Instruction
🟥	**my_hot_function:**	pushq %rbp
🟥		movq %rsp, %rbp
🟥		pushq %rbx
🟥		pushq %rax
🟥		movq %rsi, %rbx
🟥		testl %edi, %edi
🟥		je **.LBB2_2**
▨		movl $1, %edi
▨		callq uncommon@PLT
▨		jmp **.LBB2_3**
🟥	**.LBB2_2:**	movl $2, %edi
🟥		callq common@PLT
🟥		andl $32767, %eax
🟥	**.LBB2_3:**	movl %eax, (%rbx)
🟥		addq $8, %rsp
🟥		popq %rbx
🟥		popq %rbp
🟥		retq

-O2 -g

Heat Map		Instruction	
🟥	**my_hot_function:**	`pushq %rbp`	10:0
🟥		`movq %rsp, %rbp`	
🟥		`pushq %rbx`	
🟥		`pushq %rax`	
🟥		`movq %rsi, %rbx`	
🟥		`testl %edi, %edi`	12:0
🟥		`je` **.LBB2_2**	
▨		`movl $1, %edi`	13:0
▨		`callq uncommon@PLT`	
▨		`jmp` **.LBB2_3**	
🟥	**.LBB2_2:**	`movl $2, %edi`	15:0
🟥		`callq common@PLT`	
🟥		`andl $32767, %eax`	
🟥	**.LBB2_3:**	`movl %eax, (%rbx)`	13:0
🟥		`addq $8, %rsp`	16:0
🟥		`popq %rbx`	
🟥		`popq %rbp`	
🟥		`retq`	

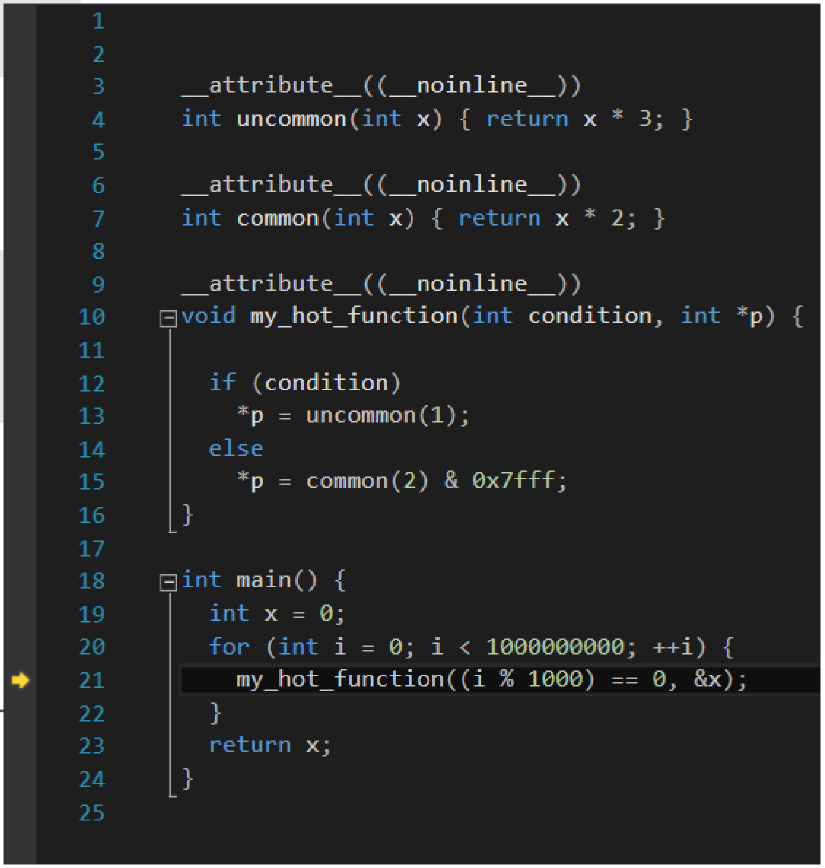
-O2 -g

-O2 -g

Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▫		movl $1, %edi	13:0
▫		callq uncommon@PLT	
▫		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	

-O2 -g

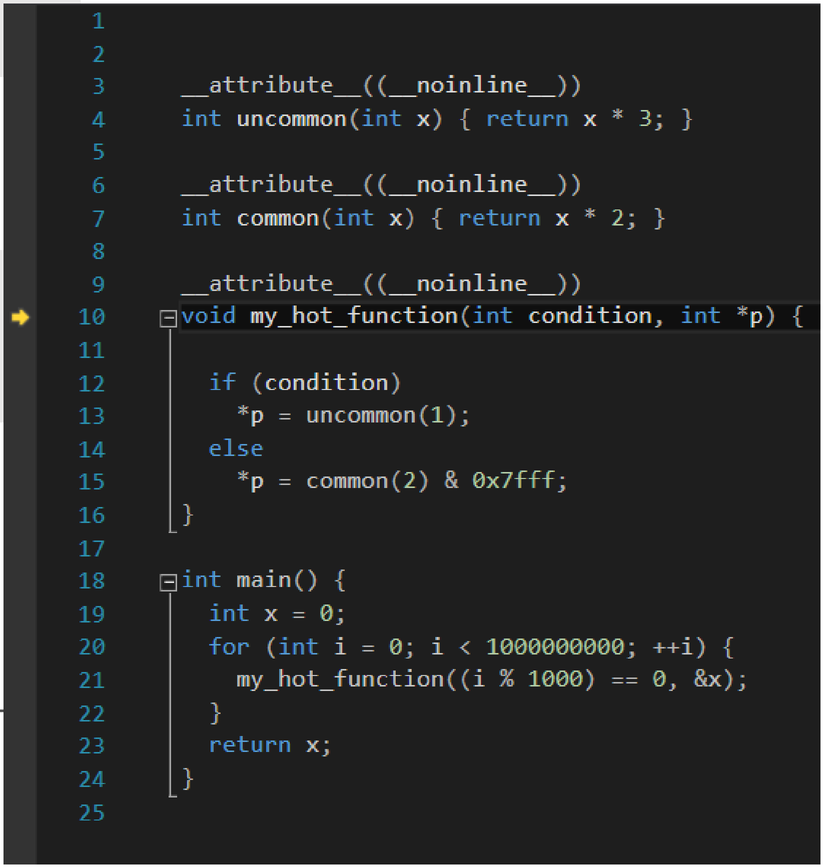
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
░		movl $1, %edi	13:0
░		callq uncommon@PLT	
░		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

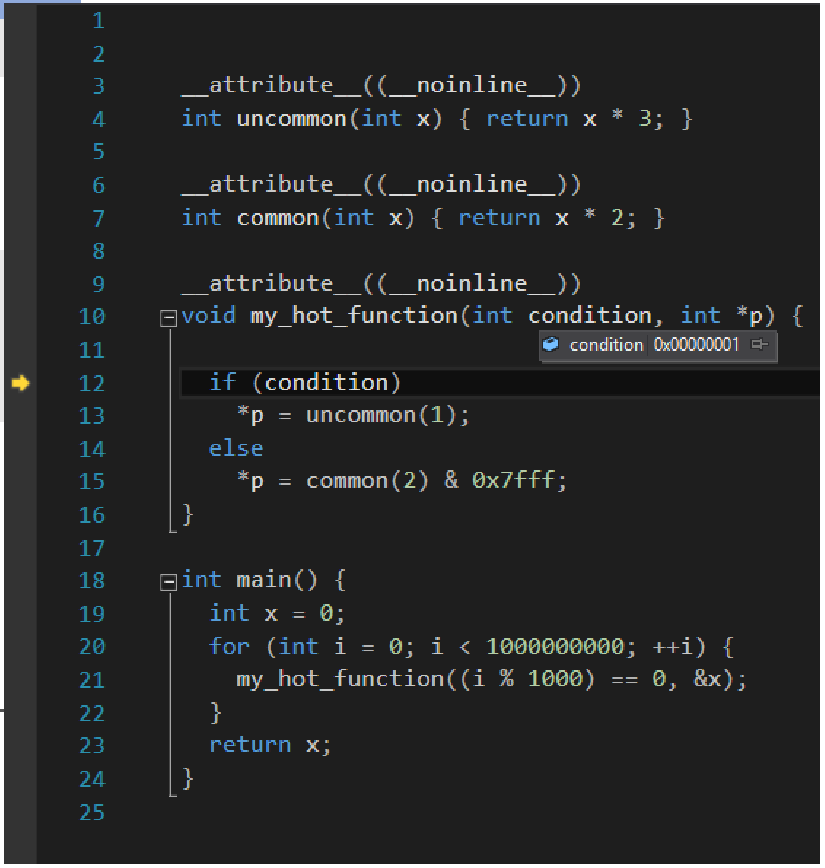
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
1
2
3     __attribute__((__noinline__))
4     int uncommon(int x) { return x * 3; }
5
6     __attribute__((__noinline__))
7     int common(int x) { return x * 2; }
8
9     __attribute__((__noinline__))
10    void my_hot_function(int condition, int *p) {
11
12      if (condition)
13        *p = uncommon(1);
14      else
15        *p = common(2) & 0x7fff;
16    }
17
18    int main() {
19      int x = 0;
20      for (int i = 0; i < 1000000000; ++i) {
21        my_hot_function((i % 1000) == 0, &x);
22      }
23      return x;
24    }
25
```

-O2 -g

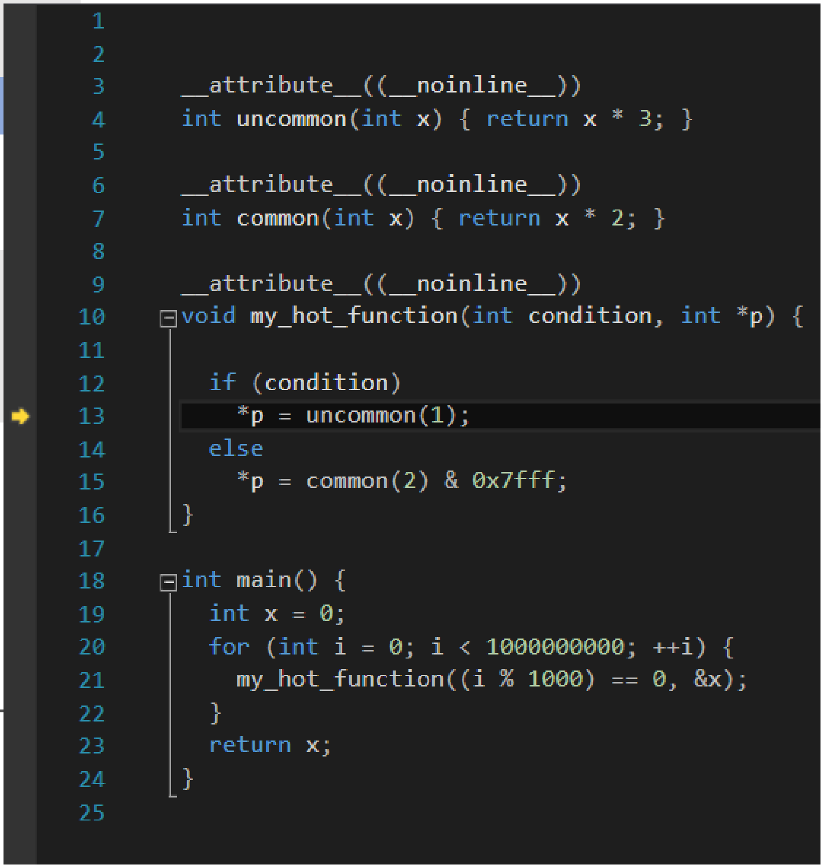
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
		movl $1, %edi	13:0
		callq uncommon@PLT	
		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```

__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {
                                    condition 0x00000001
  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

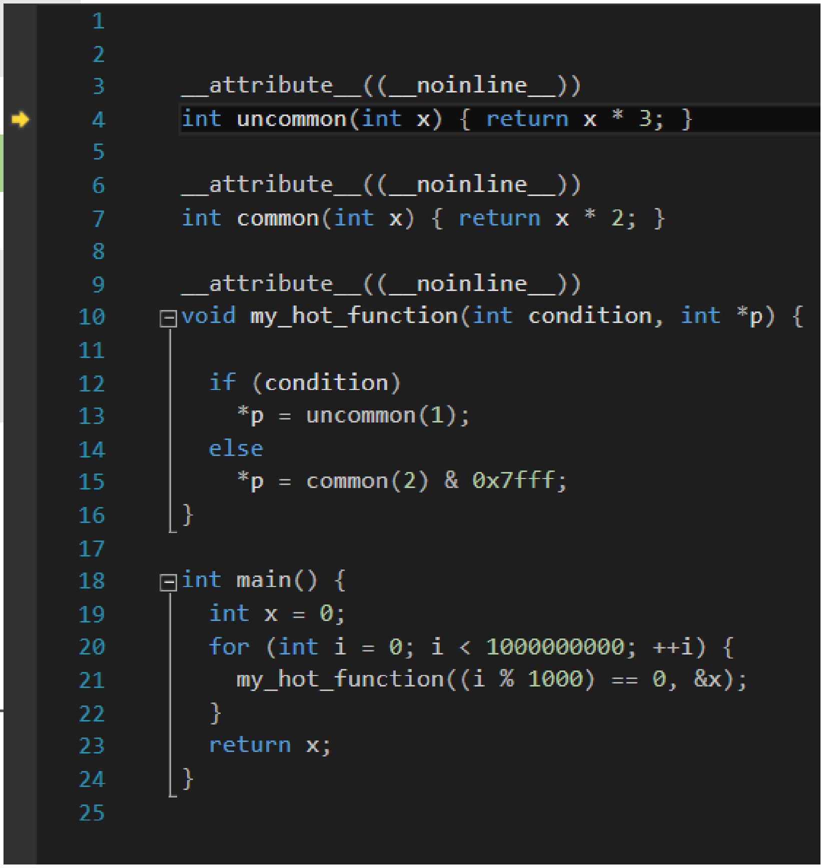
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
1
2
3   __attribute__((__noinline__))
4   int uncommon(int x) { return x * 3; }
5
6   __attribute__((__noinline__))
7   int common(int x) { return x * 2; }
8
9   __attribute__((__noinline__))
10  void my_hot_function(int condition, int *p) {
11
12    if (condition)
13      *p = uncommon(1);
14    else
15      *p = common(2) & 0x7fff;
16  }
17
18  int main() {
19    int x = 0;
20    for (int i = 0; i < 1000000000; ++i) {
21      my_hot_function((i % 1000) == 0, &x);
22    }
23    return x;
24  }
25
```

-O2 -g

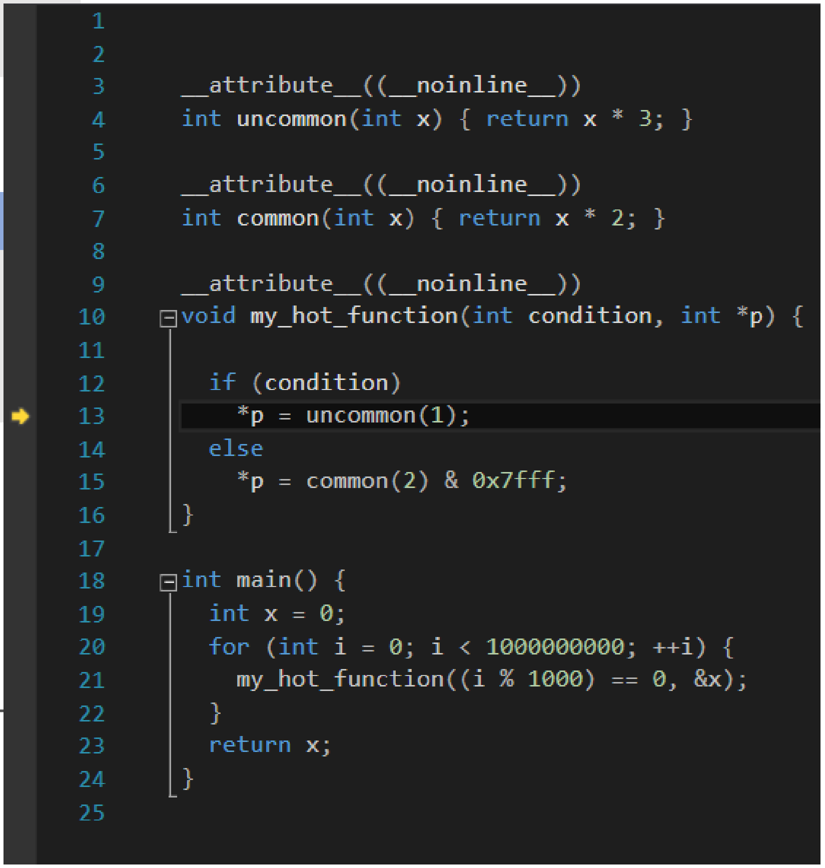
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	

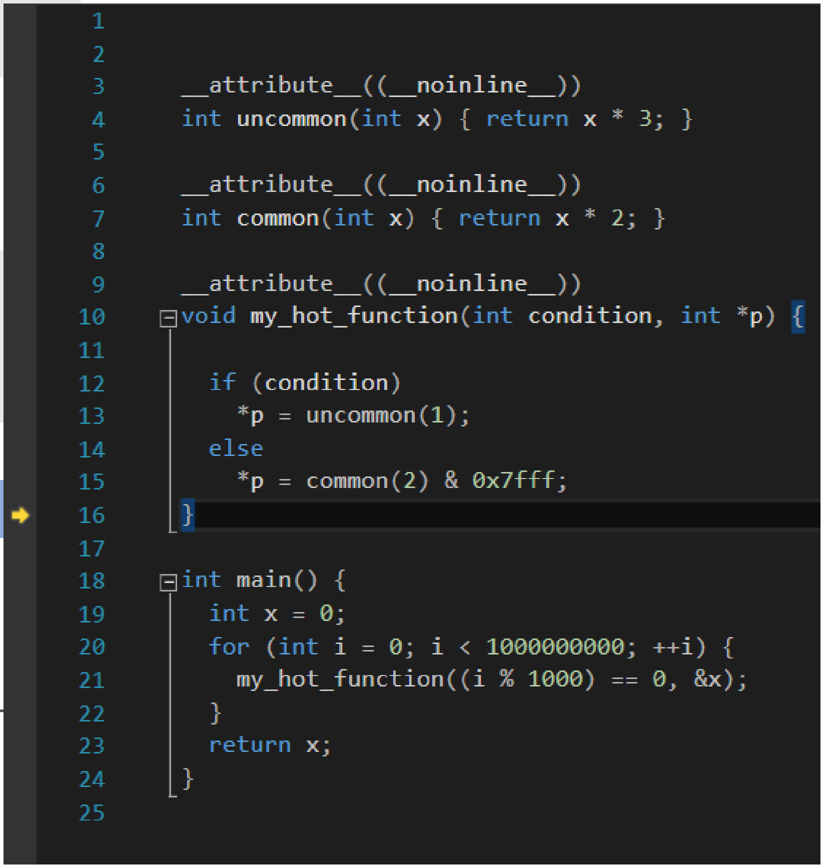
-O2 -g



```
 1
 2
 3  __attribute__((__noinline__))
 4  int uncommon(int x) { return x * 3; }
 5
 6  __attribute__((__noinline__))
 7  int common(int x) { return x * 2; }
 8
 9  __attribute__((__noinline__))
10  void my_hot_function(int condition, int *p) {
11
12    if (condition)
13      *p = uncommon(1);
14    else
15      *p = common(2) & 0x7fff;
16  }
17
18  int main() {
19    int x = 0;
20    for (int i = 0; i < 1000000000; ++i) {
21      my_hot_function((i % 1000) == 0, &x);
22    }
23    return x;
24  }
25
```

-O2 -g

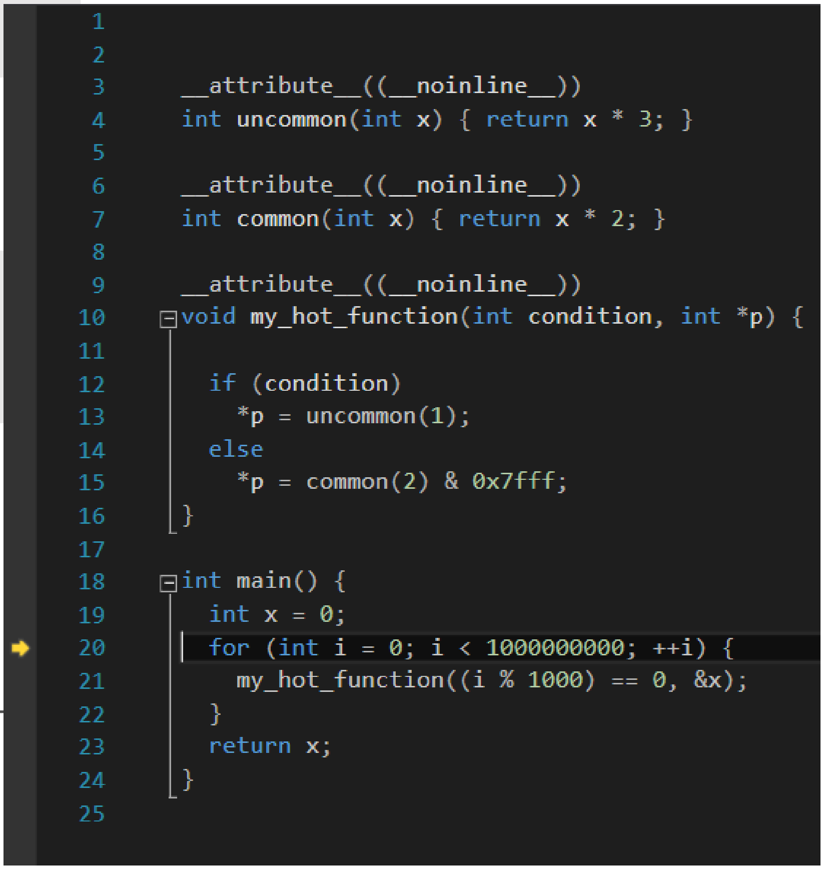
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
1
2
3   __attribute__((__noinline__))
4   int uncommon(int x) { return x * 3; }
5
6   __attribute__((__noinline__))
7   int common(int x) { return x * 2; }
8
9   __attribute__((__noinline__))
10  void my_hot_function(int condition, int *p) {
11
12    if (condition)
13      *p = uncommon(1);
14    else
15      *p = common(2) & 0x7fff;
16  }
17
18  int main() {
19    int x = 0;
20    for (int i = 0; i < 1000000000; ++i) {
21      my_hot_function((i % 1000) == 0, &x);
22    }
23    return x;
24  }
25
```

-O2 -g

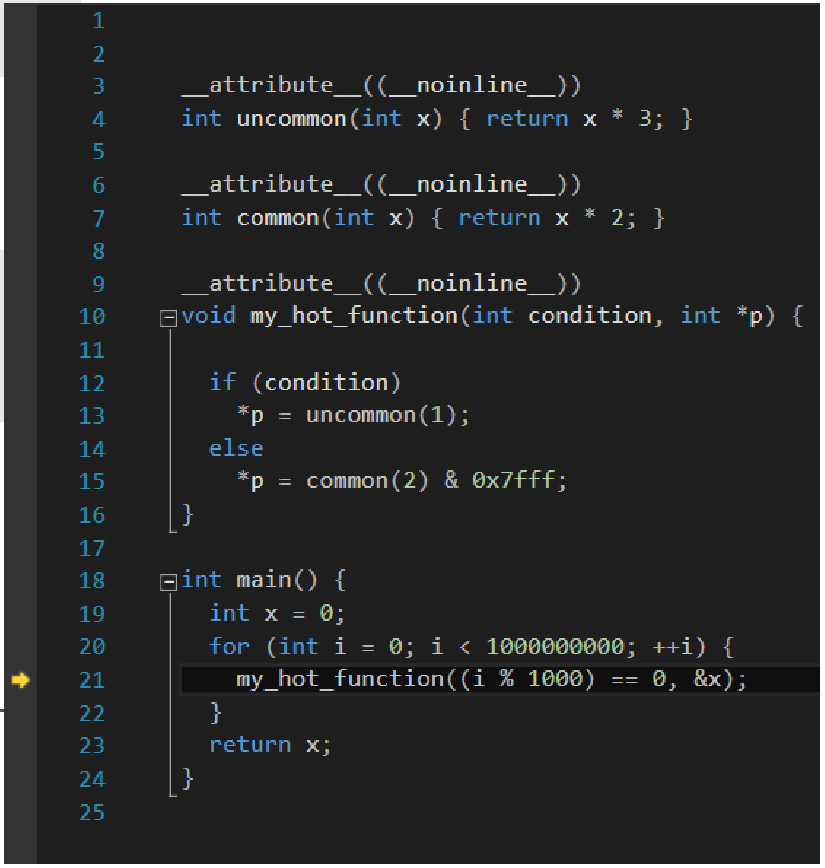
Heat Map		Instruction		
🟥	**my_hot_function:**	pushq	%rbp	10:0
🟥		movq	%rsp, %rbp	
🟥		pushq	%rbx	
🟥		pushq	%rax	
🟥		movq	%rsi, %rbx	
🟥		testl	%edi, %edi	12:0
🟥		je	**.LBB2_2**	
▨		movl	$1, %edi	13:0
▨		callq	uncommon@PLT	
▨		jmp	**.LBB2_3**	
🟥	**.LBB2_2:**	movl	$2, %edi	15:0
🟥		callq	common@PLT	
🟥		andl	$32767, %eax	
🟥	**.LBB2_3:**	movl	%eax, (%rbx)	13:0
🟥		addq	$8, %rsp	16:0
🟥		popq	%rbx	
🟥		popq	%rbp	
🟥		retq		



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

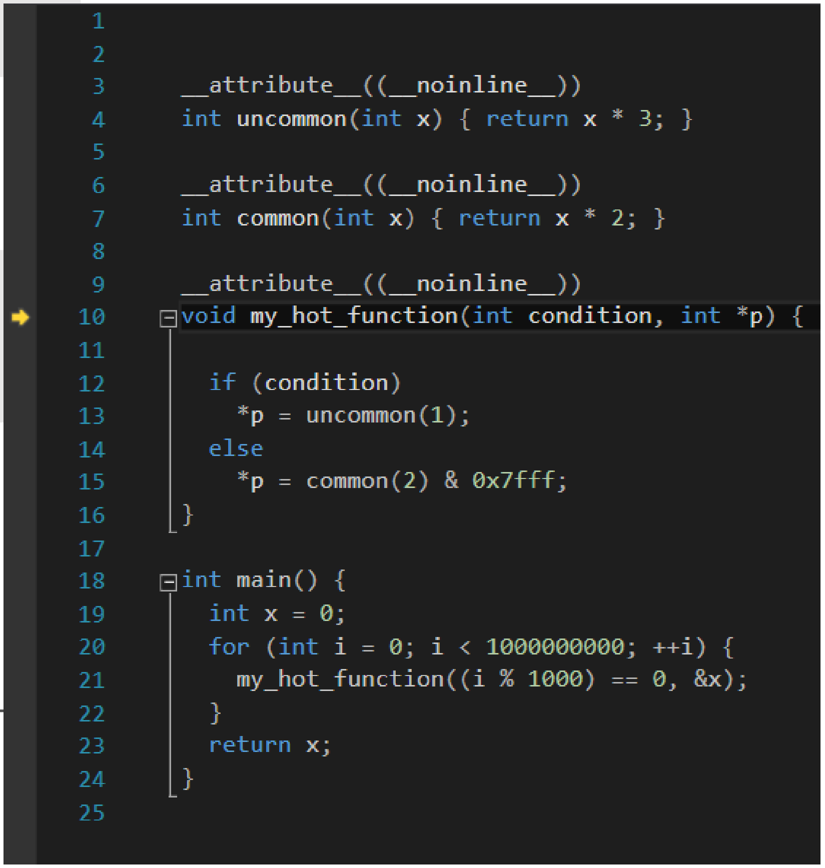
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

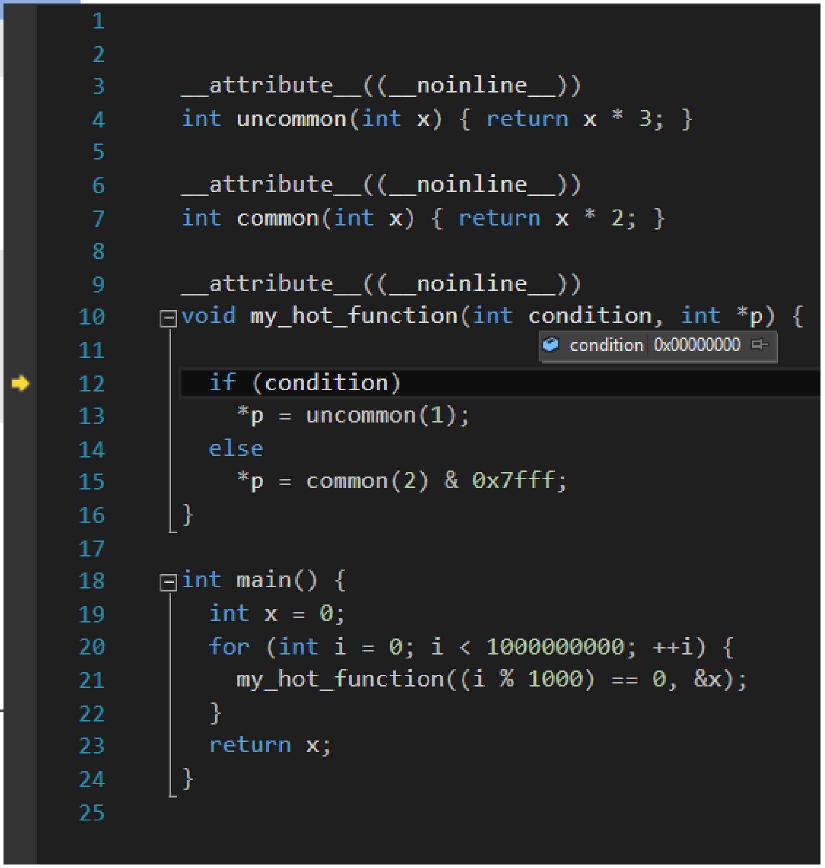
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
1
2
3    __attribute__((__noinline__))
4    int uncommon(int x) { return x * 3; }
5
6    __attribute__((__noinline__))
7    int common(int x) { return x * 2; }
8
9    __attribute__((__noinline__))
10   void my_hot_function(int condition, int *p) {
11
12     if (condition)
13       *p = uncommon(1);
14     else
15       *p = common(2) & 0x7fff;
16   }
17
18   int main() {
19     int x = 0;
20     for (int i = 0; i < 1000000000; ++i) {
21       my_hot_function((i % 1000) == 0, &x);
22     }
23     return x;
24   }
25
```

-O2 -g

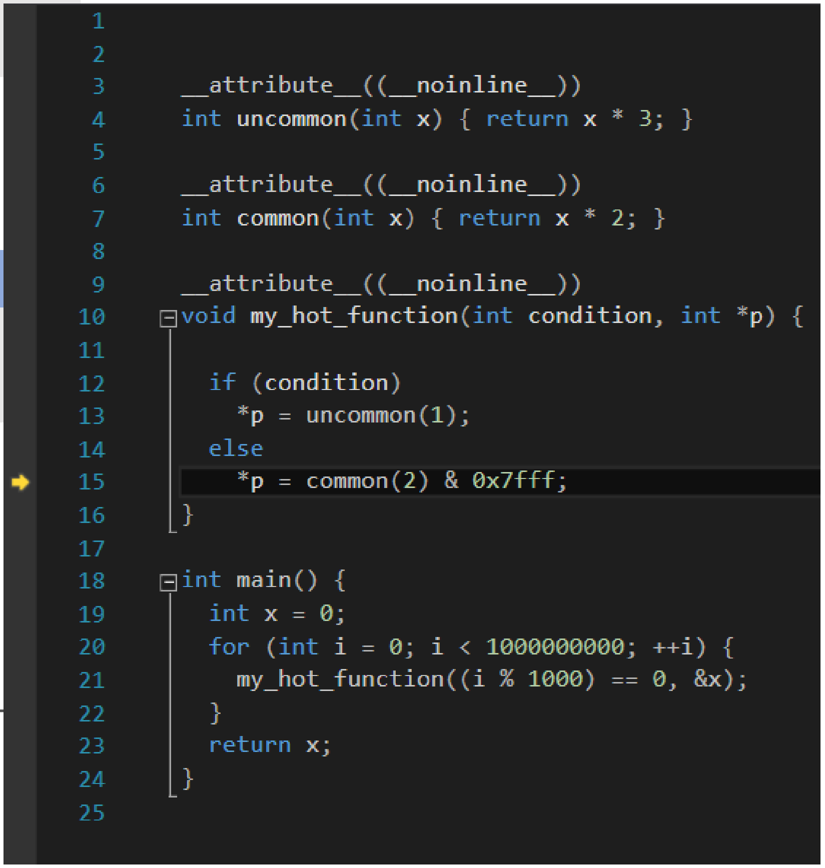
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
1
2
3    __attribute__((__noinline__))
4    int uncommon(int x) { return x * 3; }
5
6    __attribute__((__noinline__))
7    int common(int x) { return x * 2; }
8
9    __attribute__((__noinline__))
10   void my_hot_function(int condition, int *p) {
11                                    condition 0x00000000
12     if (condition)
13       *p = uncommon(1);
14     else
15       *p = common(2) & 0x7fff;
16   }
17
18   int main() {
19     int x = 0;
20     for (int i = 0; i < 1000000000; ++i) {
21       my_hot_function((i % 1000) == 0, &x);
22     }
23     return x;
24   }
25
```

-O2 -g

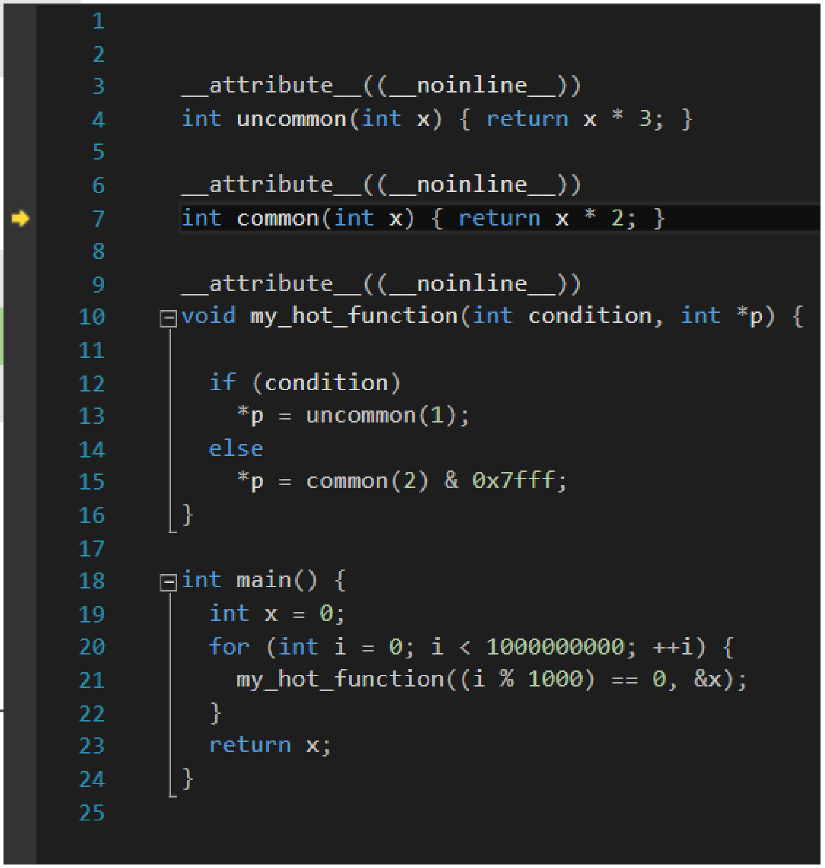
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```c


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

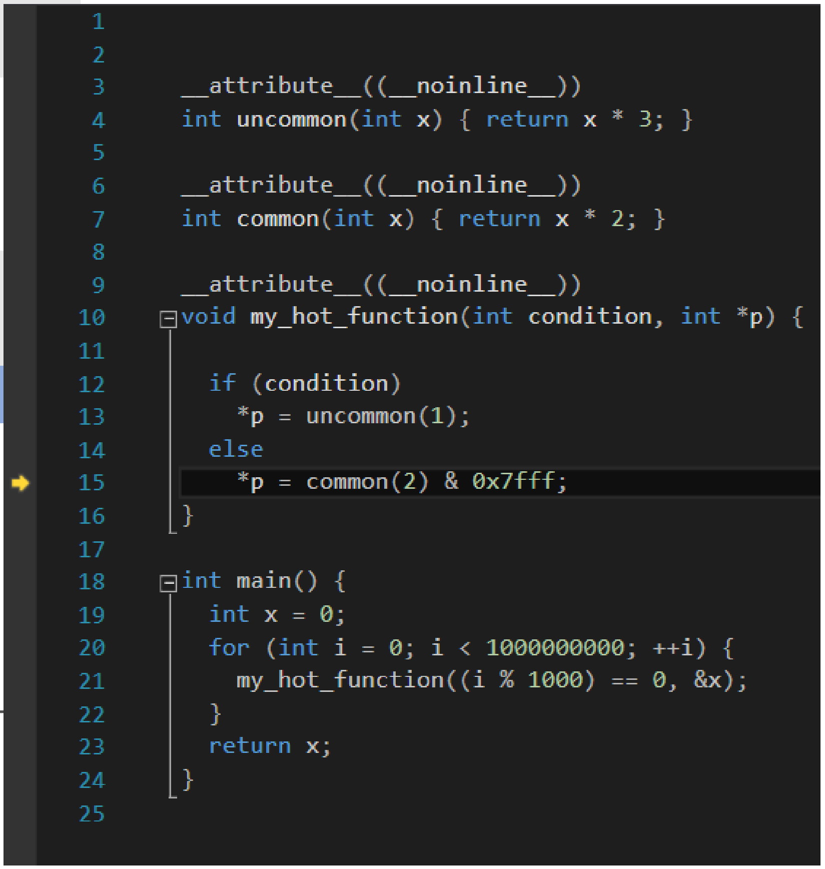
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}
```

-O2 -g

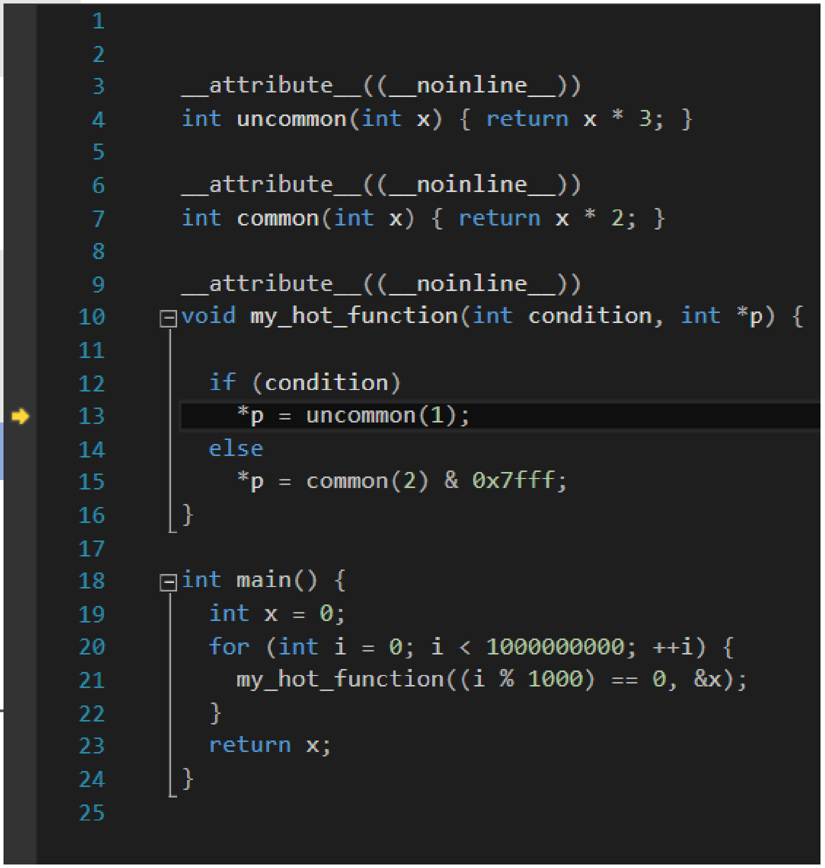
Heat Map		Instruction	
	my_hot_function:	pushq %rbp	10:0
		movq %rsp, %rbp	
		pushq %rbx	
		pushq %rax	
		movq %rsi, %rbx	
		testl %edi, %edi	12:0
		je .LBB2_2	
		movl $1, %edi	13:0
		callq uncommon@PLT	
		jmp .LBB2_3	
	.LBB2_2:	movl $2, %edi	15:0
		callq common@PLT	
		andl $32767, %eax	
	.LBB2_3:	movl %eax, (%rbx)	13:0
		addq $8, %rsp	16:0
		popq %rbx	
		popq %rbp	
		retq	



```
1
2
3    __attribute__((__noinline__))
4    int uncommon(int x) { return x * 3; }
5
6    __attribute__((__noinline__))
7    int common(int x) { return x * 2; }
8
9    __attribute__((__noinline__))
10   void my_hot_function(int condition, int *p) {
11
12     if (condition)
13       *p = uncommon(1);
14     else
15       *p = common(2) & 0x7fff;
16   }
17
18   int main() {
19     int x = 0;
20     for (int i = 0; i < 1000000000; ++i) {
21       my_hot_function((i % 1000) == 0, &x);
22     }
23     return x;
24   }
25
```

-O2 -g

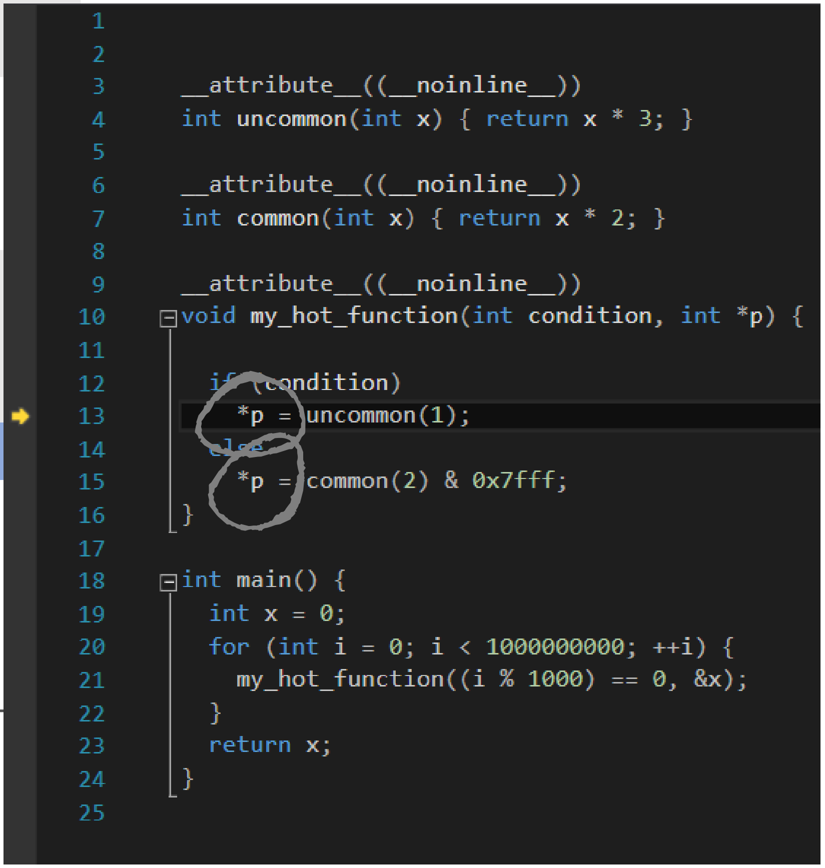
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
1
2
3    __attribute__((__noinline__))
4    int uncommon(int x) { return x * 3; }
5
6    __attribute__((__noinline__))
7    int common(int x) { return x * 2; }
8
9    __attribute__((__noinline__))
10   void my_hot_function(int condition, int *p) {
11
12     if (condition)
13       *p = uncommon(1);
14     else
15       *p = common(2) & 0x7fff;
16   }
17
18   int main() {
19     int x = 0;
20     for (int i = 0; i < 1000000000; ++i) {
21       my_hot_function((i % 1000) == 0, &x);
22     }
23     return x;
24   }
25
```

-O2 -g

Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```
1
2
3    __attribute__((__noinline__))
4    int uncommon(int x) { return x * 3; }
5
6    __attribute__((__noinline__))
7    int common(int x) { return x * 2; }
8
9    __attribute__((__noinline__))
10   void my_hot_function(int condition, int *p) {
11
12     if (condition)
13       *p = uncommon(1);
14     else
15       *p = common(2) & 0x7fff;
16   }
17
18   int main() {
19     int x = 0;
20     for (int i = 0; i < 1000000000; ++i) {
21       my_hot_function((i % 1000) == 0, &x);
22     }
23     return x;
24   }
25
```

```
BBI = DestBB->getFirstInsertionPt();
StoreInst *NewSI = new StoreInst(MergedVal, SI.getOperand(1),
         SI.isVolatile(),
         SI.getAlignment(),
         SI.getOrdering(),
         SI.getSynchScope());
InsertNewInstBefore(NewSI, *BBI);
NewSI->setDebugLoc(OtherStore->getDebugLoc());

// If the two stores had AA tags, merge them.
AAMDNodes AATags;
SI.getAAMetadata(AATags);
if (AATags) {
         OtherStore->getAAMetadata(AATags, /* Merge = */ true);
         NewSI->setAAMetadata(AATags);
}
```

```
BBI = DestBB->getFirstInsertionPt();
StoreInst *NewSI = new StoreInst(MergedVal, SI.getOperand(1),
         SI.isVolatile(),
         SI.getAlignment(),
         SI.getOrdering(),
         SI.getSynchScope());
InsertNewInstBefore(NewSI, *BBI);
NewSI->setDebugLoc(SI.getDebugLoc());

// If the two stores had AA tags, merge them.
AAMDNodes AATags;
SI.getAAMetadata(AATags);
if (AATags) {
         OtherStore->getAAMetadata(AATags, /* Merge = */ true);
         NewSI->setAAMetadata(AATags);
}
```

“An unsigned integer indicating a source line number. Lines are numbered beginning at 1. The compiler may emit the value 0 in cases where an instruction cannot be attributed to any source line.”

6.2.2 State Machine Registers
DWARF Debugging Information Format
Version 4
http://dwarfstd.org/doc/DWARF4.pdf

*“An unsigned integer indicating a source line number. Lines are numbered beginning at 1. **The compiler may emit the value 0 in cases where an instruction cannot be attributed to any source line.**”*

6.2.2 State Machine Registers
DWARF Debugging Information Format
Version 4
http://dwarfstd.org/doc/DWARF4.pdf

```
BBI = DestBB->getFirstInsertionPt();
StoreInst *NewSI = new StoreInst(MergedVal, SI.getOperand(1),
         SI.isVolatile(),
         SI.getAlignment(),
         SI.getOrdering(),
         SI.getSynchScope());
InsertNewInstBefore(NewSI, *BBI);
NewSI->setDebugLoc(OtherStore->getDebugLoc());

// If the two stores had AA tags, merge them.
AAMDNodes AATags;
SI.getAAMetadata(AATags);
if (AATags) {
         OtherStore->getAAMetadata(AATags, /* Merge = */ true);
         NewSI->setAAMetadata(AATags);
}
```

```
BBI = DestBB->getFirstInsertionPt();
StoreInst *NewSI = new StoreInst(MergedVal, SI.getOperand(1),
         SI.isVolatile(),
         SI.getAlignment(),
         SI.getOrdering(),
         SI.getSynchScope());
InsertNewInstBefore(NewSI, *BBI);
NewSI->setDebugLoc(DebugLoc());

// If the two stores had AA tags, merge them.
AAMDNodes AATags;
SI.getAAMetadata(AATags);
if (AATags) {
         OtherStore->getAAMetadata(AATags, /* Merge = */ true);
         NewSI->setAAMetadata(AATags);
}
```

-O2 -g

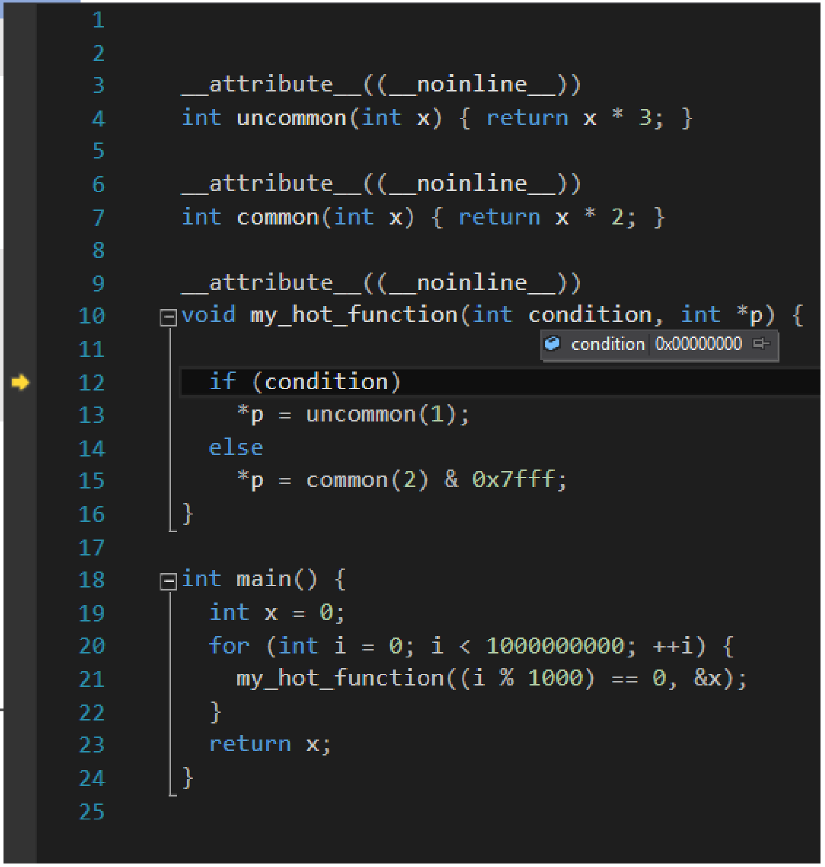
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▫		movl $1, %edi	13:0
▫		callq uncommon@PLT	
▫		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	13:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	

-O2 -g

Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp **.LBB2_3**	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	0:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	

-O2 -g

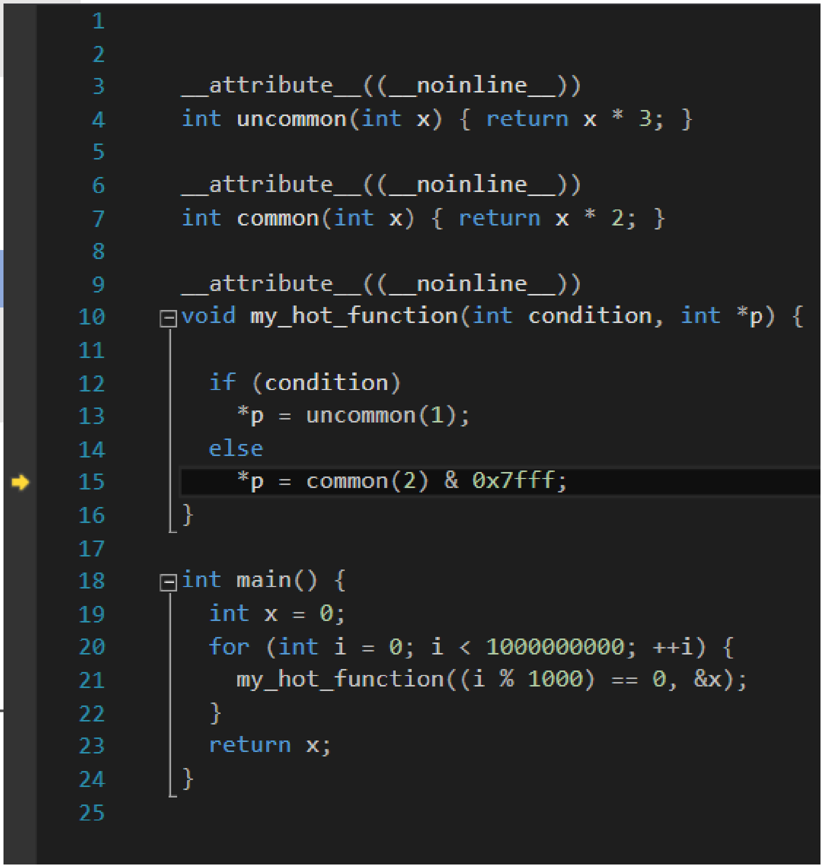
Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	0:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

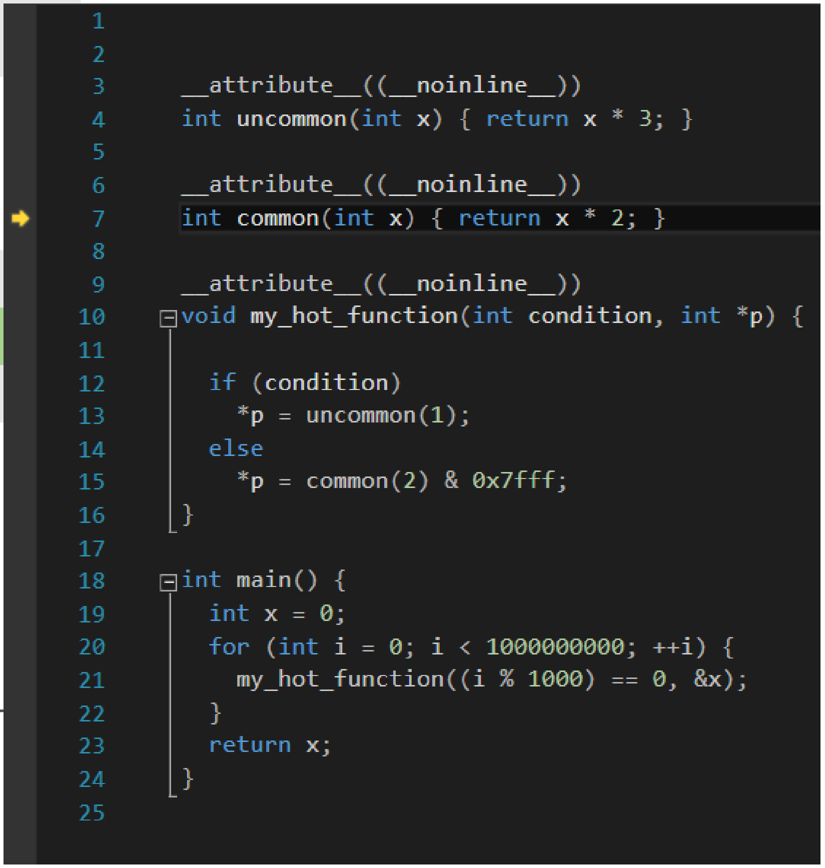
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
		movl $1, %edi	13:0
		callq uncommon@PLT	
		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	0:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

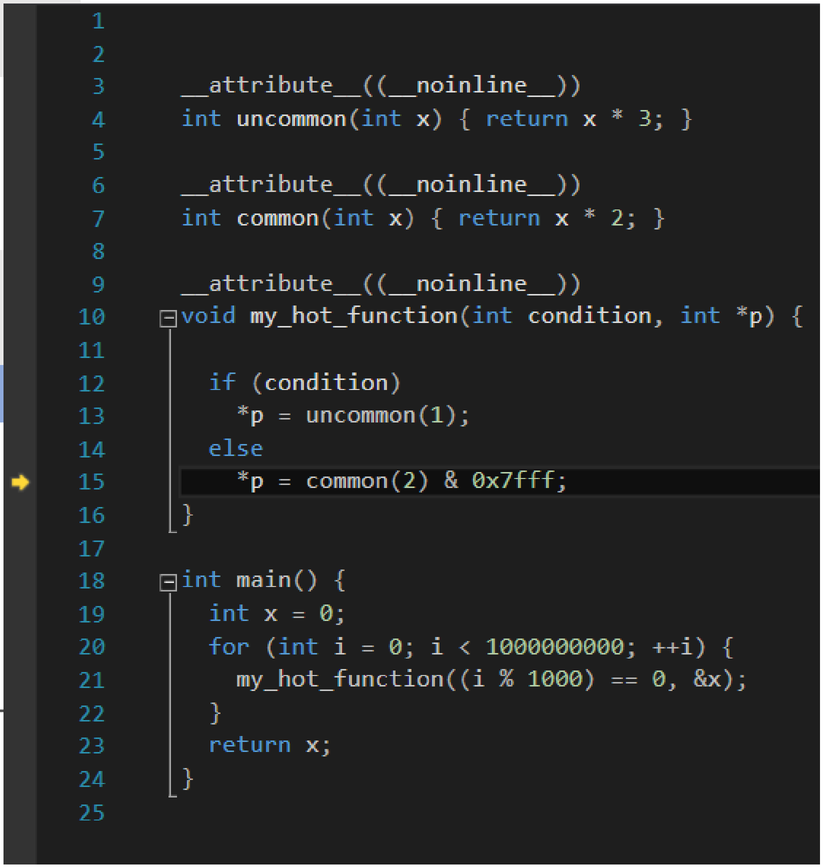
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	0:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

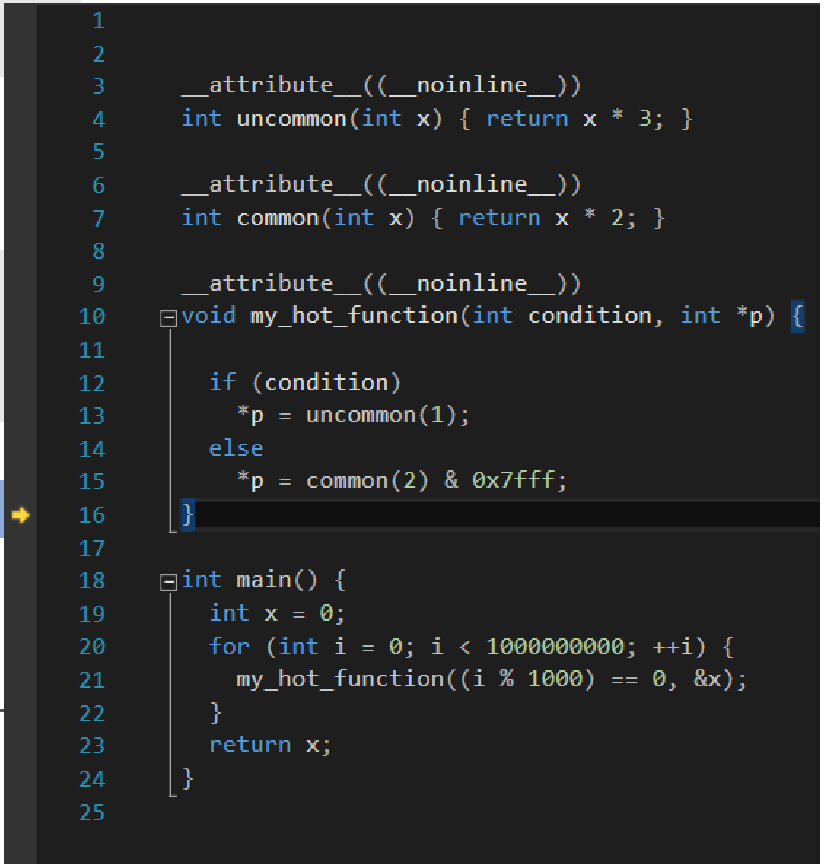
Heat Map		Instruction	
🟥	my_hot_function:	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je .LBB2_2	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp .LBB2_3	
🟥	.LBB2_2:	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	.LBB2_3:	movl %eax, (%rbx)	0:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

-O2 -g

Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		je **.LBB2_2**	
▨		movl $1, %edi	13:0
▨		callq uncommon@PLT	
▨		jmp **.LBB2_3**	0:0
🟥	**.LBB2_2:**	movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	0:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	



```


__attribute__((__noinline__))
int uncommon(int x) { return x * 3; }

__attribute__((__noinline__))
int common(int x) { return x * 2; }

__attribute__((__noinline__))
void my_hot_function(int condition, int *p) {

  if (condition)
    *p = uncommon(1);
  else
    *p = common(2) & 0x7fff;
}

int main() {
  int x = 0;
  for (int i = 0; i < 1000000000; ++i) {
    my_hot_function((i % 1000) == 0, &x);
  }
  return x;
}

```

```
void my_hot_function(int condition, int *p) {   45081

  if (condition)                                45728
    *p = uncommon(1);                           46922, calls: uncommon:48
  else
    *p = common(2) & 0x7fff;                    46871, calls: common:46654
}                                               47252
```

```
void my_hot_function(int condition, int *p) {    47066

  if (condition)                                 47725
    *p = uncommon(1);                            52, calls: uncommon:52
  else
    *p = common(2) & 0x7fff;                     48731, calls: common:48587
}                                                49093
```

Heat Map		Instruction	
🟥	**my_hot_function:**	pushq %rbp	10:0
🟥		movq %rsp, %rbp	
🟥		pushq %rbx	
🟥		pushq %rax	
🟥		movq %rsi, %rbx	
🟥		testl %edi, %edi	12:0
🟥		jne **.LBB2_1**	
🟥		movl $2, %edi	15:0
🟥		callq common@PLT	
🟥		andl $32767, %eax	
🟥	**.LBB2_3:**	movl %eax, (%rbx)	0:0
🟥		addq $8, %rsp	16:0
🟥		popq %rbx	
🟥		popq %rbp	
🟥		retq	
░	**.LBB2_1:**	movl $1, %edi	13:0
░		callq uncommon@PLT	
░		jmp **.LBB2_3**	0:0

-O2 -g
-fprofile-sample-use

```
BBI = DestBB->getFirstInsertionPt();
StoreInst *NewSI = new StoreInst(MergedVal, SI.getOperand(1),
         SI.isVolatile(),
         SI.getAlignment(),
         SI.getOrdering(),
         SI.getSynchScope());
InsertNewInstBefore(NewSI, *BBI);
NewSI->setDebugLoc(DebugLoc());

// If the two stores had AA tags, merge them.
AAMDNodes AATags;
SI.getAAMetadata(AATags);
if (AATags) {
         OtherStore->getAAMetadata(AATags, /* Merge = */ true);
         NewSI->setAAMetadata(AATags);
}
```

```
/// When two instructions are combined into a single instruction we also
/// need to combine the original locations into a single location.
///
/// When the locations are the same we can use either location. When they
/// differ, we need a third location which is distinct from either. If
/// they have the same file/line but have a different discriminator we
/// could create a location with a new discriminator. If they are from
/// different files/lines the location is ambiguous and can't be
/// represented in a single line entry.  In this case, no location
/// should be set.
///
/// Currently the function does not create a new location. If the locations
/// are the same, or cannot be discriminated, the first location is returned.
/// Otherwise an empty location will be used.
static const DILocation *getMergedLocation(const DILocation *LocA,
                                           const DILocation *LocB) {
  if (LocA && LocB && (LocA == LocB || !LocA->canDiscriminate(*LocB)))
    return LocA;
  return nullptr;
}
```

```
BBI = DestBB->getFirstInsertionPt();
StoreInst *NewSI = new StoreInst(MergedVal, SI.getOperand(1),
         SI.isVolatile(),
         SI.getAlignment(),
         SI.getOrdering(),
         SI.getSynchScope());
InsertNewInstBefore(NewSI, *BBI);
NewSI->setDebugLoc(DILocation::getMergedLocation(
                            SI.getDebugLoc(),
                            OtherStore->getDebugLoc()));

// If the two stores had AA tags, merge them.
AAMDNodes AATags;
SI.getAAMetadata(AATags);
if (AATags) {
         OtherStore->getAAMetadata(AATags, /* Merge = */ true);
         NewSI->setAAMetadata(AATags);
}
```

Example 2

Hoisting

```c


__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new) {
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}
```

```c


__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new) {
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}
```

-O2 -g

Heat Map		Instruction	Debug Line
🟥	.LBB1_2:	movb (%r14), %cl	
🟥		testb %cl, %cl	
🟥		je .LBB1_8	
🟥		movq -64(%rbp), %rdx	
🟥		leaq (%rdx,%r15), %r13	
🟥		jmp .LBB1_4	
🟥	.LBB1_10:	movzbl (%r12), %eax	
🟥		incq %r14	
🟥	.LBB1_4:	cmpb %al, %c	
🟥		jne .LBB1_6	
		movq %r14, %rdi	
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
🟥	.LBB1_6:	movzbl 1(%r14), %ecx	
🟥		testb %cl, %cl	
🟥		jne .LBB1_10	
🟥		movq -56(%rbp), %r14	
🟥	.LBB1_8:	movq -48(%rbp), %rax	

-O2 -g
-fprofile-sample-use

Heat Map		Instruction
░	**.LBB1_2:**	movb (%rdi), %cl
░		movq %rsi, -64(%rbp)
░		testb %cl, %cl
░		**je .LBB1_7**
░		leaq (%rdx,%rsi), %r13
░		movq %rdi, %r14
░		**jmp .LBB1_4**
▒	**.LBB1_9:**	movzbl (%r12), %eax
▒		incq %r14
█	**.LBB1_4:**	cmpb %al, %cl
█		je **.LBB1_5**
█	**.LBB1_6:**	movzbl 1(%r14), %ecx
█		testb %cl, %cl
█		jne **.LBB1_9**
▒		jmp **.LBB1_7**
	.LBB1_5:	movq %r14, %rdi
		movq %r13, %rsi
		callq set@PLT
		incl %ebx
		jmp **.LBB1_6**
░	**.LBB1_7:**	movq -64(%rbp), %rsi
░		movq -56(%rbp), %rdx
░		movq -48(%rbp), %rdi

```c
int replace(char *str, char *old, char *new) {
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}
```

```
10    int replace(char *str, char *old, char *new) {    491006
11      int r = 0;
12
13      for(int i = 0; old[i]; i++)                      2022452
14        for(int j = 0; str[j]; j++)                    8052510
15          if(str[j] == old[i])                         8042125
16            r += set(&str[j], &new[i]);                2002778
17
18      return r;                                        519684
19    }
```

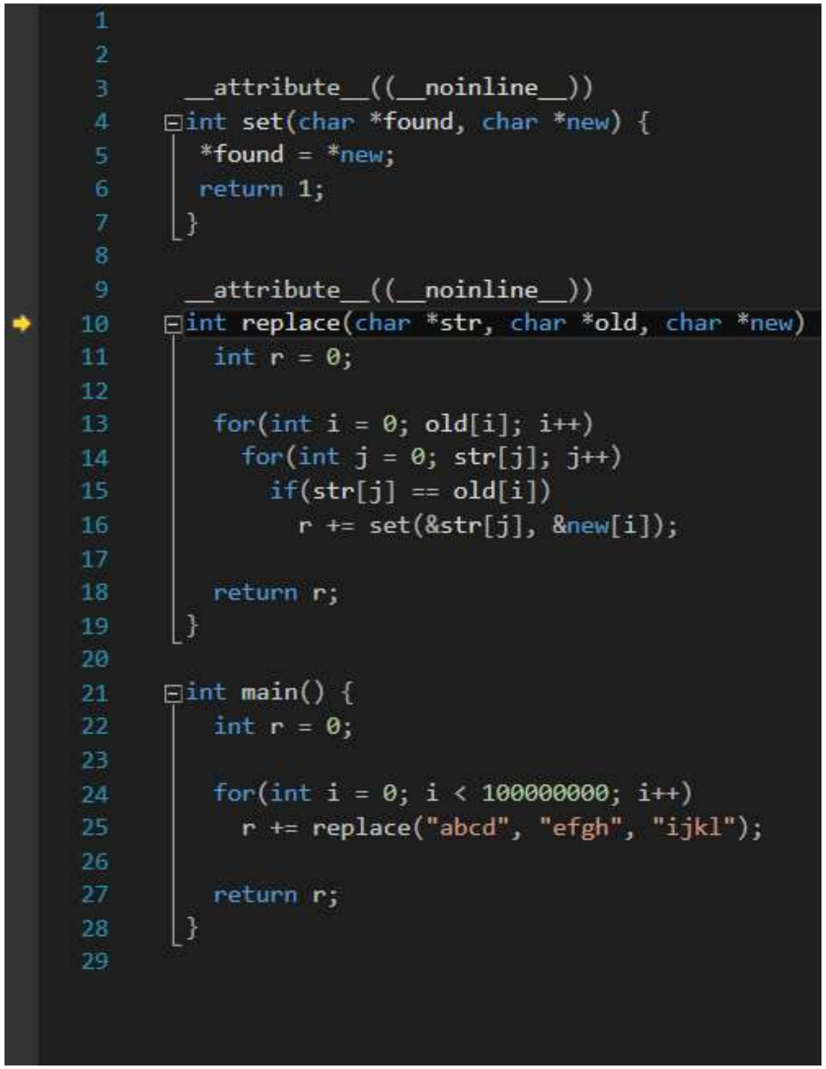
```
int replace(char *str, char *old, char *new) {        491006
  int r = 0;

  for(int i = 0; old[i]; i++)                         2022452
    for(int j = 0; str[j]; j++)                       8052510
      if(str[j] == old[i])                            8042125
        r += set(&str[j], &new[i]);                   2002778

  return r;                                           519684
}
```

-O2 -g

Heat Map		Instruction	Debug Line
▒	**.LBB1_2:**	movb (%r14), %cl	
▒		testb %cl, %cl	
▒		je **.LBB1_8**	
▒		movq -64(%rbp), %rdx	
▒		leaq (%rdx,%r15), %r13	
▒		jmp **.LBB1_4**	
▓	**.LBB1_10:**	movzbl (%r12), %eax	
▓		incq %r14	
█	**.LBB1_4:**	cmpb %al, %c	
█		jne **.LBB1_6**	
		movq %r14, %rdi	
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
█	**.LBB1_6:**	movzbl 1(%r14), %ecx	
█		testb %cl, %cl	
█		jne **.LBB1_10**	
▒		movq -56(%rbp), %r14	
▒	**.LBB1_8:**	movq -48(%rbp), %rax	

Heat Map		Instruction
░	.LBB1_2:	movb (%r14), %cl
░		testb %cl, %cl
░		je .LBB1_8
░		movq -64(%rbp), %rdx
░		leaq (%rdx,%r15), %r13
░		jmp .LBB1_4
▒	.LBB1_10:	movzbl (%r12), %eax
▒		incq %r14
█	.LBB1_4:	cmpb %al, %c
█		jne .LBB1_6
		movq %r14, %rdi
		movq %r13, %rsi
		callq set@PLT
		incl %ebx
█	.LBB1_6:	movzbl 1(%r14), %ecx
█		testb %cl, %cl
█		jne .LBB1_10
░		movq -56(%rbp), %r14
░	.LBB1_8:	movq -48(%rbp), %rax

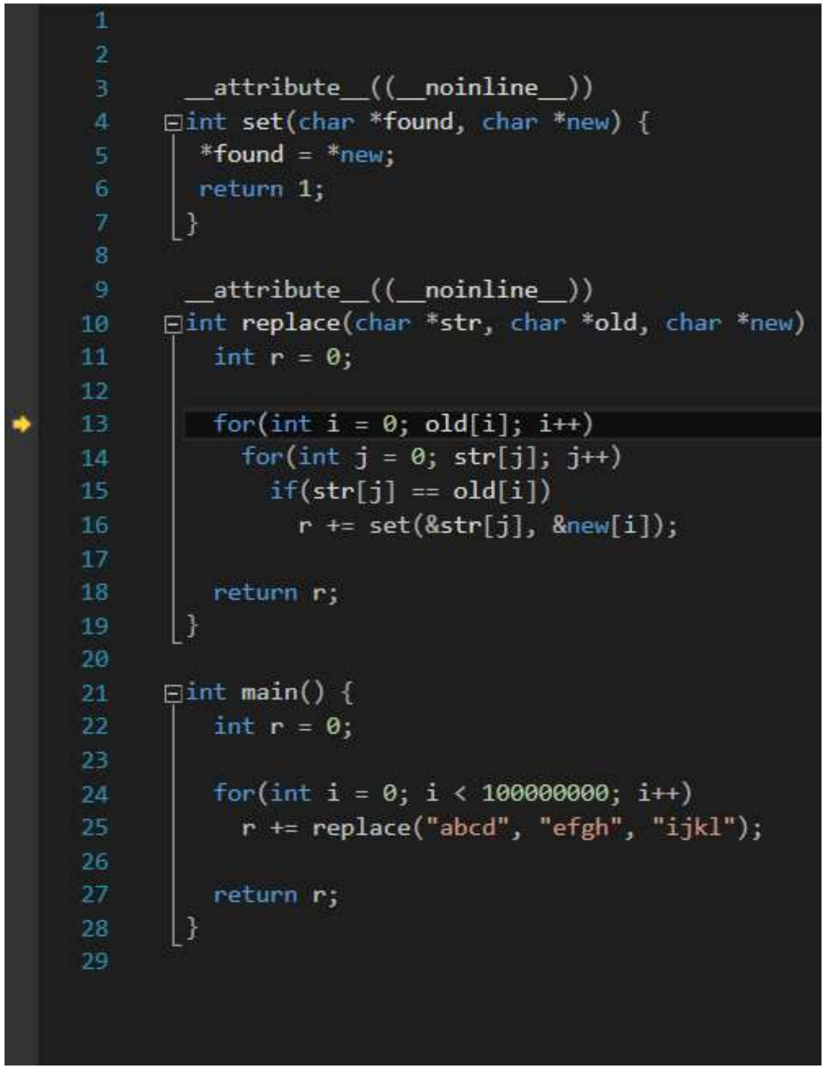
-O2 -g



```


__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}

```

Heat Map		Instruction
░	.LBB1_2:	movb (%r14), %cl
░		testb %cl, %cl
░		je .LBB1_8
░		movq -64(%rbp), %rdx
░		leaq (%rdx,%r15), %r13
░		jmp .LBB1_4
▒	.LBB1_10:	movzbl (%r12), %eax
▒		incq %r14
█	.LBB1_4:	cmpb %al, %c
█		jne .LBB1_6
		movq %r14, %rdi
		movq %r13, %rsi
		callq set@PLT
		incl %ebx
█	.LBB1_6:	movzbl 1(%r14), %ecx
█		testb %cl, %cl
█		jne .LBB1_10
░		movq -56(%rbp), %r14
░	.LBB1_8:	movq -48(%rbp), %rax

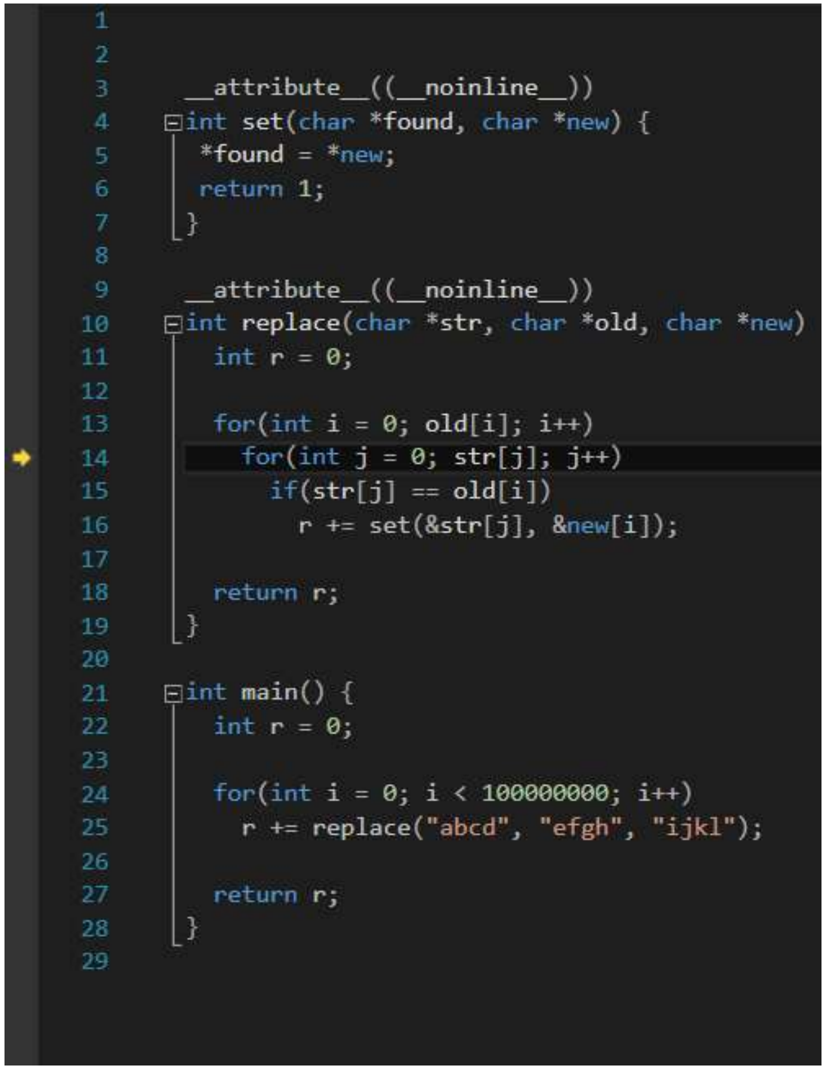
-O2 -g



```
__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}
```

-O2 -g

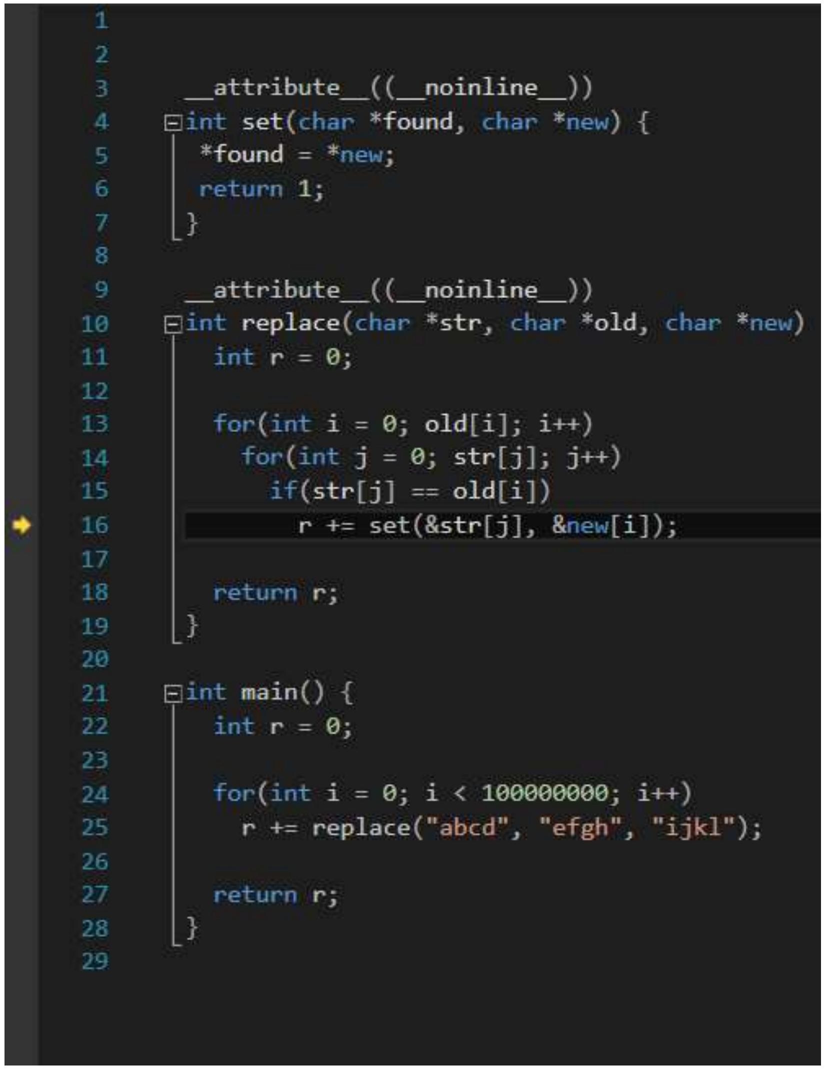
Heat Map		Instruction
🟨	.LBB1_2:	movb (%r14), %cl
🟨		testb %cl, %cl
🟨		je .LBB1_8
🟨		movq -64(%rbp), %rdx
🟨		leaq (%rdx,%r15), %r13
🟨		jmp .LBB1_4
🟧	.LBB1_10:	movzbl (%r12), %eax
🟧		incq %r14
🟥	.LBB1_4:	cmpb %al, %c
🟥		jne .LBB1_6
		movq %r14, %rdi
		movq %r13, %rsi
		callq set@PLT
		incl %ebx
🟥	.LBB1_6:	movzbl 1(%r14), %ecx
🟥		testb %cl, %cl
🟥		jne .LBB1_10
🟨		movq -56(%rbp), %r14
🟨	.LBB1_8:	movq -48(%rbp), %rax



```


__attribute__((__noinline__))
int set(char *found, char *new) {
 *found = *new;
 return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}

```

Heat Map		Instruction
🟥	.LBB1_2:	movb (%r14), %cl
🟥		testb %cl, %cl
🟥		je .LBB1_8
🟥		movq -64(%rbp), %rdx
🟥		leaq (%rdx,%r15), %r13
🟥		jmp .LBB1_4
🟥	.LBB1_10:	movzbl (%r12), %eax
🟥		incq %r14
🟥	.LBB1_4:	cmpb %al, %c
🟥		jne .LBB1_6
		movq %r14, %rdi
		movq %r13, %rsi
		callq set@PLT
		incl %ebx
🟥	.LBB1_6:	movzbl 1(%r14), %ecx
🟥		testb %cl, %cl
🟥		jne .LBB1_10
🟥		movq -56(%rbp), %r14
🟥	.LBB1_8:	movq -48(%rbp), %rax

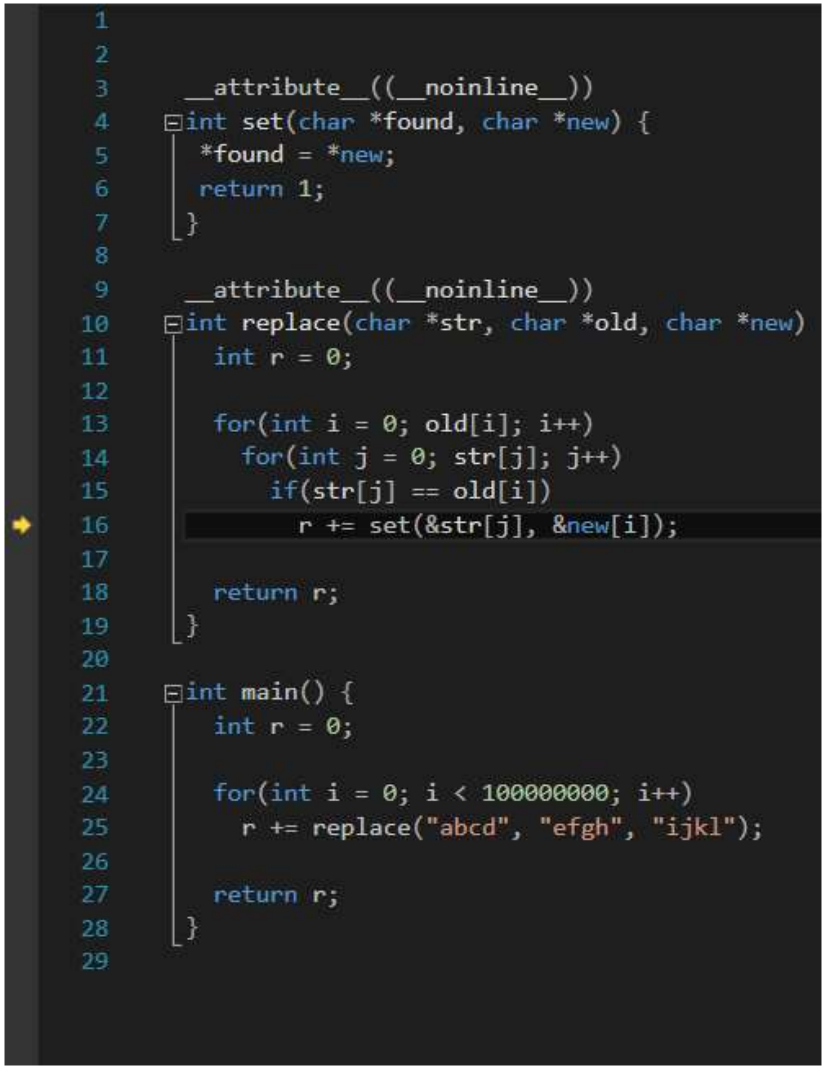
-O2 -g



```
__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}
```

-O2 -g

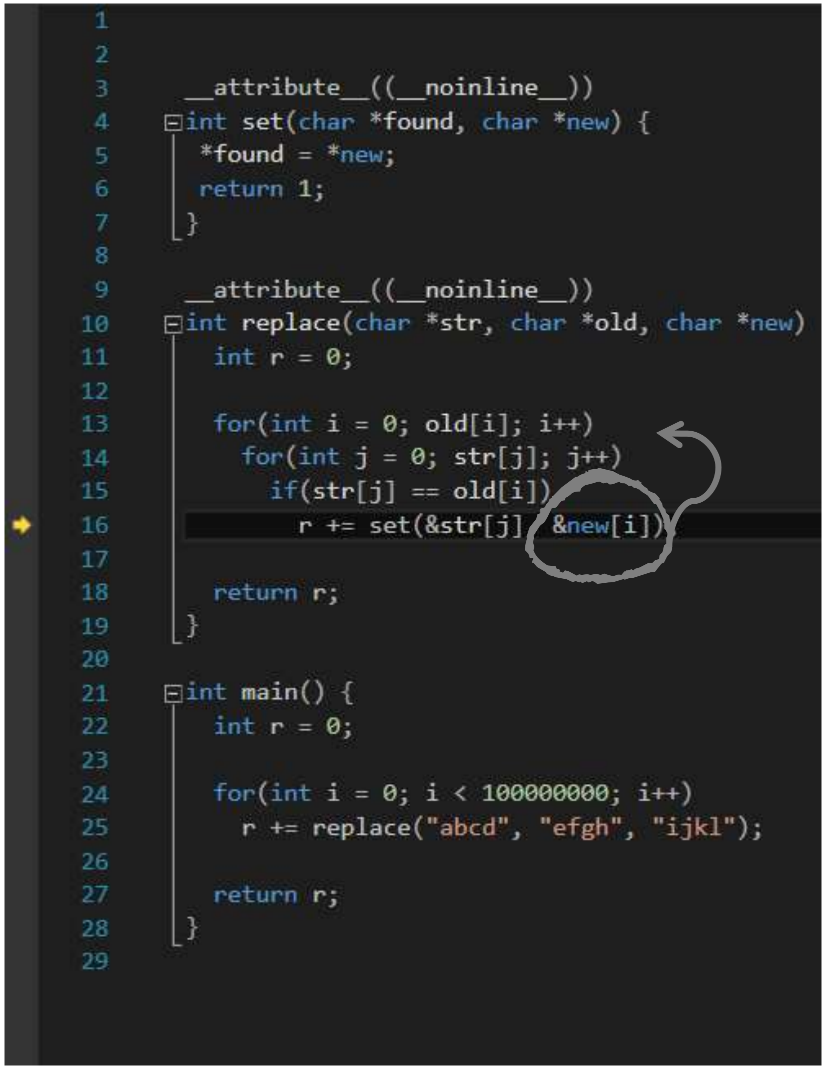
Heat Map		Instruction	Debug Line
░	.LBB1_2:	movb (%r14), %cl	14:0
░		testb %cl, %cl	
░		je .LBB1_8	
░		movq -64(%rbp), %rdx	16:0
░		leaq (%rdx,%r15), %r13	
░		jmp .LBB1_4	
▒	.LBB1_10:	movzbl (%r12), %eax	15:0
▒		incq %r14	14:0
█	.LBB1_4:	cmpb %al, %c	15:0
█		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
█	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
█		testb %cl, %cl	
█		jne .LBB1_10	
░		movq -56(%rbp), %r14	
░	.LBB1_8:	movq -48(%rbp), %rax	



```
__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}
```

-O2 -g

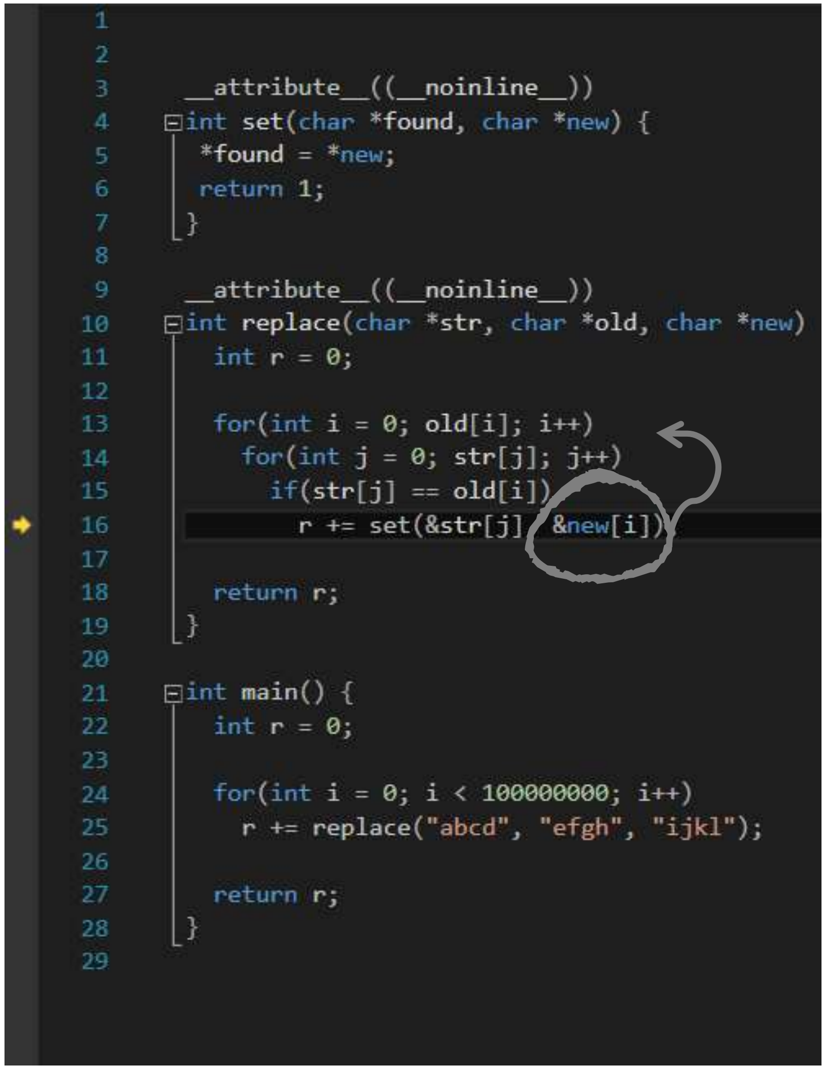
Heat Map		Instruction	Debug Line
▒	.LBB1_2:	movb (%r14), %cl	14:0
▒		testb %cl, %cl	
▒		je .LBB1_8	
▒		movq -64(%rbp), %rdx	16:0
▒		leaq (%rdx,%r15), %r13	
▒		jmp .LBB1_4	
▓	.LBB1_10:	movzbl (%r12), %eax	15:0
▓		incq %r14	14:0
█	.LBB1_4:	cmpb %al, %c	15:0
█		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
█	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
█		testb %cl, %cl	
█		jne .LBB1_10	
▒		movq -56(%rbp), %r14	
▒	.LBB1_8:	movq -48(%rbp), %rax	



```
1
2
3     __attribute__((__noinline__))
4   int set(char *found, char *new) {
5     *found = *new;
6     return 1;
7   }
8
9     __attribute__((__noinline__))
10  int replace(char *str, char *old, char *new)
11      int r = 0;
12
13      for(int i = 0; old[i]; i++)
14        for(int j = 0; str[j]; j++)
15          if(str[j] == old[i])
16            r += set(&str[j], &new[i]);
17
18      return r;
19  }
20
21  int main() {
22      int r = 0;
23
24      for(int i = 0; i < 100000000; i++)
25        r += replace("abcd", "efgh", "ijkl");
26
27      return r;
28  }
29
```

-O2 -g

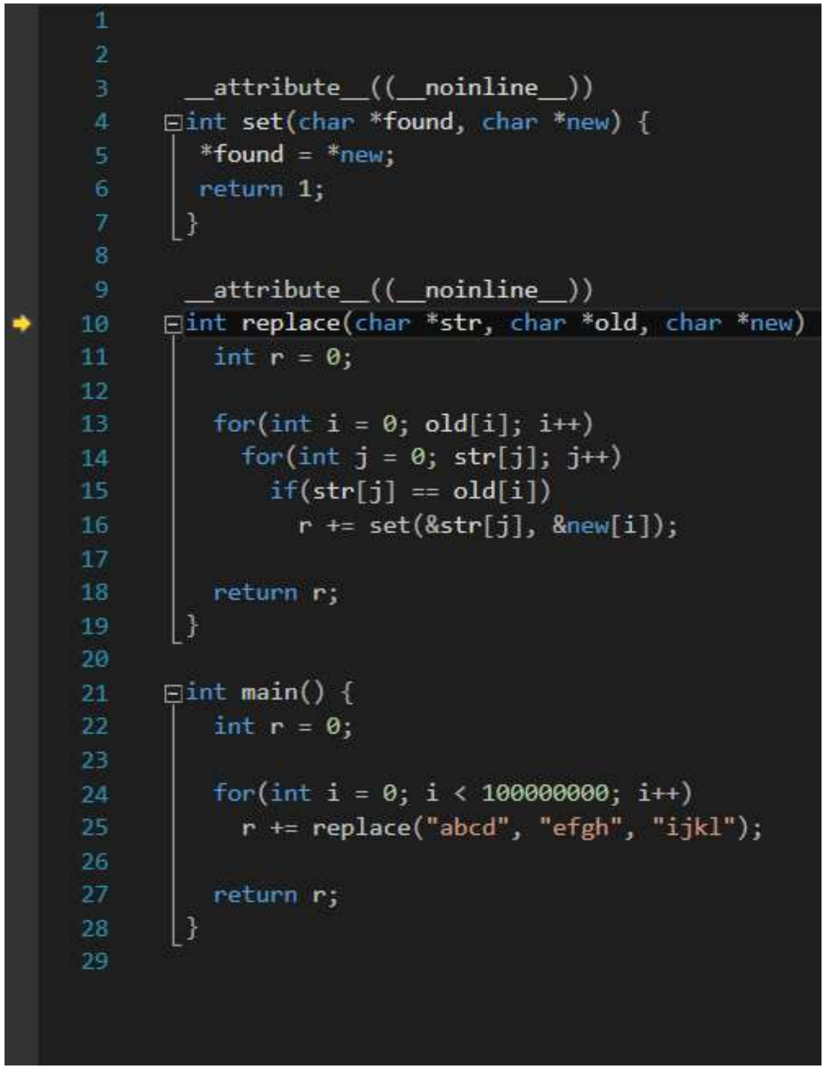
Heat Map		Instruction	Debug Line
░	.LBB1_2:	movb (%r14), %cl	14:0
░		testb %cl, %cl	
░		je .LBB1_8	
░		movq -64(%rbp), %rdx	0:0
░		leaq (%rdx,%r15), %r13	
░		jmp .LBB1_4	
▒	.LBB1_10:	movzbl (%r12), %eax	
▒		incq %r14	14:0
█	.LBB1_4:	cmpb %al, %c	15:0
█		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
█	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
█		testb %cl, %cl	
█		jne .LBB1_10	
░		movq -56(%rbp), %r14	
░	.LBB1_8:	movq -48(%rbp), %rax	



```
1
2
3      __attribute__((__noinline__))
4     int set(char *found, char *new) {
5       *found = *new;
6       return 1;
7     }
8
9      __attribute__((__noinline__))
10    int replace(char *str, char *old, char *new)
11      int r = 0;
12
13      for(int i = 0; old[i]; i++)
14        for(int j = 0; str[j]; j++)
15          if(str[j] == old[i])
16            r += set(&str[j], &new[i]);
17
18      return r;
19    }
20
21    int main() {
22      int r = 0;
23
24      for(int i = 0; i < 100000000; i++)
25        r += replace("abcd", "efgh", "ijkl");
26
27      return r;
28    }
29
```

-O2 -g

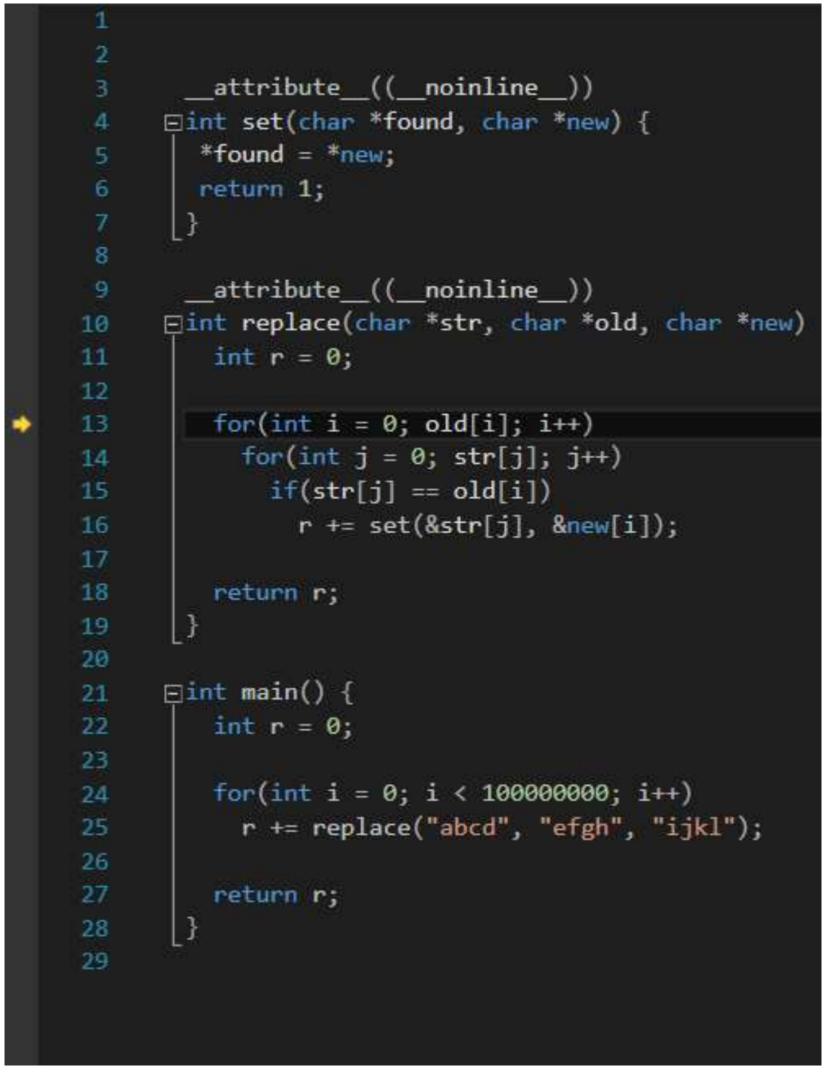
Heat Map		Instruction	Debug Line
░	.LBB1_2:	movb (%r14), %cl	14:0
░		testb %cl, %cl	
░		je .LBB1_8	
░		movq -64(%rbp), %rdx	0:0
░		leaq (%rdx,%r15), %r13	
░		jmp .LBB1_4	
▒	.LBB1_10:	movzbl (%r12), %eax	
▒		incq %r14	14:0
█	.LBB1_4:	cmpb %al, %c	15:0
█		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
█	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
█		testb %cl, %cl	
█		jne .LBB1_10	
░		movq -56(%rbp), %r14	
░	.LBB1_8:	movq -48(%rbp), %rax	



```


__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}

```

-O2 -g

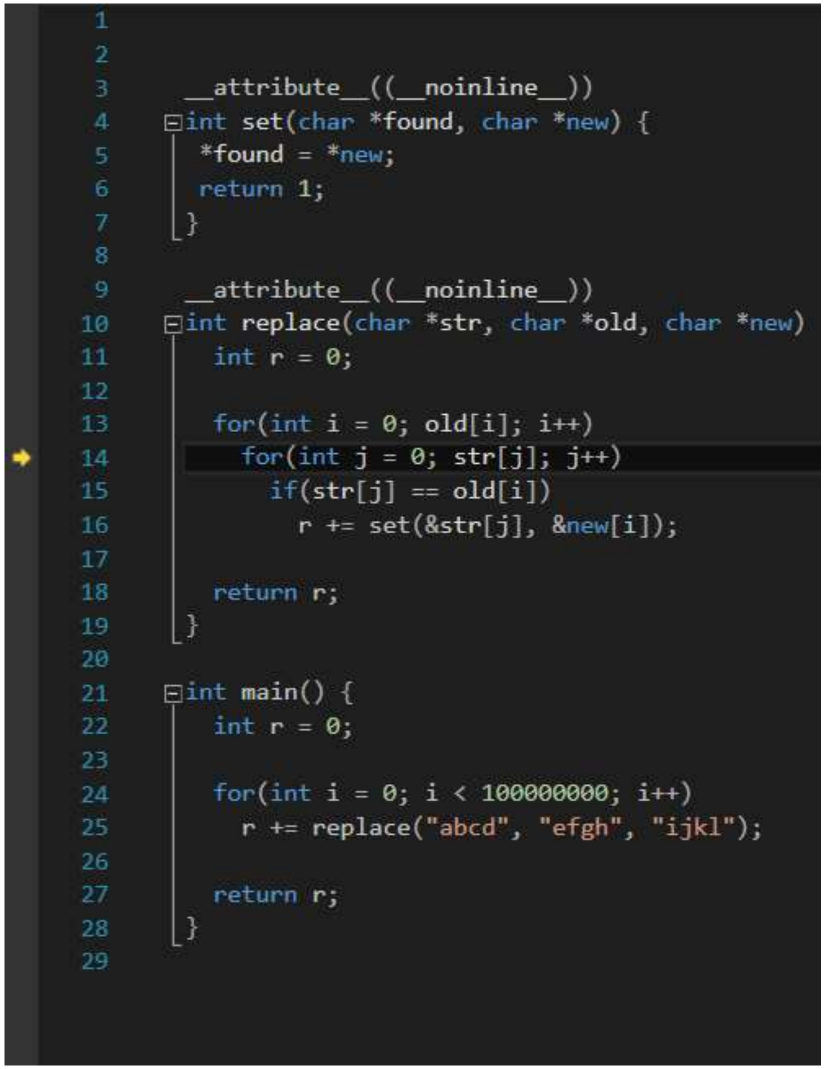
Heat Map		Instruction	Debug Line
🟥	.LBB1_2:	movb (%r14), %cl	14:0
🟥		testb %cl, %cl	
🟥		je .LBB1_8	
🟥		movq -64(%rbp), %rdx	0:0
🟥		leaq (%rdx,%r15), %r13	
🟥		jmp .LBB1_4	
🟥	.LBB1_10:	movzbl (%r12), %eax	
🟥		incq %r14	14:0
🟥	.LBB1_4:	cmpb %al, %c	15:0
🟥		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
🟥	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
🟥		testb %cl, %cl	
🟥		jne .LBB1_10	
🟥		movq -56(%rbp), %r14	
🟥	.LBB1_8:	movq -48(%rbp), %rax	



```
 1
 2
 3      __attribute__((__noinline__))
 4     int set(char *found, char *new) {
 5       *found = *new;
 6       return 1;
 7     }
 8
 9      __attribute__((__noinline__))
10     int replace(char *str, char *old, char *new)
11       int r = 0;
12
13       for(int i = 0; old[i]; i++)
14         for(int j = 0; str[j]; j++)
15           if(str[j] == old[i])
16             r += set(&str[j], &new[i]);
17
18       return r;
19     }
20
21     int main() {
22       int r = 0;
23
24       for(int i = 0; i < 100000000; i++)
25         r += replace("abcd", "efgh", "ijkl");
26
27       return r;
28     }
29
```

`-O2 -g`

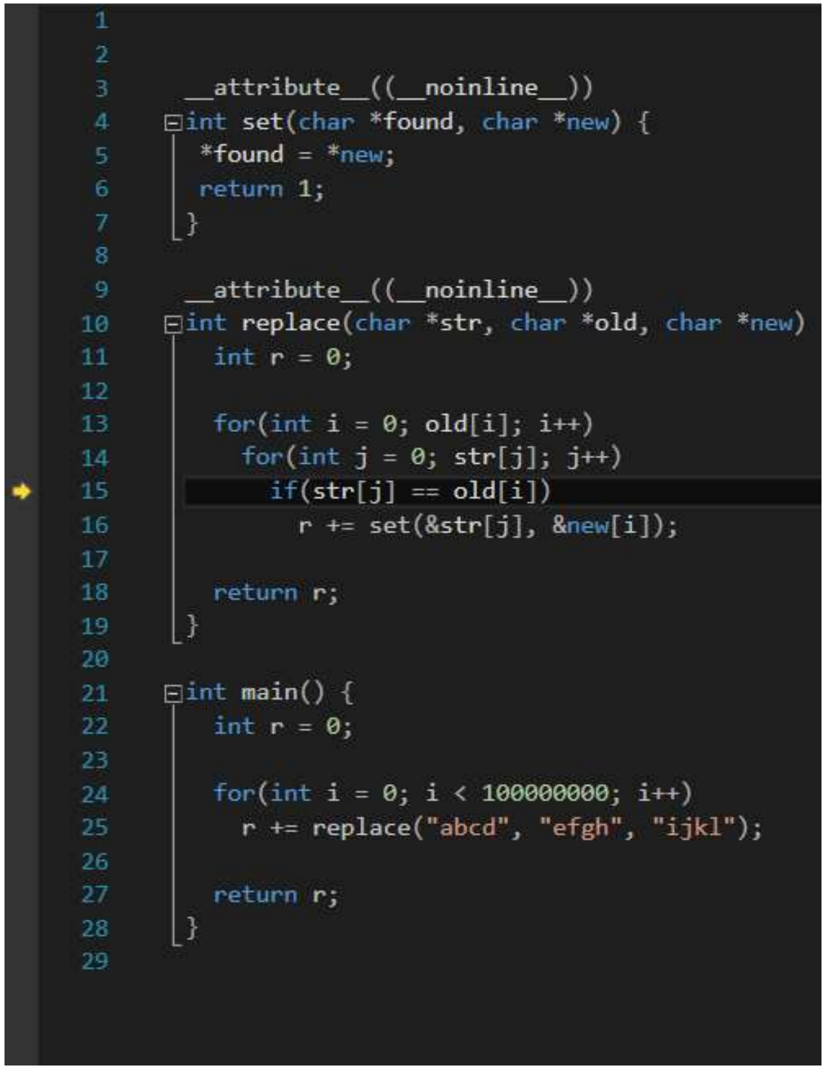
Heat Map		Instruction	Debug Line
🟥	.LBB1_2:	movb (%r14), %cl	14:0
🟥		testb %cl, %cl	
🟥		je .LBB1_8	
🟥		movq -64(%rbp), %rdx	0:0
🟥		leaq (%rdx,%r15), %r13	
🟥		jmp .LBB1_4	
🟥	.LBB1_10:	movzbl (%r12), %eax	
🟥		incq %r14	14:0
🟥	.LBB1_4:	cmpb %al, %c	15:0
🟥		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
🟥	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
🟥		testb %cl, %cl	
🟥		jne .LBB1_10	
🟥		movq -56(%rbp), %r14	
🟥	.LBB1_8:	movq -48(%rbp), %rax	



```
1
2
3     __attribute__((__noinline__))
4   int set(char *found, char *new) {
5     *found = *new;
6     return 1;
7   }
8
9     __attribute__((__noinline__))
10  int replace(char *str, char *old, char *new)
11      int r = 0;
12
13      for(int i = 0; old[i]; i++)
14        for(int j = 0; str[j]; j++)
15          if(str[j] == old[i])
16            r += set(&str[j], &new[i]);
17
18      return r;
19  }
20
21  int main() {
22      int r = 0;
23
24      for(int i = 0; i < 100000000; i++)
25        r += replace("abcd", "efgh", "ijkl");
26
27      return r;
28  }
29
```

Heat Map		Instruction	Debug Line
🟥	.LBB1_2:	movb (%r14), %cl	14:0
🟥		testb %cl, %cl	
🟥		je .LBB1_8	
🟥		movq -64(%rbp), %rdx	0:0
🟥		leaq (%rdx,%r15), %r13	
🟥		jmp .LBB1_4	
🟥	.LBB1_10:	movzbl (%r12), %eax	
🟥		incq %r14	14:0
🟥	.LBB1_4:	cmpb %al, %c	15:0
🟥		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
🟥	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
🟥		testb %cl, %cl	
🟥		jne .LBB1_10	
🟥		movq -56(%rbp), %r14	
🟥	.LBB1_8:	movq -48(%rbp), %rax	

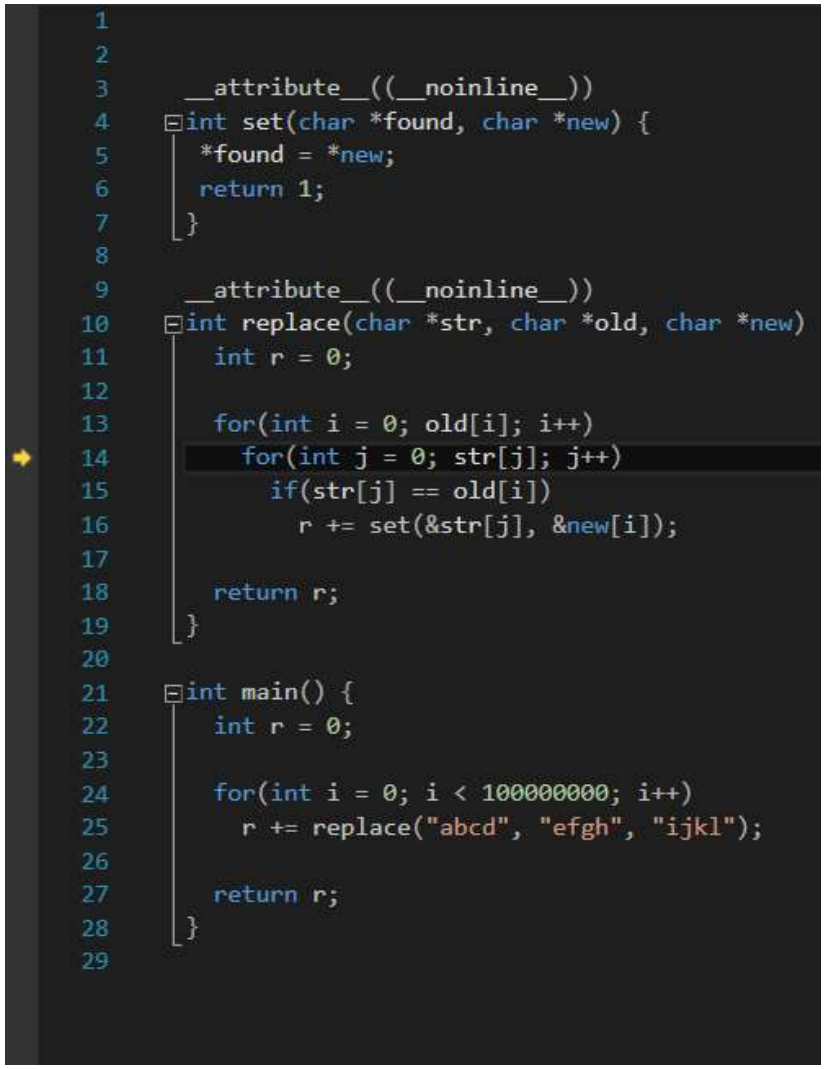
-O2 -g



```
__attribute__((__noinline__))
int set(char *found, char *new) {
  *found = *new;
  return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}
```

-O2 -g

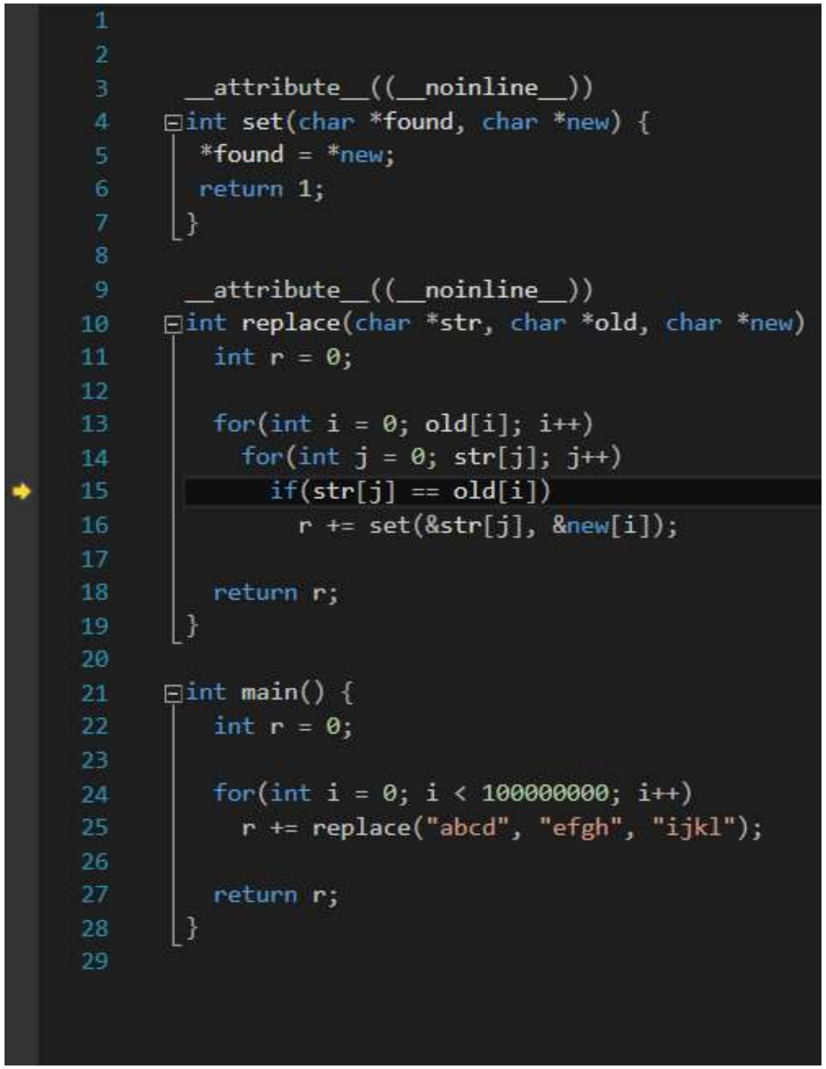
Heat Map		Instruction	Debug Line
🟥	.LBB1_2:	movb (%r14), %cl	14:0
		testb %cl, %cl	
		je .LBB1_8	
		movq -64(%rbp), %rdx	0:0
		leaq (%rdx,%r15), %r13	
		jmp .LBB1_4	
🟥	.LBB1_10:	movzbl (%r12), %eax	
		incq %r14	14:0
🟥	.LBB1_4:	cmpb %al, %c	15:0
		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
🟥	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
		testb %cl, %cl	
		jne .LBB1_10	
🟥		movq -56(%rbp), %r14	
	.LBB1_8:	movq -48(%rbp), %rax	



```
1
2
3      __attribute__((__noinline__))
4     int set(char *found, char *new) {
5       *found = *new;
6       return 1;
7     }
8
9      __attribute__((__noinline__))
10    int replace(char *str, char *old, char *new)
11      int r = 0;
12
13      for(int i = 0; old[i]; i++)
14        for(int j = 0; str[j]; j++)
15          if(str[j] == old[i])
16            r += set(&str[j], &new[i]);
17
18      return r;
19    }
20
21    int main() {
22      int r = 0;
23
24      for(int i = 0; i < 100000000; i++)
25        r += replace("abcd", "efgh", "ijkl");
26
27      return r;
28    }
29
```

-O2 -g

Heat Map		Instruction	Debug Line
🟥	.LBB1_2:	movb (%r14), %cl	14:0
🟥		testb %cl, %cl	
🟥		je .LBB1_8	
🟥		movq -64(%rbp), %rdx	0:0
🟥		leaq (%rdx,%r15), %r13	
🟥		jmp .LBB1_4	
🟥	.LBB1_10:	movzbl (%r12), %eax	
🟥		incq %r14	14:0
🟥	.LBB1_4:	cmpb %al, %c	15:0
🟥		jne .LBB1_6	
		movq %r14, %rdi	16:0
		movq %r13, %rsi	
		callq set@PLT	
		incl %ebx	
🟥	.LBB1_6:	movzbl 1(%r14), %ecx	14:0
🟥		testb %cl, %cl	
🟥		jne .LBB1_10	
🟥		movq -56(%rbp), %r14	
🟥	.LBB1_8:	movq -48(%rbp), %rax	



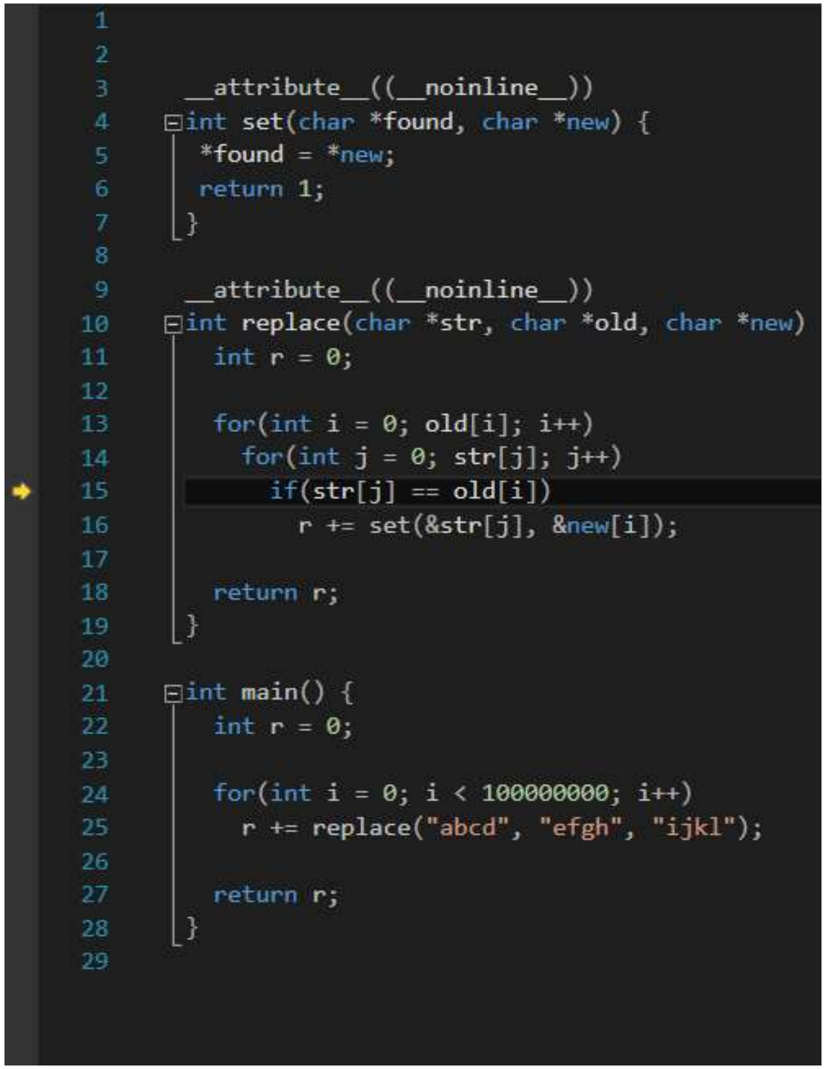
```
1
2
3      __attribute__((__noinline__))
4   int set(char *found, char *new) {
5     *found = *new;
6     return 1;
7   }
8
9      __attribute__((__noinline__))
10  int replace(char *str, char *old, char *new)
11      int r = 0;
12
13      for(int i = 0; old[i]; i++)
14        for(int j = 0; str[j]; j++)
15          if(str[j] == old[i])
16            r += set(&str[j], &new[i]);
17
18      return r;
19  }
20
21  int main() {
22      int r = 0;
23
24      for(int i = 0; i < 100000000; i++)
25        r += replace("abcd", "efgh", "ijkl");
26
27      return r;
28  }
29
```

```
int replace(char *str, char *old, char *new) {     491006
  int r = 0;

  for(int i = 0; old[i]; i++)                      2022452
    for(int j = 0; str[j]; j++)                    8052510
      if(str[j] == old[i])                         8042125
        r += set(&str[j], &new[i]);                2002778

  return r;                                        519684
}
```

```
10    int replace(char *str, char *old, char *new) {    451903
11      int r = 0;
12
13      for(int i = 0; old[i]; i++)                     1873885
14        for(int j = 0; str[j]; j++)                   7449646
15          if(str[j] == old[i])                        7437374
16            r += set(&str[j], &new[i]);               0
17
18      return r;                                       485259
19    }
```

`-O2 -g`
`-fprofile-sample-use`

Heat Map		Instruction	Debug Line
🟥		popq %rbx	
🟥		popq %r12	
🟥		**popq %r13**	
🟥		popq %r14	
🟥		popq %r15	
🟥		**popq %rbp**	
🟥		retq	
	.LBB1_5:	movq %rdi, -56(%rbp)	
		movq %r12, %rdi	
		movl %eax, -44(%rbp)	
		movq %rsi, %rbx	
		movq %rdx, -64(%rbp)	
		callq set@PLT	
		movl -44(%rbp), %eax	
		movq -56(%rbp), %rdi	
		movq -64(%rbp), %rdx	
		movq %rbx, %rsi	
		incl %eax	
		jmp .LBB1_6	



```


__attribute__((__noinline__))
int set(char *found, char *new) {
 *found = *new;
 return 1;
}

__attribute__((__noinline__))
int replace(char *str, char *old, char *new)
  int r = 0;

  for(int i = 0; old[i]; i++)
    for(int j = 0; str[j]; j++)
      if(str[j] == old[i])
        r += set(&str[j], &new[i]);

  return r;
}

int main() {
  int r = 0;

  for(int i = 0; i < 100000000; i++)
    r += replace("abcd", "efgh", "ijkl");

  return r;
}

```

Example 3

Rematerialization

```c


typedef float vec4 __attribute__((ext_vector_type(4)));

__attribute__((__noinline__))
vec4 do_something(vec4 X) {
  return X * X;
}

__attribute__((__noinline__))
vec4 hot_function(vec4 X, int i) {
const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};

  if(i)
    X += V;

 return V - do_something(X);
}

int main() {
 vec4 r = (vec4){0, 0, 0, 0};

 for(int i = 0; i < 100000000; i++)
   r += hot_function(r, 0);

 return r[0] > 10.0f;
}
```

```


typedef float vec4 __attribute__((ext_vector_type(4)));

__attribute__((__noinline__))
vec4 do_something(vec4 X) {
  return X * X;
}

__attribute__((__noinline__))
vec4 hot_function(vec4 X, int i) {
const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};

  if(i)
    X += V;

 return V - do_something(X);
}

int main() {
 vec4 r = (vec4){0, 0, 0, 0};

 for(int i = 0; i < 100000000; i++)
   r += hot_function(r, 0);

 return r[0] > 10.0f;
}
```

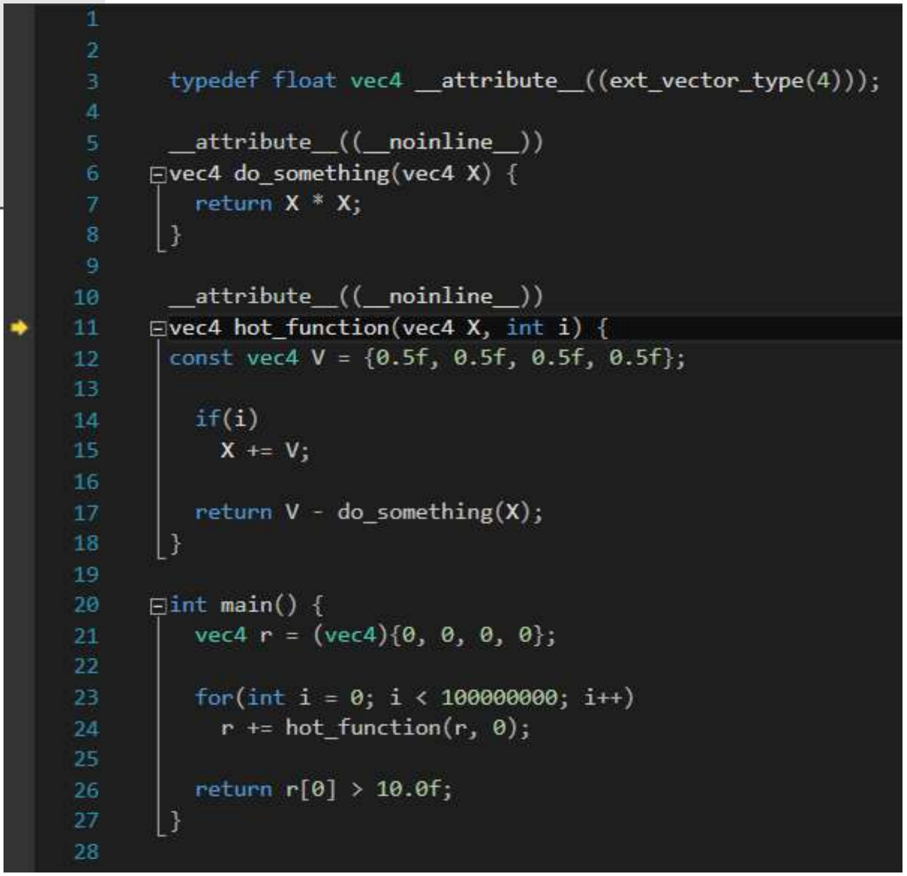
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

-O2 -g

11	`vec4 hot_function(vec4 X, int i) {`	
12	`  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};`	
13		
14	`  if(i)`	48897
15	`    X += V;`	49164
16		
17	`  return V - do_something(X);`	49592, calls: do_something:49402
18	`}`	

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

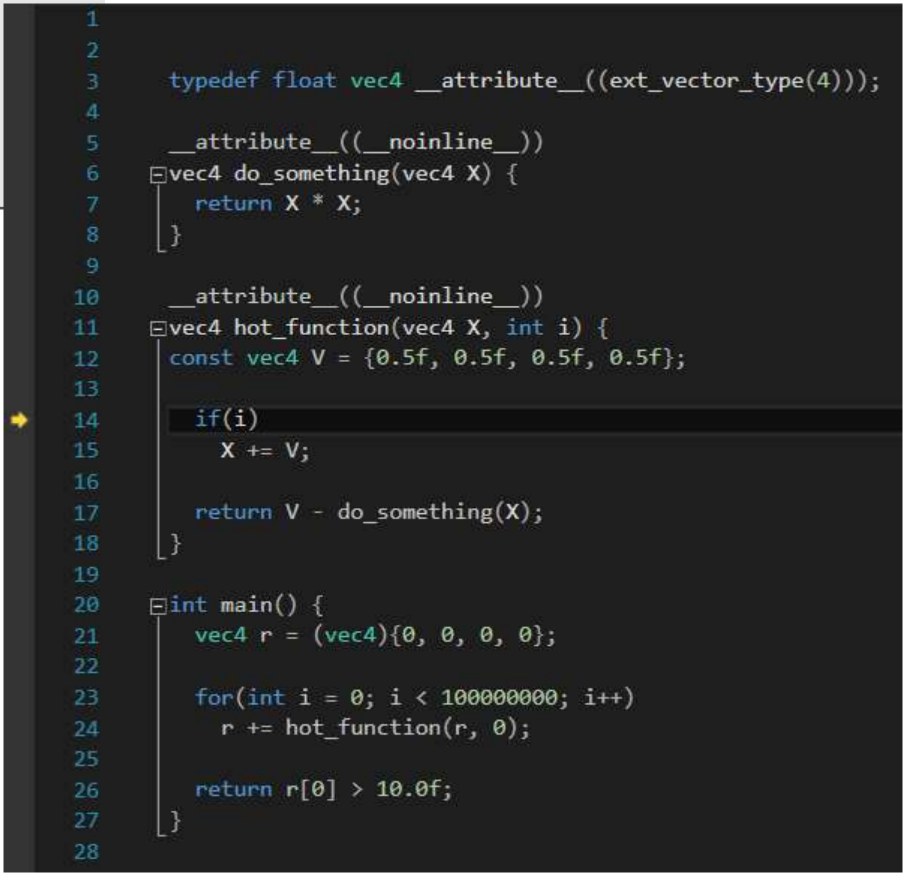
-O2 -g



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

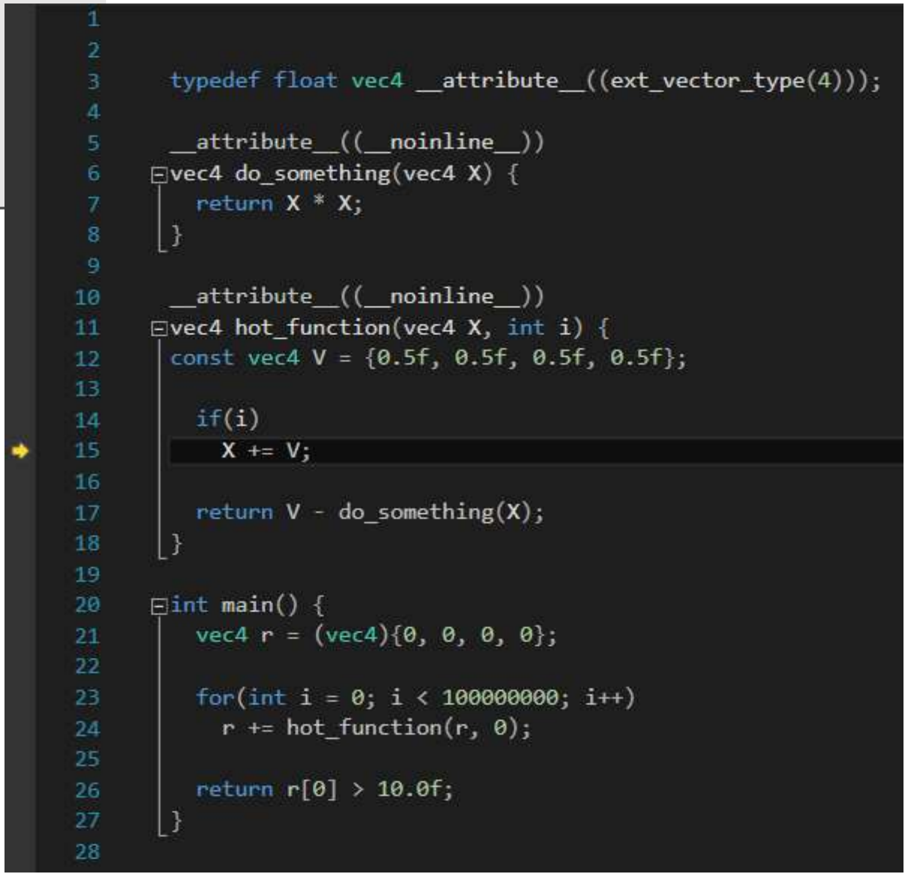
-O2 -g



```
1
2
3   typedef float vec4 __attribute__((ext_vector_type(4)));
4
5   __attribute__((__noinline__))
6   vec4 do_something(vec4 X) {
7     return X * X;
8   }
9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

-O2 -g

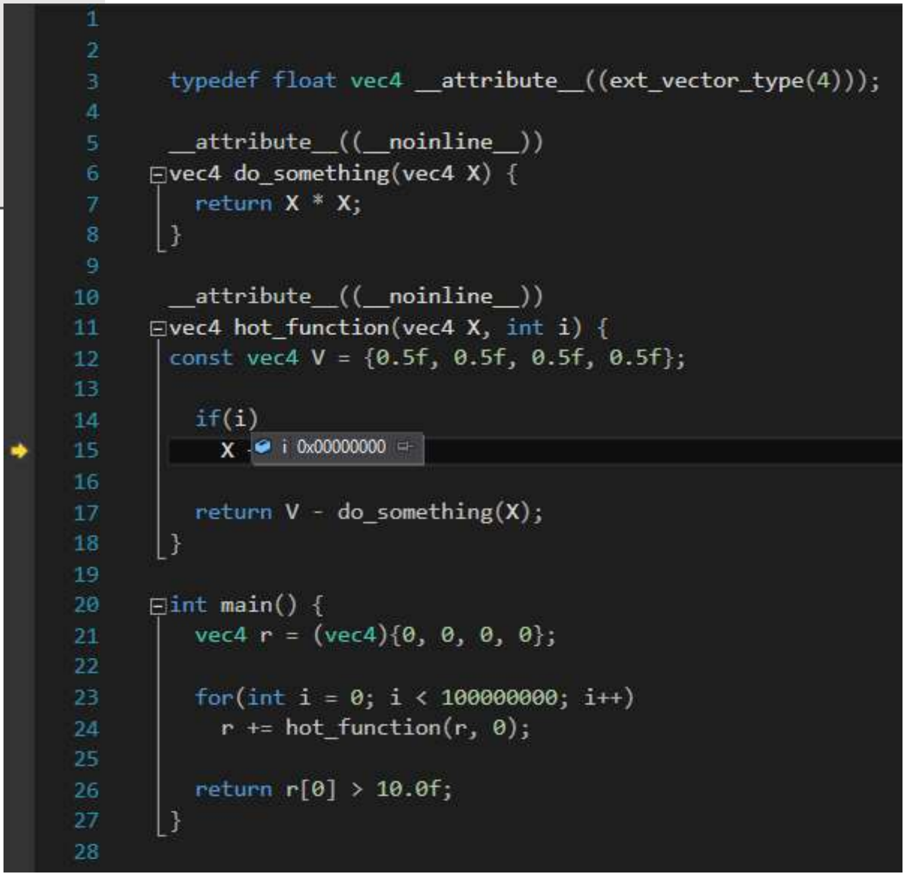
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

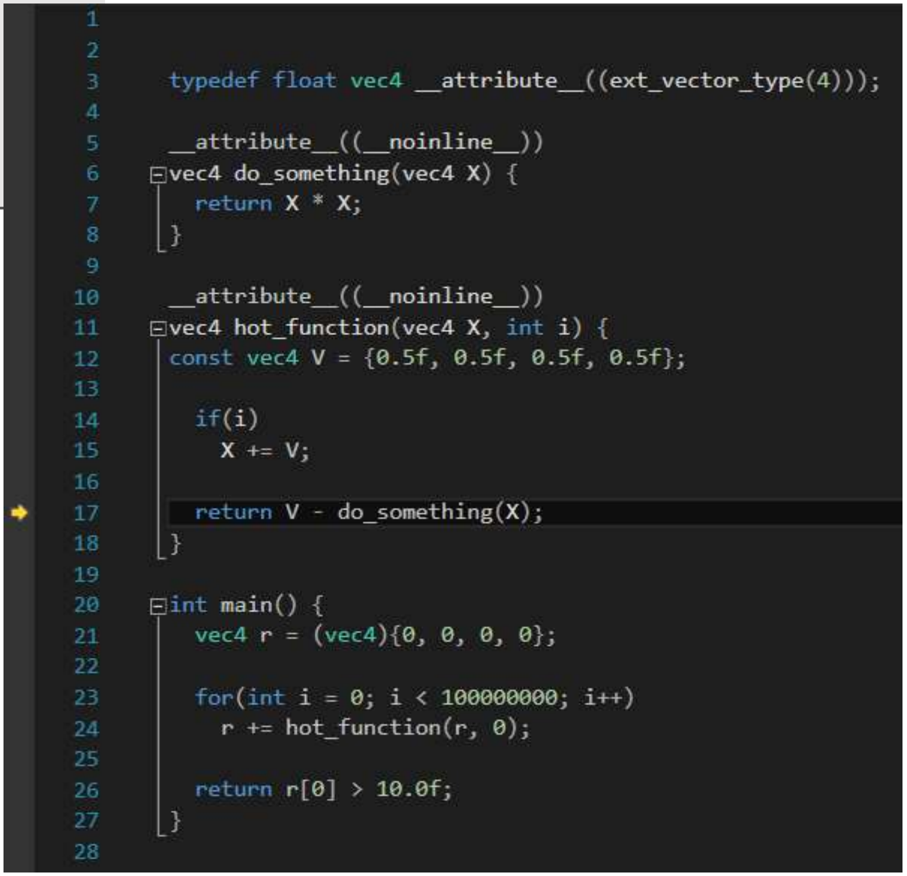
-O2 -g



```

typedef float vec4 __attribute__((ext_vector_type(4)));

__attribute__((__noinline__))
vec4 do_something(vec4 X) {
  return X * X;
}

__attribute__((__noinline__))
vec4 hot_function(vec4 X, int i) {
const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};

  if(i)
    X  i 0x00000000

  return V - do_something(X);
}

int main() {
  vec4 r = (vec4){0, 0, 0, 0};

  for(int i = 0; i < 100000000; i++)
    r += hot_function(r, 0);

  return r[0] > 10.0f;
}

```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

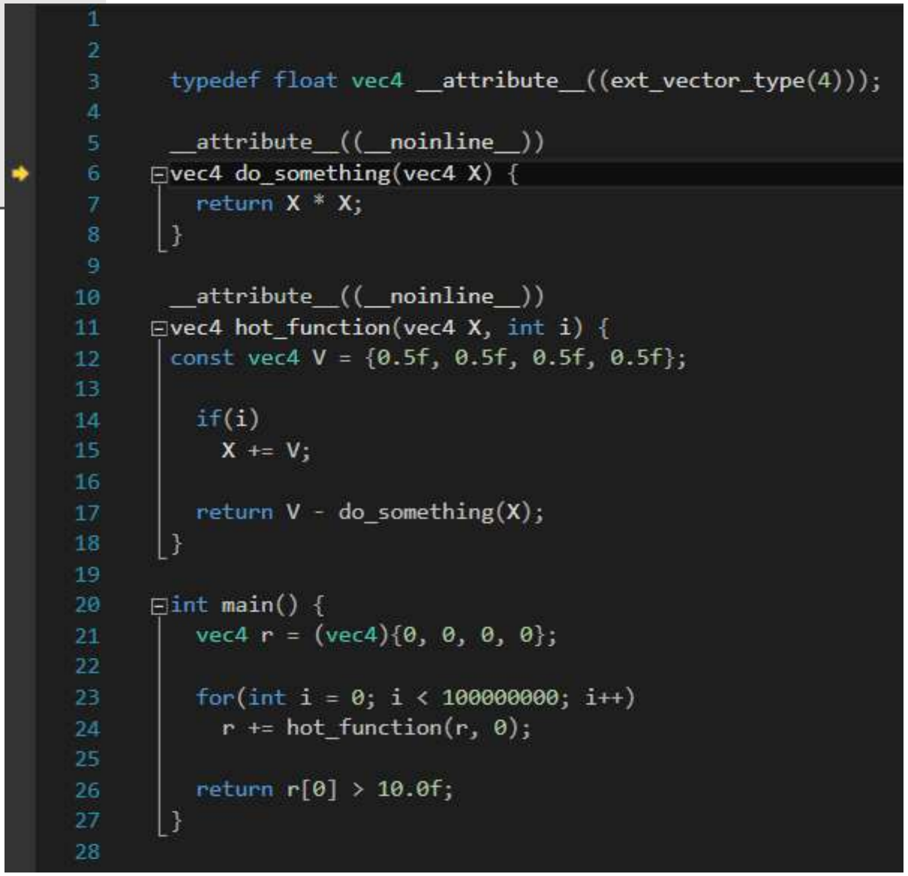
-O2 -g



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

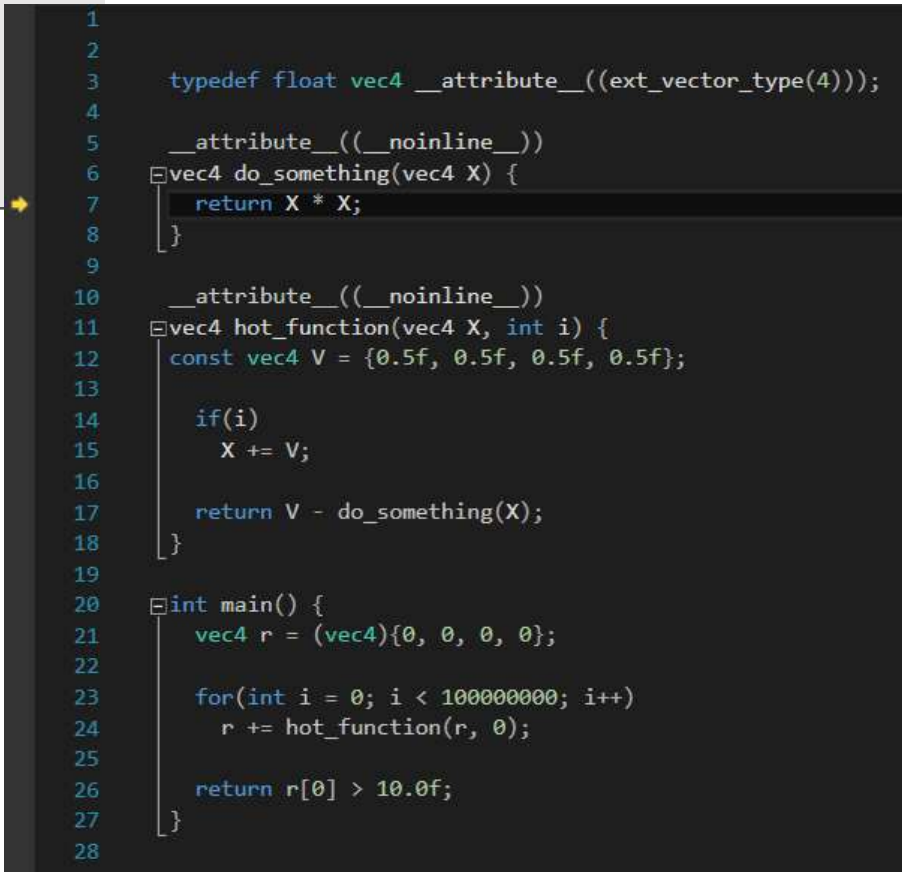
-O2 -g



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

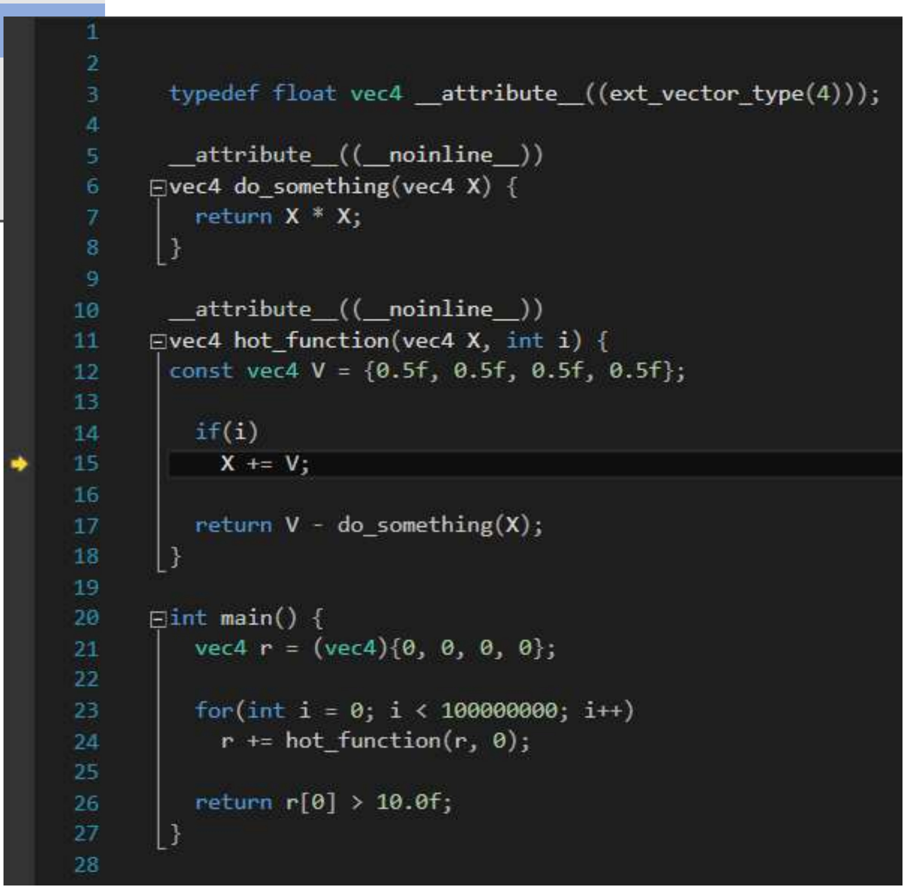
-O2 -g



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

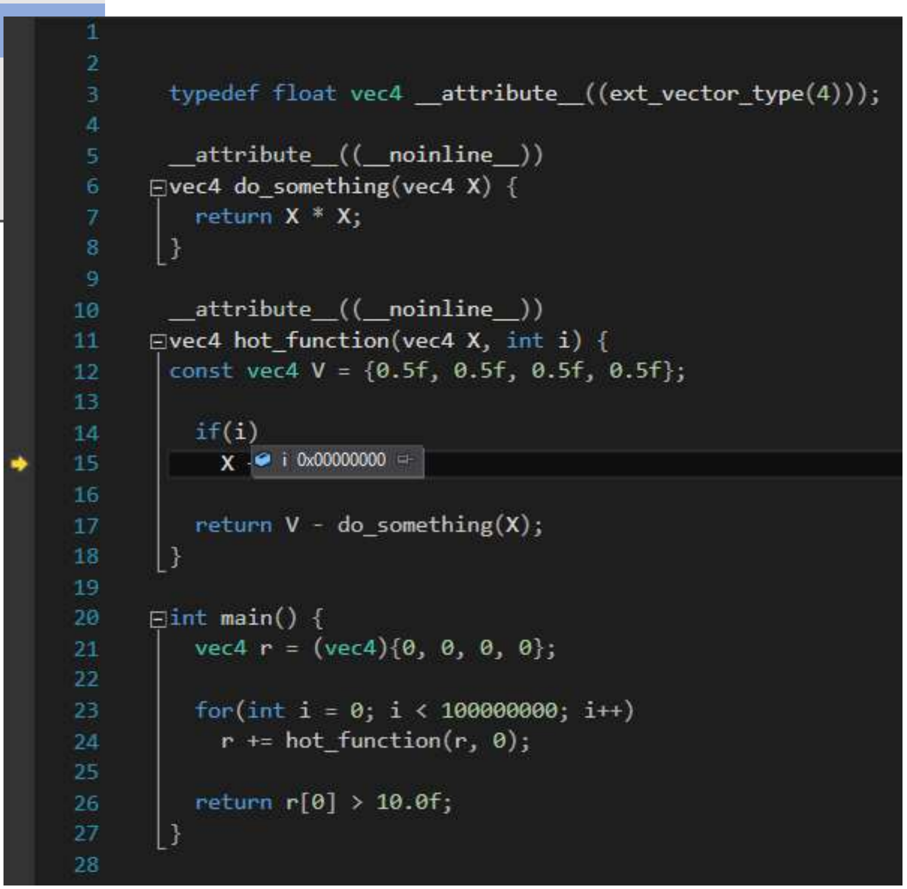
-O2 -g



```

typedef float vec4 __attribute__((ext_vector_type(4)));

__attribute__((__noinline__))
vec4 do_something(vec4 X) {
  return X * X;
}

__attribute__((__noinline__))
vec4 hot_function(vec4 X, int i) {
const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};

  if(i)
    X += V;

  return V - do_something(X);
}

int main() {
  vec4 r = (vec4){0, 0, 0, 0};

  for(int i = 0; i < 100000000; i++)
    r += hot_function(r, 0);

  return r[0] > 10.0f;
}
```

-O2 -g

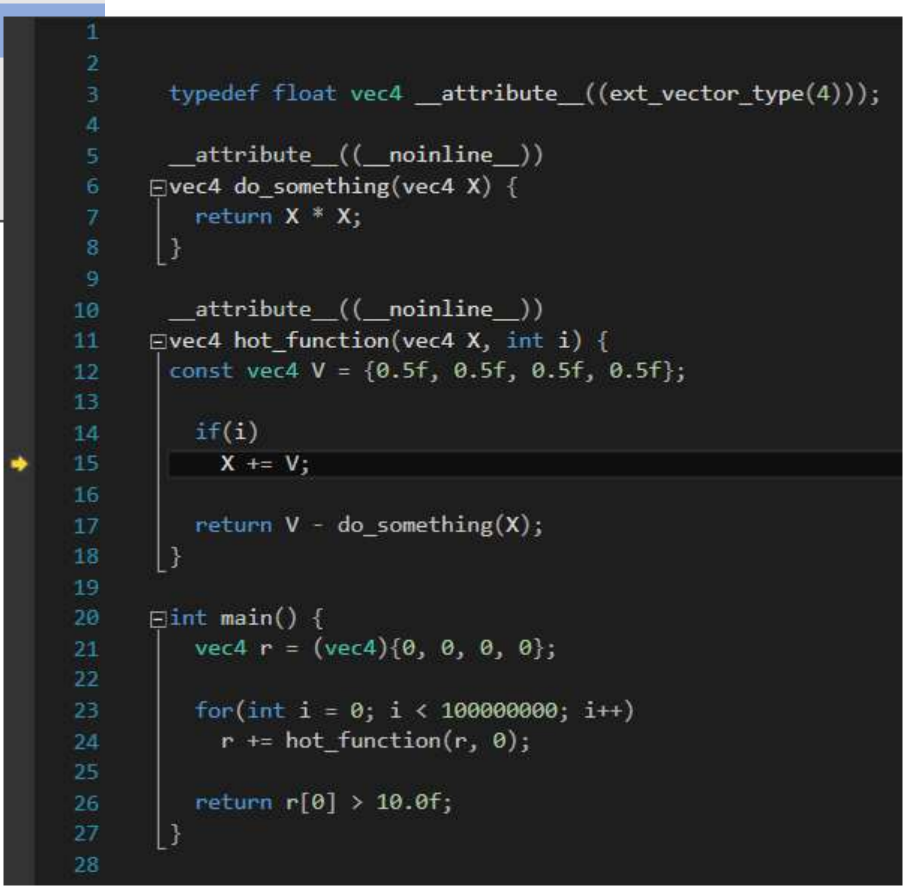
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	



```
1
2
3   typedef float vec4 __attribute__((ext_vector_type(4)));
4
5   __attribute__((__noinline__))
6   vec4 do_something(vec4 X) {
7     return X * X;
8   }
9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X  i 0x00000000
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

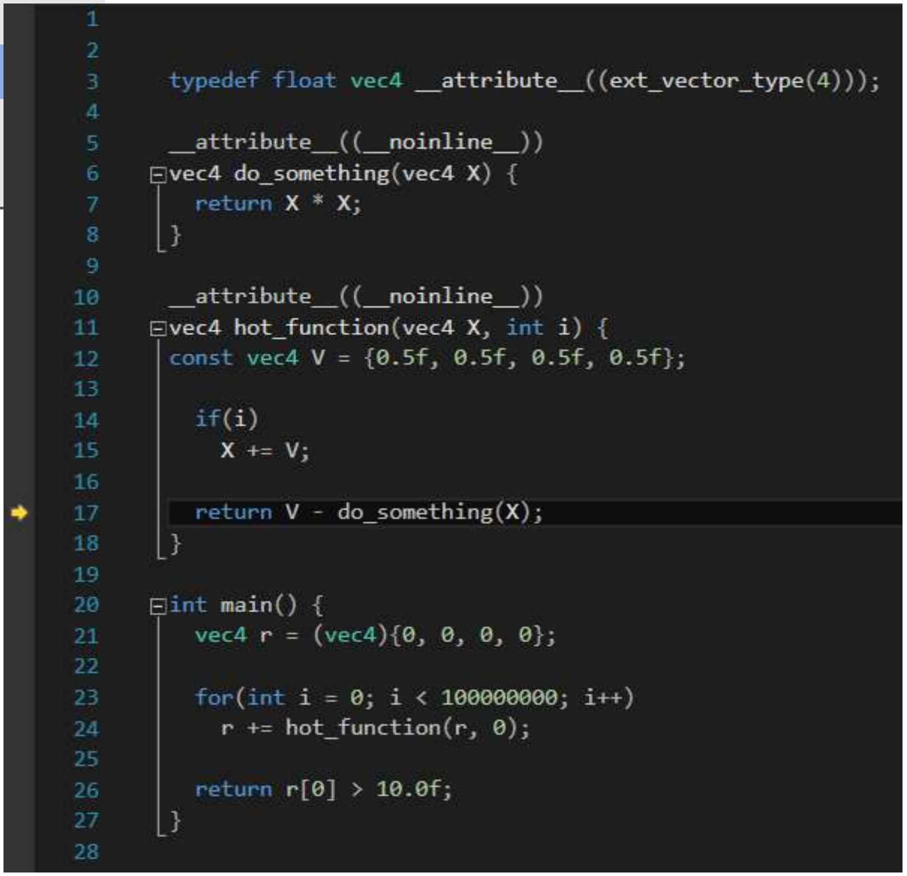
-O2 -g



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	

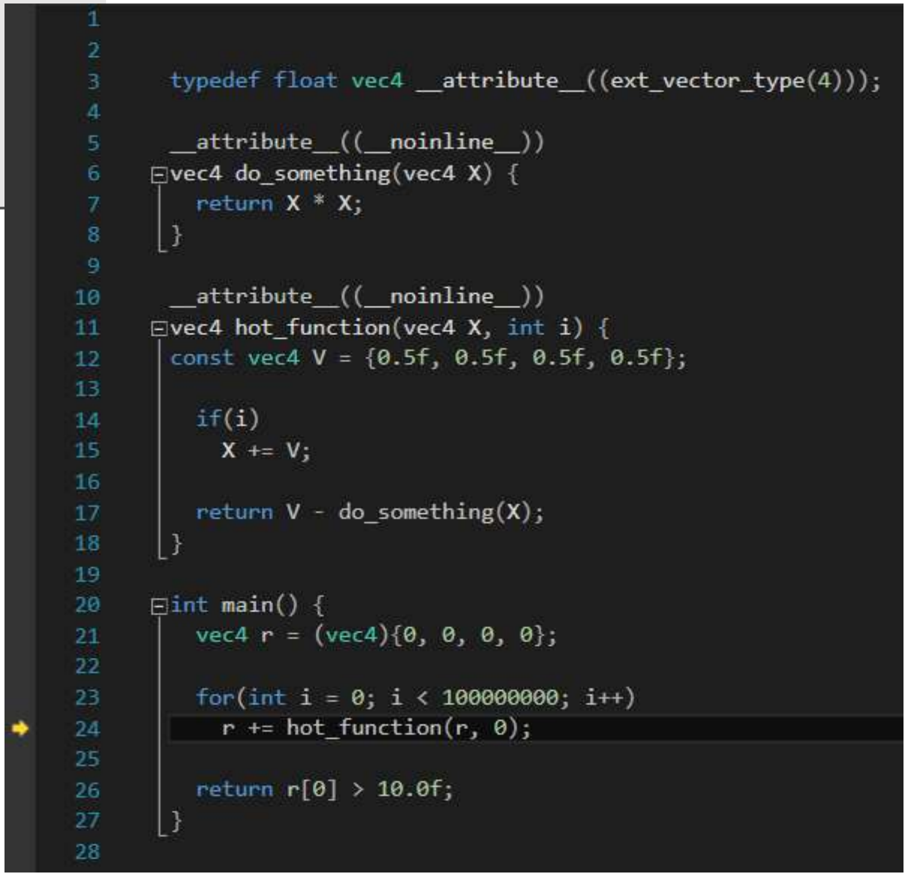
-O2 -g



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

-O2 -g

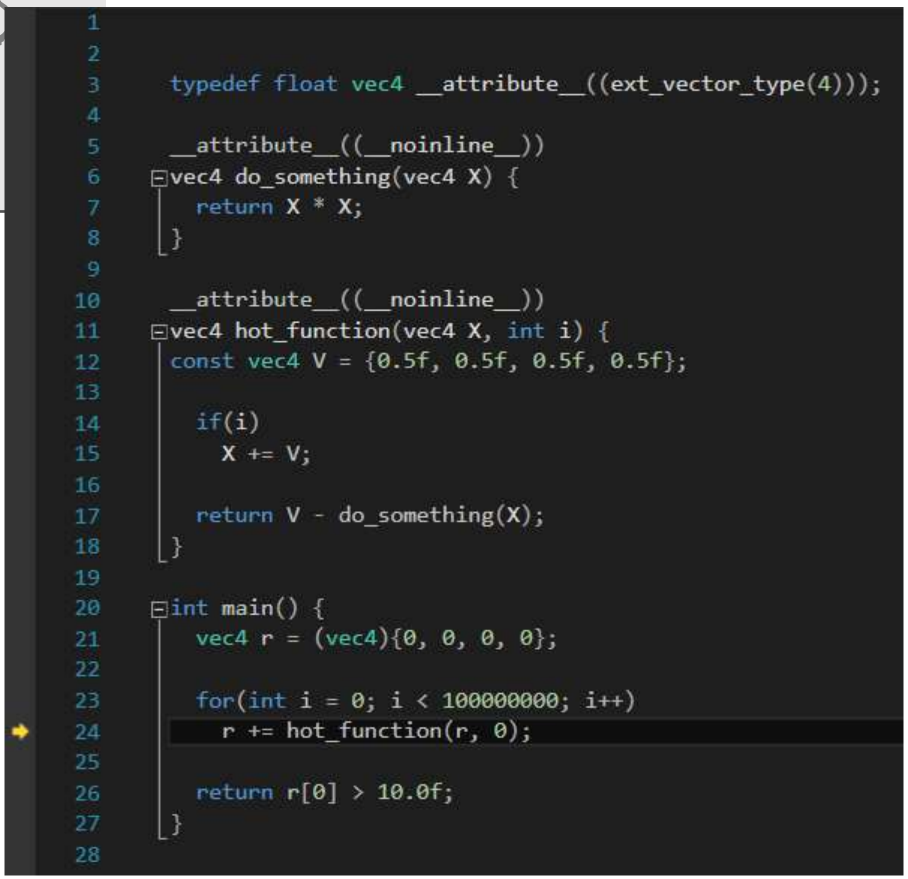
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	



```
1
2
3    typedef float vec4 __attribute__((ext_vector_type(4)));
4
5    __attribute__((__noinline__))
6   vec4 do_something(vec4 X) {
7      return X * X;
8    }
9
10   __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12   const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14     if(i)
15       X += V;
16
17     return V - do_something(X);
18   }
19
20  int main() {
21     vec4 r = (vec4){0, 0, 0, 0};
22
23     for(int i = 0; i < 100000000; i++)
24       r += hot_function(r, 0);
25
26     return r[0] > 10.0f;
27   }
28
```

-O2 -g

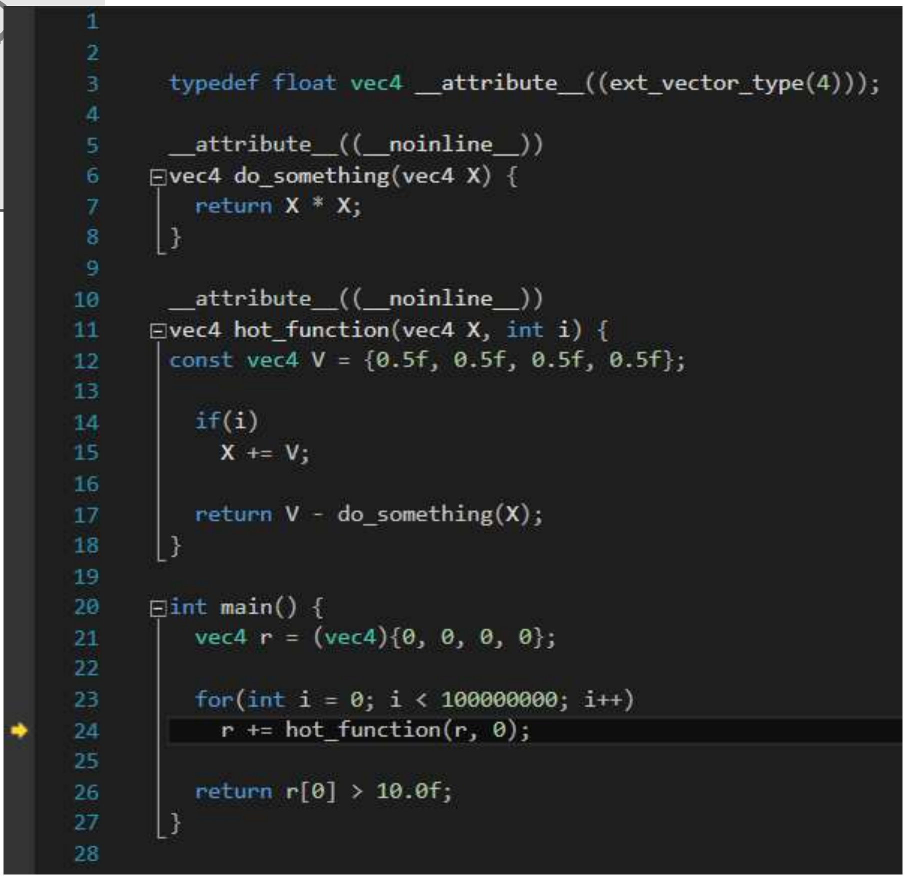
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	15:0
	vsubps %xmm0, %xmm1, %xmm0	17:0
	popq %rbp	
	retq	



```
1
2
3   typedef float vec4 __attribute__((ext_vector_type(4)));
4
5   __attribute__((__noinline__))
6   vec4 do_something(vec4 X) {
7     return X * X;
8   }
9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

-O2 -g

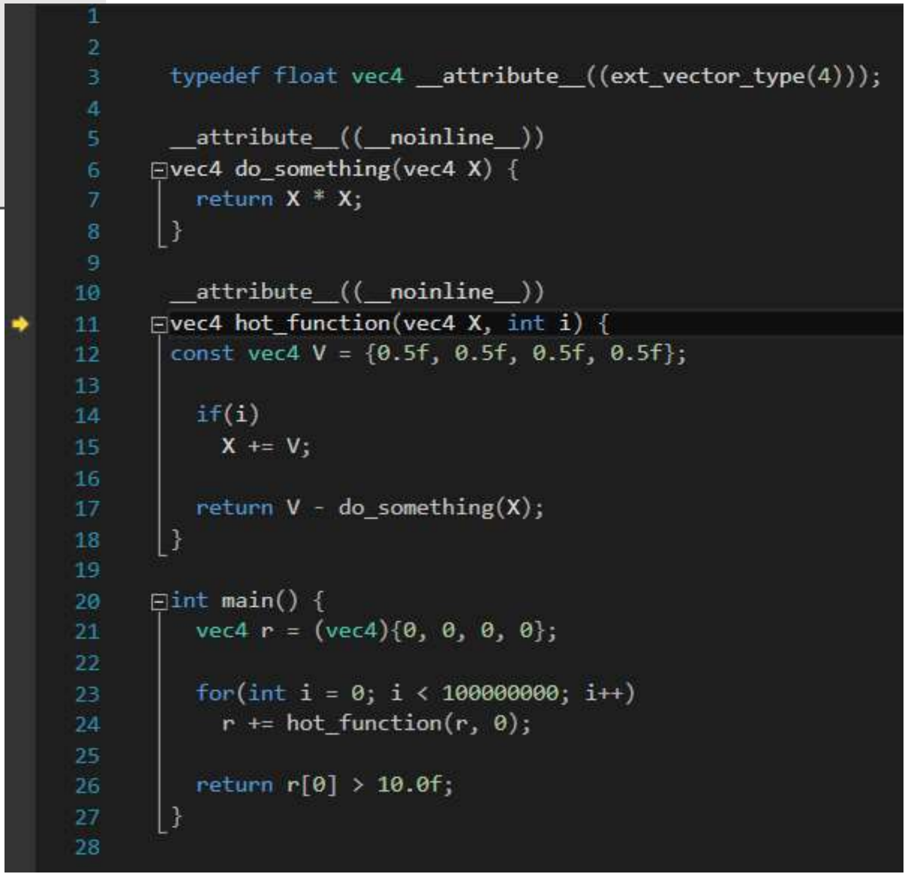
	Instruction	Debug Line
hot_function:	`testl %edi, %edi`	11:0
	`je .LBB1_2`	14:0
	`vaddps .LCPI1_0(%rip), %xmm0, %xmm0`	15:0
.LBB1_2:	`pushq %rbp`	0:0
	`movq %rsp, %rbp`	
	`callq do_something@PLT`	17:0
	`vmovups .LCPI1_0(%rip), %xmm1`	
	`vsubps %xmm0, %xmm1, %xmm0`	
	`popq %rbp`	
	`retq`	



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	`testl %edi, %edi`	11:0
	`je .LBB1_2`	14:0
	`vaddps .LCPI1_0(%rip), %xmm0, %xmm0`	15:0
.LBB1_2:	`pushq %rbp`	0:0
	`movq %rsp, %rbp`	
	`callq do_something@PLT`	17:0
	`vmovups .LCPI1_0(%rip), %xmm1`	
	`vsubps %xmm0, %xmm1, %xmm0`	
	`popq %rbp`	
	`retq`	

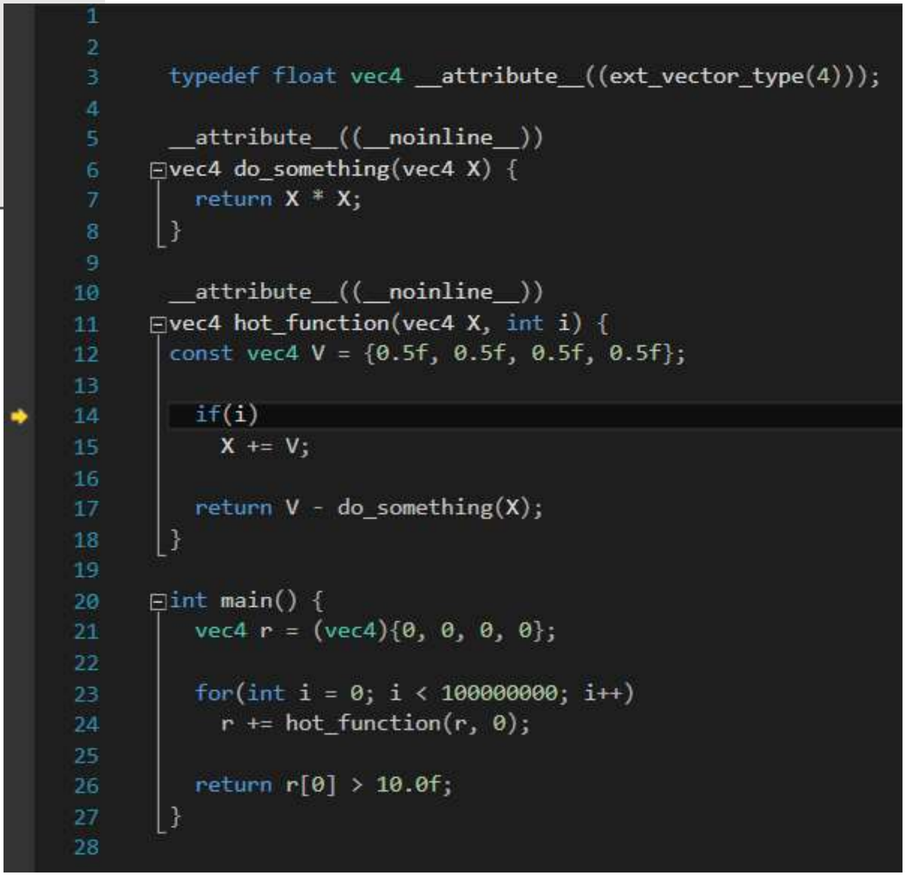
-O2 -g



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

-O2 -g

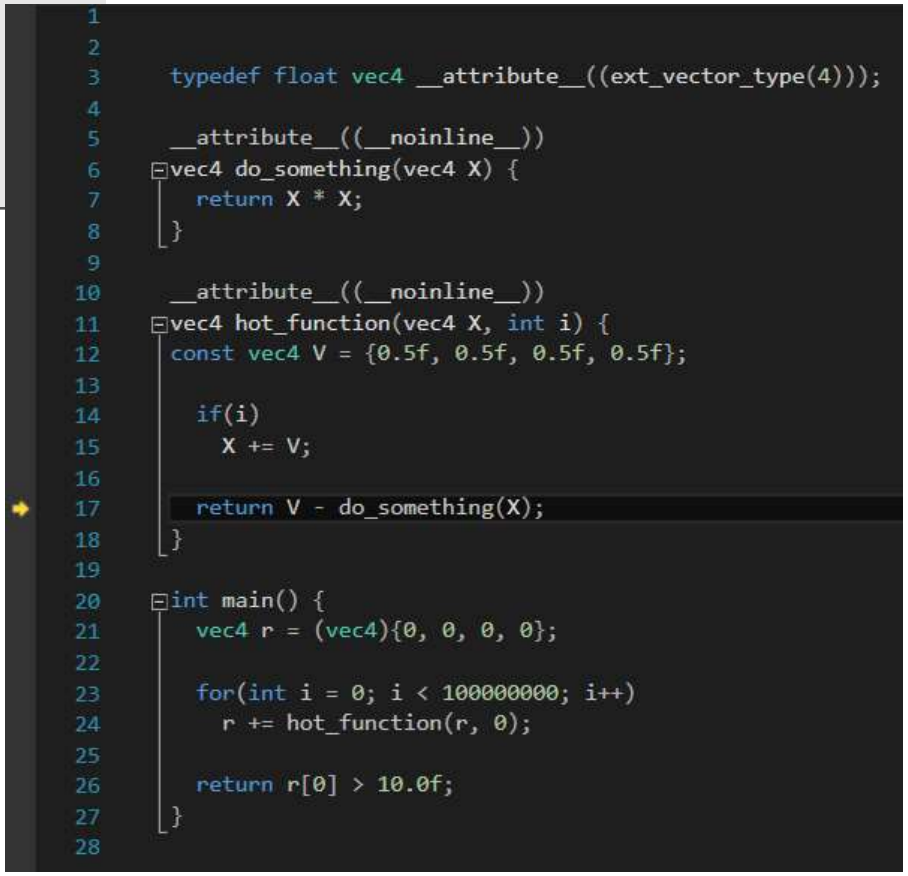
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	0:0
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	
	vsubps %xmm0, %xmm1, %xmm0	
	popq %rbp	
	retq	



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

-O2 -g

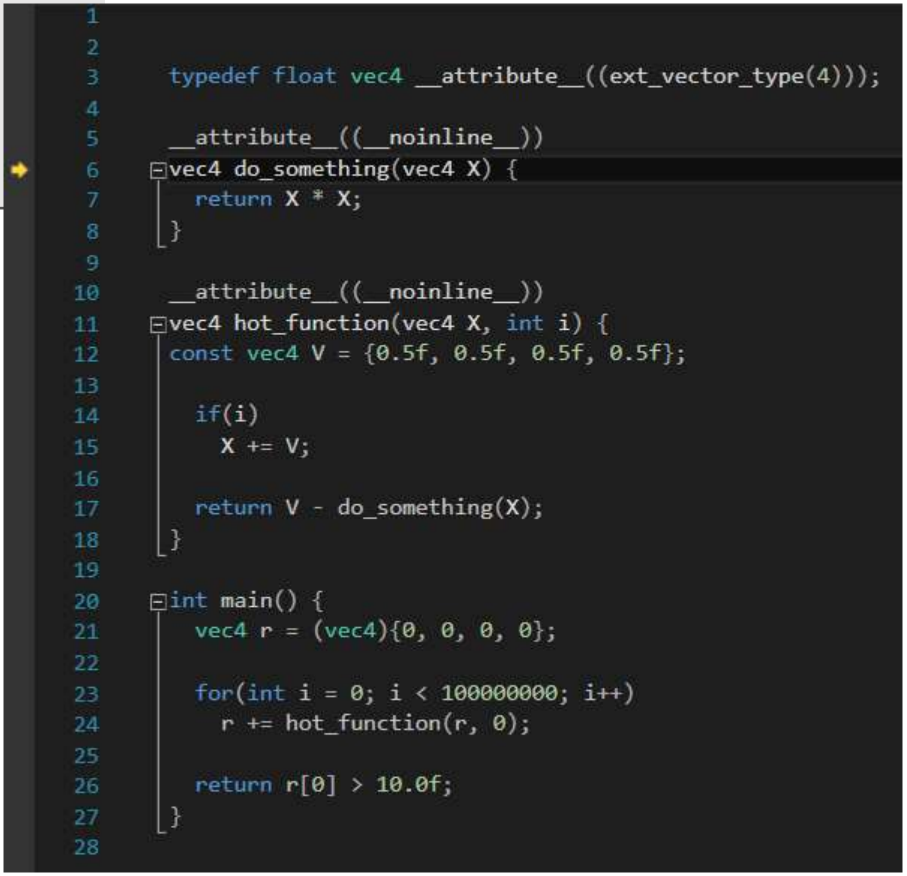
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	0:0
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	
	vsubps %xmm0, %xmm1, %xmm0	
	popq %rbp	
	retq	



```
 1
 2
 3  typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5  __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7    return X * X;
 8  }
 9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	0:0
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	
	vsubps %xmm0, %xmm1, %xmm0	
	popq %rbp	
	retq	

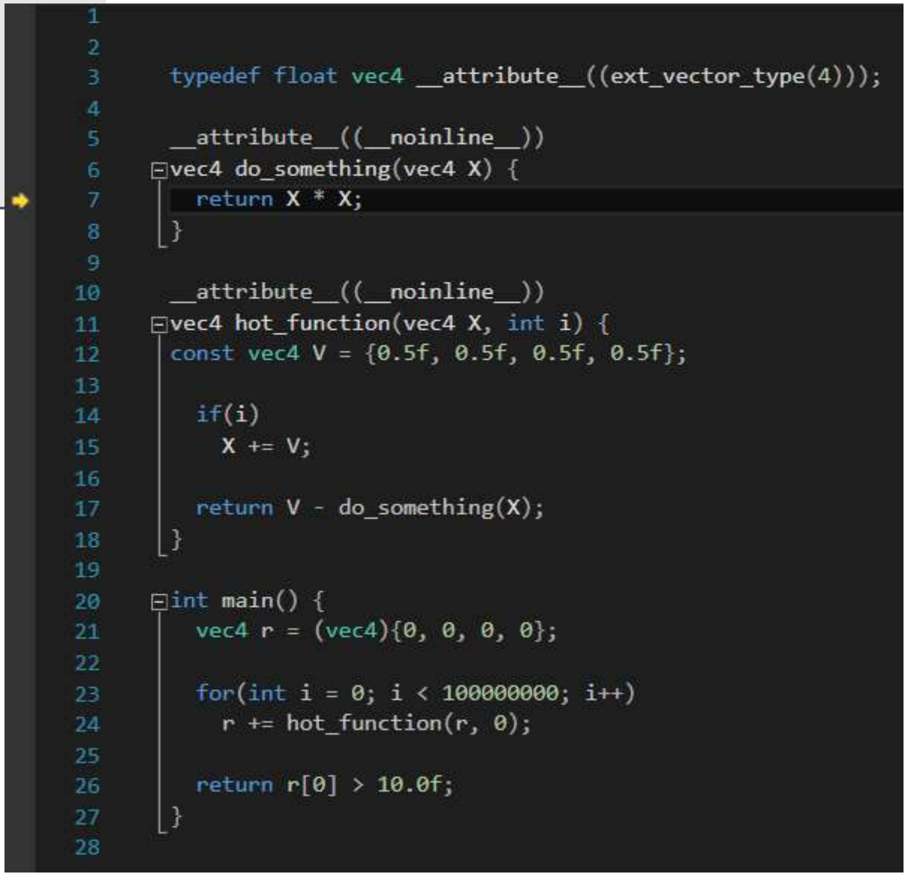
-O2 -g



```
1
2
3   typedef float vec4 __attribute__((ext_vector_type(4)));
4
5   __attribute__((__noinline__))
6   vec4 do_something(vec4 X) {
7     return X * X;
8   }
9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	0:0
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	
	vsubps %xmm0, %xmm1, %xmm0	
	popq %rbp	
	retq	

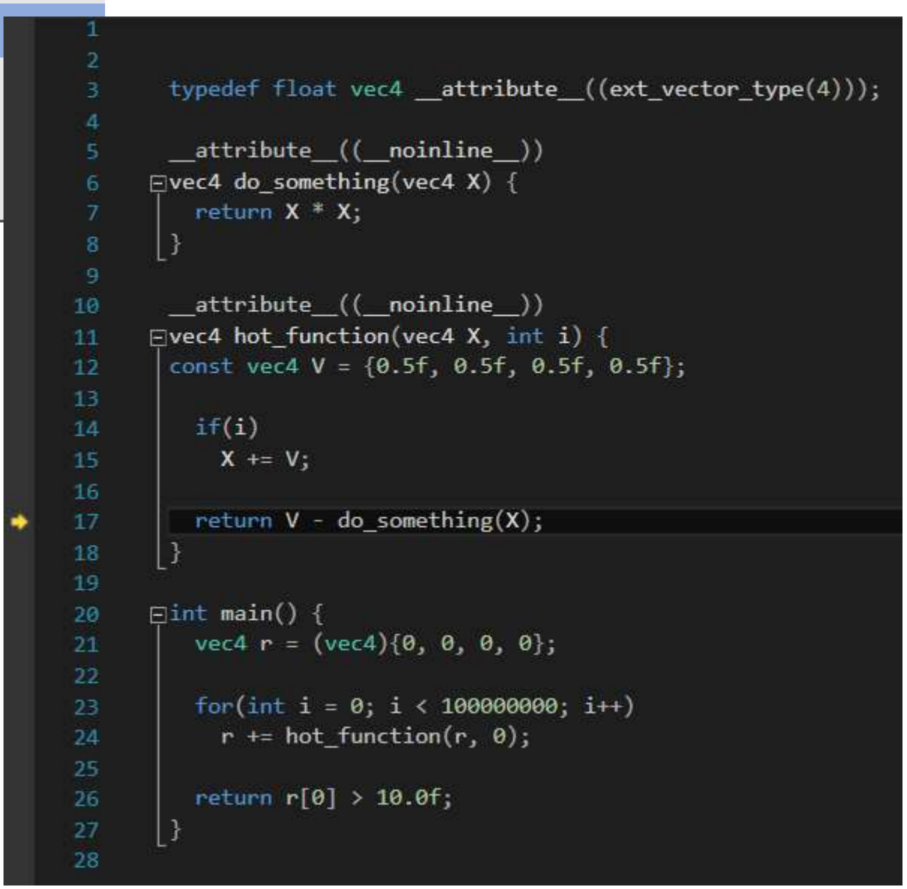
-O2 -g



```
 1
 2
 3   typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5   __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7     return X * X;
 8   }
 9
10   __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12   const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14     if(i)
15       X += V;
16
17     return V - do_something(X);
18   }
19
20  int main() {
21     vec4 r = (vec4){0, 0, 0, 0};
22
23     for(int i = 0; i < 100000000; i++)
24       r += hot_function(r, 0);
25
26     return r[0] > 10.0f;
27   }
28
```

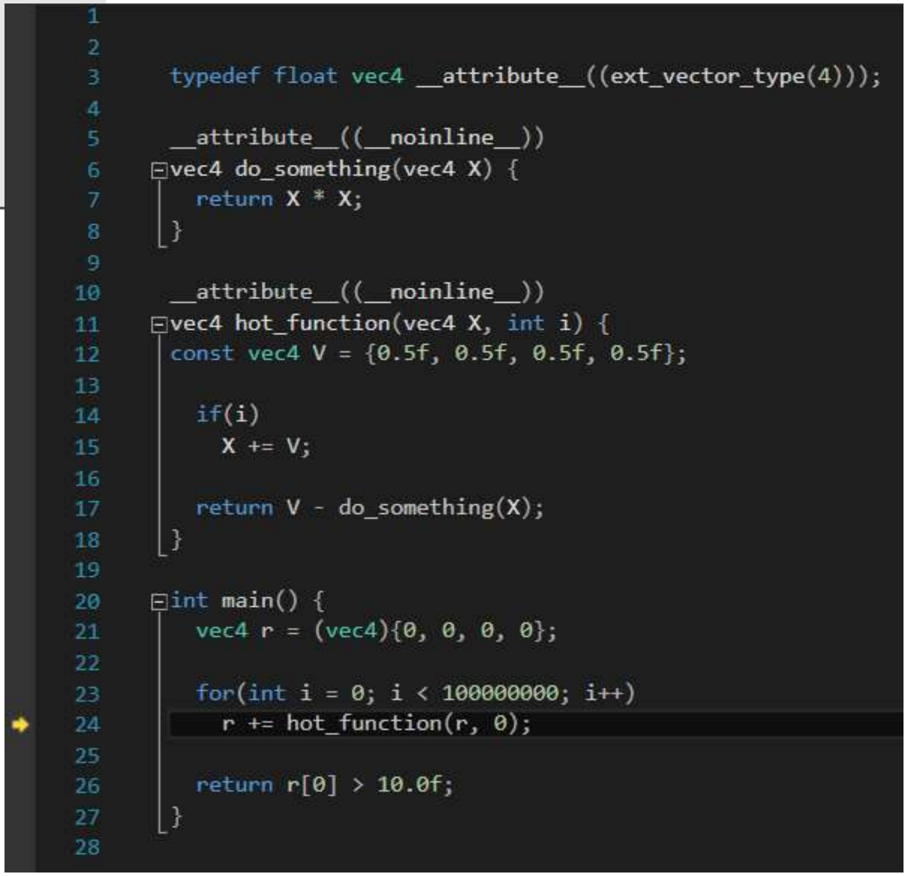
	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	0:0
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	
	vsubps %xmm0, %xmm1, %xmm0	
	popq %rbp	
	retq	

-O2 -g


```
1
2
3   typedef float vec4 __attribute__((ext_vector_type(4)));
4
5   __attribute__((__noinline__))
6   vec4 do_something(vec4 X) {
7     return X * X;
8   }
9
10  __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14    if(i)
15      X += V;
16
17    return V - do_something(X);
18  }
19
20  int main() {
21    vec4 r = (vec4){0, 0, 0, 0};
22
23    for(int i = 0; i < 100000000; i++)
24      r += hot_function(r, 0);
25
26    return r[0] > 10.0f;
27  }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	0:0
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	
	vsubps %xmm0, %xmm1, %xmm0	
	popq %rbp	
	retq	

-O2 -g



```
 1
 2
 3   typedef float vec4 __attribute__((ext_vector_type(4)));
 4
 5   __attribute__((__noinline__))
 6  vec4 do_something(vec4 X) {
 7     return X * X;
 8   }
 9
10   __attribute__((__noinline__))
11  vec4 hot_function(vec4 X, int i) {
12   const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14     if(i)
15       X += V;
16
17     return V - do_something(X);
18   }
19
20  int main() {
21     vec4 r = (vec4){0, 0, 0, 0};
22
23     for(int i = 0; i < 100000000; i++)
24       r += hot_function(r, 0);
25
26     return r[0] > 10.0f;
27   }
28
```

	Instruction	Debug Line
hot_function:	testl %edi, %edi	11:0
	je .LBB1_2	14:0
	vaddps .LCPI1_0(%rip), %xmm0, %xmm0	15:0
.LBB1_2:	pushq %rbp	0:0
	movq %rsp, %rbp	
	callq do_something@PLT	17:0
	vmovups .LCPI1_0(%rip), %xmm1	
	vsubps %xmm0, %xmm1, %xmm0	
	popq %rbp	
	retq	

-O2 -g

11	`vec4 hot_function(vec4 X, int i) {`	
12	`  const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};`	
13		
14	`  if(i)`	48897
15	`    X += V;`	49164
16		
17	`  return V - do_something(X);`	49592, calls: do_something:49402
18	`}`	

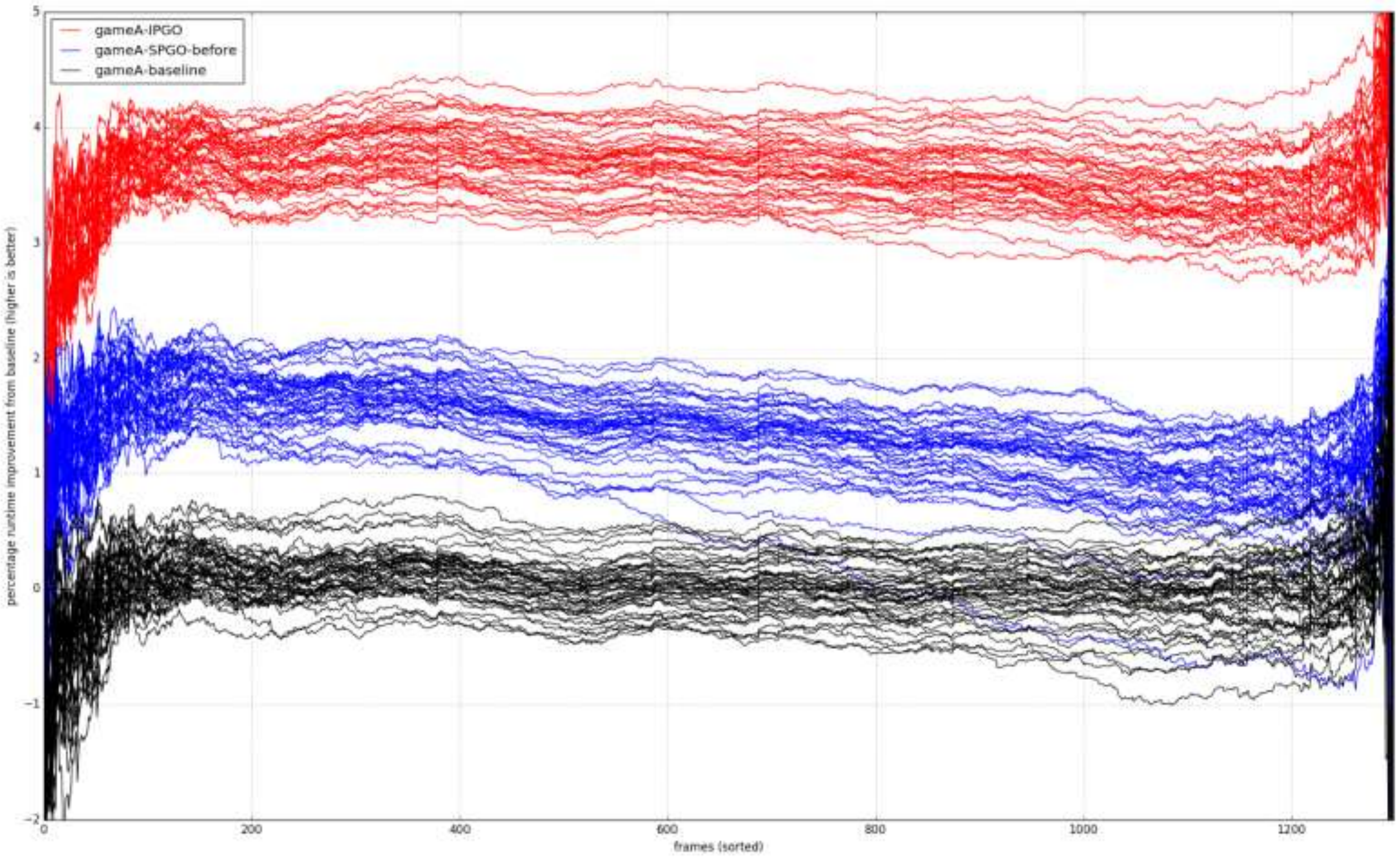
-O2 -g

	Instruction	Debug Line
hot_function:	`testl %edi, %edi`	11:0
	`je .LBB1_2`	14:0
	`vaddps .LCPI1_0(%rip), %xmm0, %xmm0`	15:0
.LBB1_2:	`pushq %rbp`	0:0
	`movq %rsp, %rbp`	
	`callq do_something@PLT`	17:0
	`vmovups .LCPI1_0(%rip), %xmm1`	
	`vsubps %xmm0, %xmm1, %xmm0`	
	`popq %rbp`	
	`retq`	

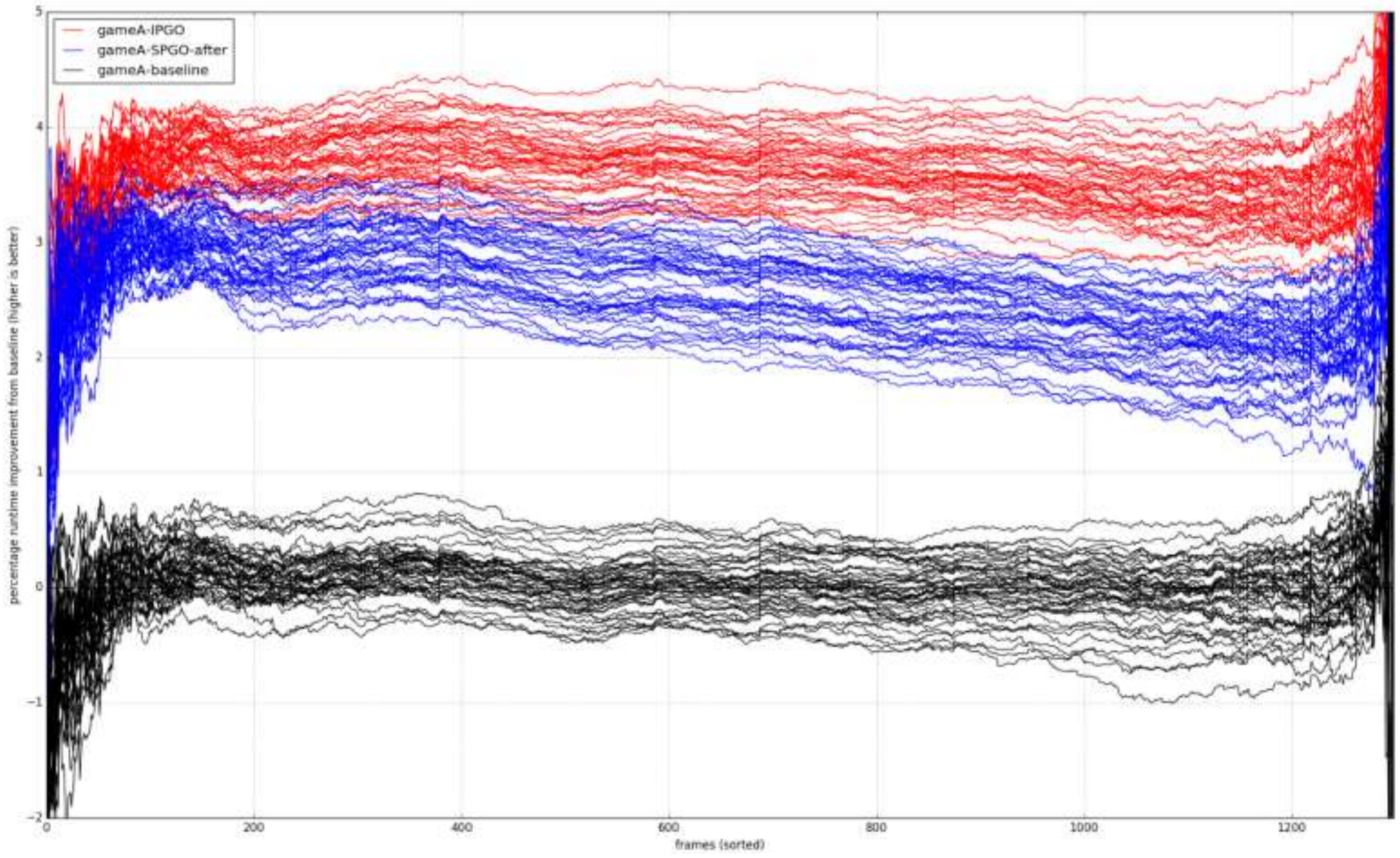
```
11    vec4 hot_function(vec4 X, int i) {
12      const vec4 V = {0.5f, 0.5f, 0.5f, 0.5f};
13
14      if(i)                                      49064
15        X += V;                                  0
16
17      return V - do_something(X);                49618, calls: do_something:49478
18    }
```

Current Results

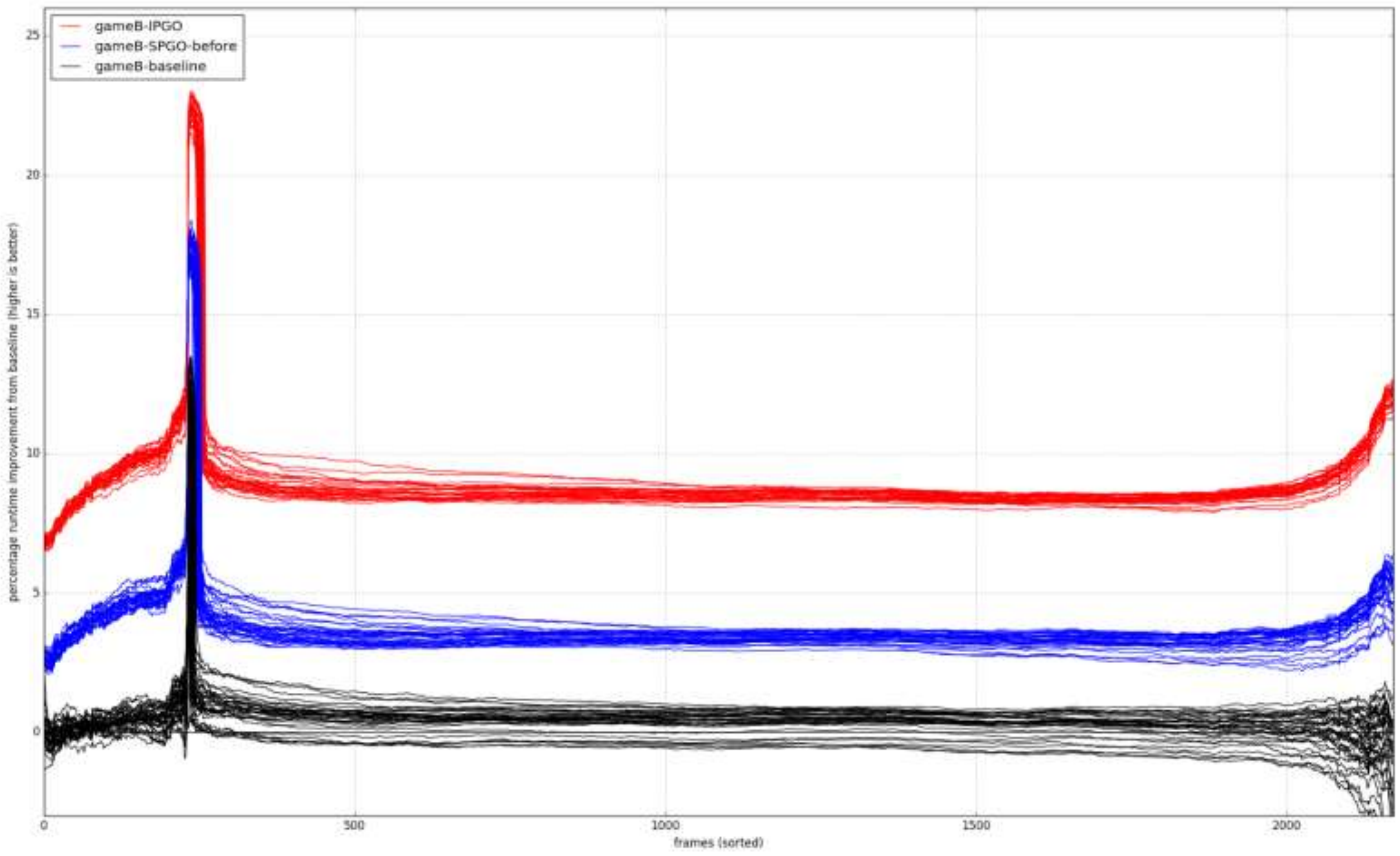
Frame time improvement (Game A)



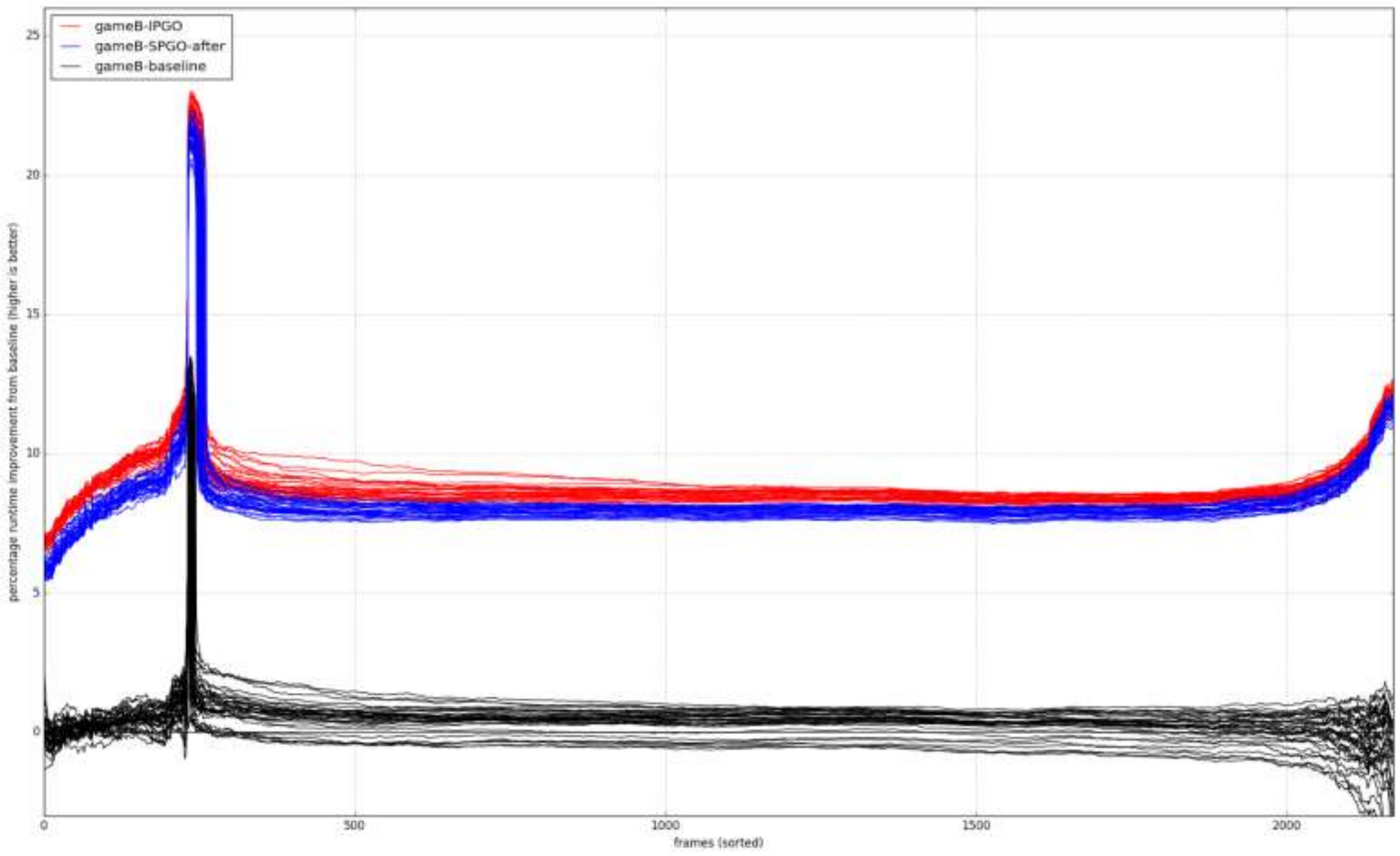
Frame time improvement (Game A)



Frame time improvement (Game B)



Frame time improvement (Game B)



Current Summary

- Sample-based PGO is now comparable to Instrumentation-based PGO in terms of performance improvement

  (and optimized debugging has improved too!)

- We now have two great technologies to offer our users
  - Sample-based PGO offers a lower barrier to entry and lower runtime intrusion when collecting profile data
  - Instrumentation-based PGO has an advantage in final performance
- **Work is still ongoing**
  - We think the gap can be closed further still

[side-note]

By tracking the runtime performance improvement of Sample-based PGO builds over time we can spot regressions in line-table data that affect the optimized debugging experience

(indirectly)

What we’ve learned

- When implementing new transformations:
  - **Consider the impact on the debug line table**
    - Especially when multiple basic blocks are involved
  - **Is it appropriate to retain the existing debug location?**
    - It’s often better to set line zero than to propagate line numbers incorrectly
    - Use the **`getMergedLocation`** API when merging common instructions

PlayStation®

sn systems