PIOTR PADLEWSKI
KRZYSZTOF PSZENICZNY

SOUND DEVIRTUALIZATION

WHAT ARE VIRTUAL CALLS

- Polymorphism in OOP

```
%vtable = load {…} %p
%vfun = load {…} %vtable
call {…} %vfun(args…)
```

DEVIRTUALIZATION

- Optimization changing virtual (indirect) calls to direct calls
- Important for performance:
  - more inlining
  - indirect calls are harder to predict
    - Spectre/Meltdown mitigation
- Security implications

DEVIRTUALIZATION BY FRONTEND

```
struct A {
  virtual void virt_meth();
};
void bar() {
  A a;
  a.virt_meth(); // devirtualized by the frontend
}
```

DEVIRTUALIZATION BY MIDDLE-END

```
void foo(A *a) {
  a->virt_meth();
}

void bar() {
  A a;
  // will be devirtualized after inlining
  foo(&a);
}
```

PROBLEM WITH EXTERNAL FUNCTIONS

```
void foo(A *a) {
  a->virt_meth();
}
void external_fun(A *a);
void bar() {
  A a;
  // assumes external_fun may clobber a's vptr
  external_fun(&a);
  foo(&a);
}
```

IT GETS EVEN WORSE

```
void bar() {
  auto a = new A;
  a->virt_meth();
  // can devirtualize only if the first call was
  // inlined
  a->virt_meth();
}
```

MARK VPTR AS INVARIANT

- !invariant.load would be sufficient for Java
- C++'s ctors/dtors/placement new/… require more tricks

```
void foo() {
  A *a = new A;
  A *b = new(a) B;
  b->virt_meth();
  a->virt_meth(); // undefined behavior
}
```

MARK VPTR AS INVARIANT

```
void A::virt_meth() {
 static_assert(sizeof(A) == sizeof(B));
 new(this) B;
}

auto *a = new A;
a->virt_meth();
a->virt_meth(); // Undefined behavior
```

OLD MODEL

- !invariant.group – delimitable !invariant.load
- llvm.invariant.group.barrier intrinsic
  - needs to be used whenever the dynamic type changes
  - stops !invariant.group optimizations
  - returns a new SSA value

OLD MODEL'S FLAW

```
void g() {
  A *a = new A;
  a->virt_meth();
  A *b = new(a) B;
  if (a == b)
    b->virt_meth();
}
```

- we could add barriers to the compared pointers
  barrier(a) == barrier(b)

NEW MODEL

NEW MODEL

- Think of pointers to dynamic objects as fat pointers
- Equip each pointer with optional virtual metadata (pun intended)
- Each !invariant.group load/store must read/write the value associated with the virtual metadata
- Comparison of objects' addresses must be done on raw pointers

LAUNDER.INVARIANT.GROUP INTRINSIC

- Creates a fat pointer with fresh virtual metadata
- Used: whenever the dynamic type could change
  - derived ctor/dtor
  - placement new
  - int to ptr
  - union members
  - std::launder

STRIP.INVARIANT.GROUP INTRINSIC

- Strips virtual metadata
- Pure (readnone) function
- Used: when we stop caring about the dynamic type
  - ptr to int
  - pointer comparisons
- Can be safely replaced with launder

INTRINSICS' PROPERTIES

- ```
  %b = launder(%a)
  %c = launder(%b)
  ; => %c = launder(%a)
  ```
- ```
  %b = launder(%a)
  %c = strip(%b)
  ; => %c = strip(%a)
  ```
- ```
  %b = strip(%a)
  %c = launder(%b)
  ; => %c = launder(%a)
  ```
- ```
  %b = strip(%a)
  %c = strip(%b)
  ; => %c = strip(%a) => %b
  ```

- Returned pointer aliases the argument
- Both intrinsics can be removed if the result is unused

STRIP.INVARIANT.GROUP INTRINSIC

- Can propagate equality

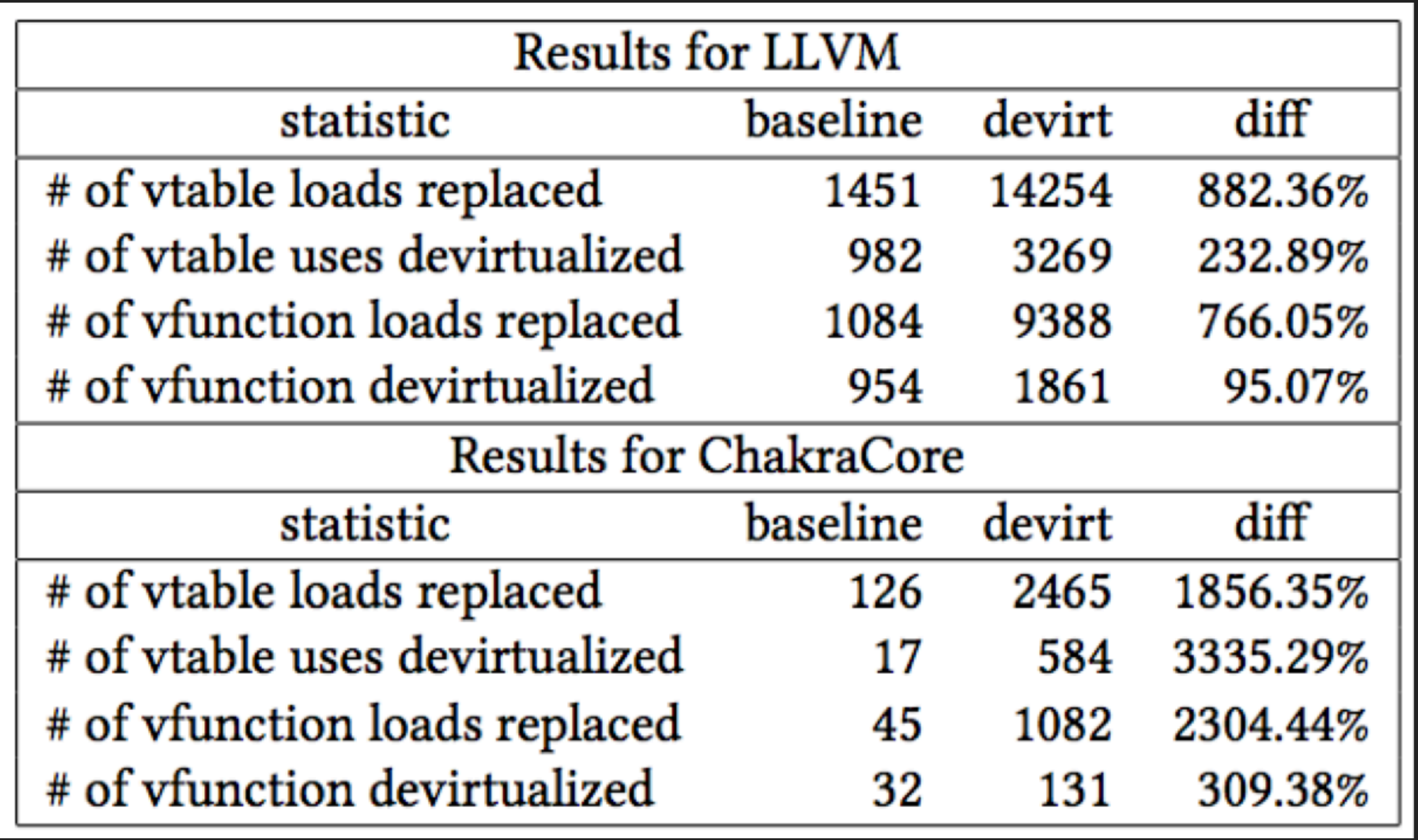
```
auto *a = new A;
a == a


%a1 = strip(%a)
%a2 = strip(%a)
; optimizes to true
%b = icmp eq %a1, %a2
```

- Even when the dynamic type changes

```
auto *a = new A;
auto *b = new(a) B;
a == b;
std::launder(a) == b;
```

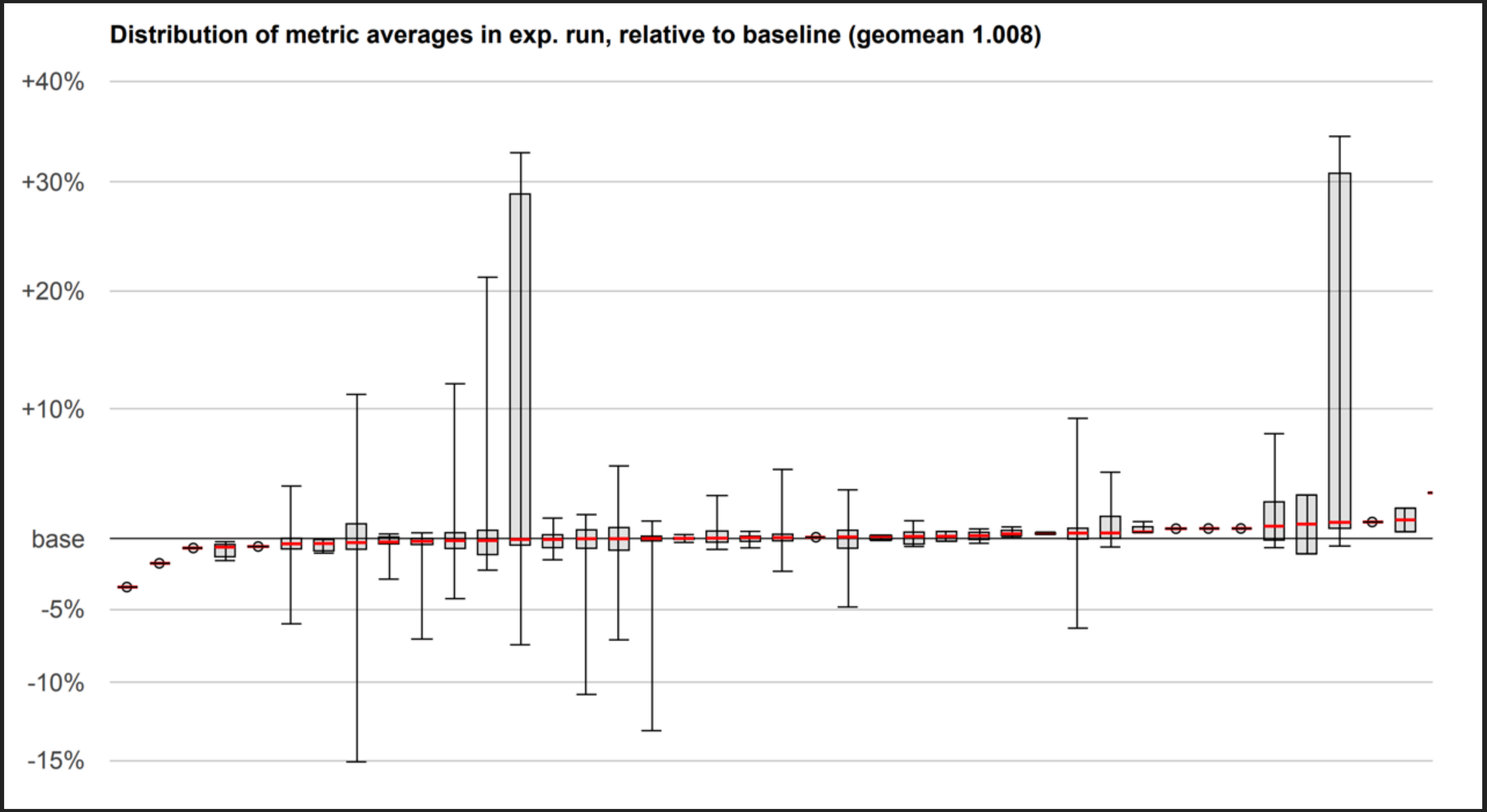
EXPERIMENTAL RESULTS

OPTIMIZATIONS STATISTICS (OLD MODEL)

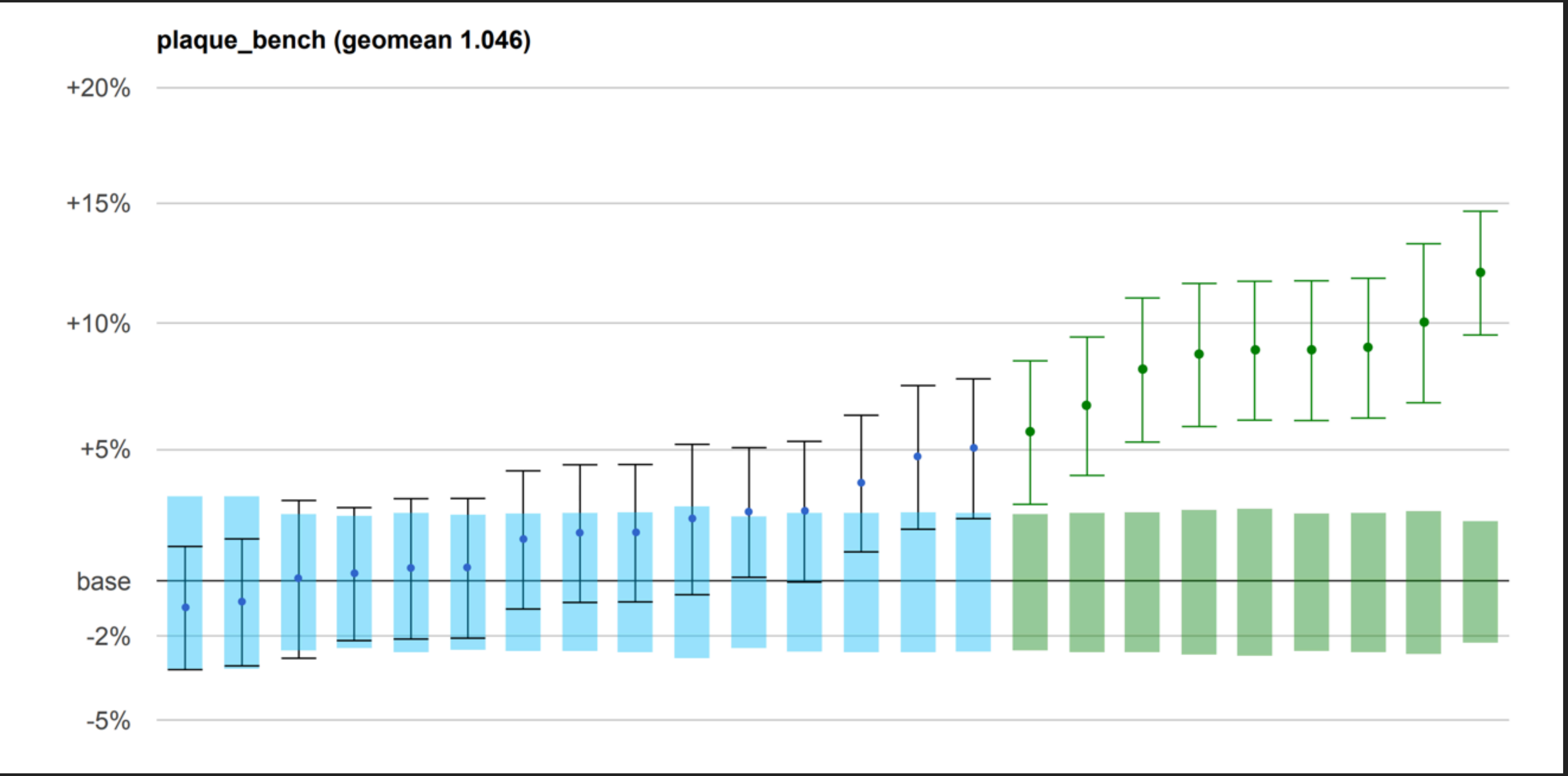


Results for LLVM			
statistic	baseline	devirt	diff
# of vtable loads replaced	1451	14254	882.36%
# of vtable uses devirtualized	982	3269	232.89%
# of vfunction loads replaced	1084	9388	766.05%
# of vfunction devirtualized	954	1861	95.07%
Results for ChakraCore			
statistic	baseline	devirt	diff
# of vtable loads replaced	126	2465	1856.35%
# of vtable uses devirtualized	17	584	3335.29%
# of vfunction loads replaced	45	1082	2304.44%
# of vfunction devirtualized	32	131	309.38%

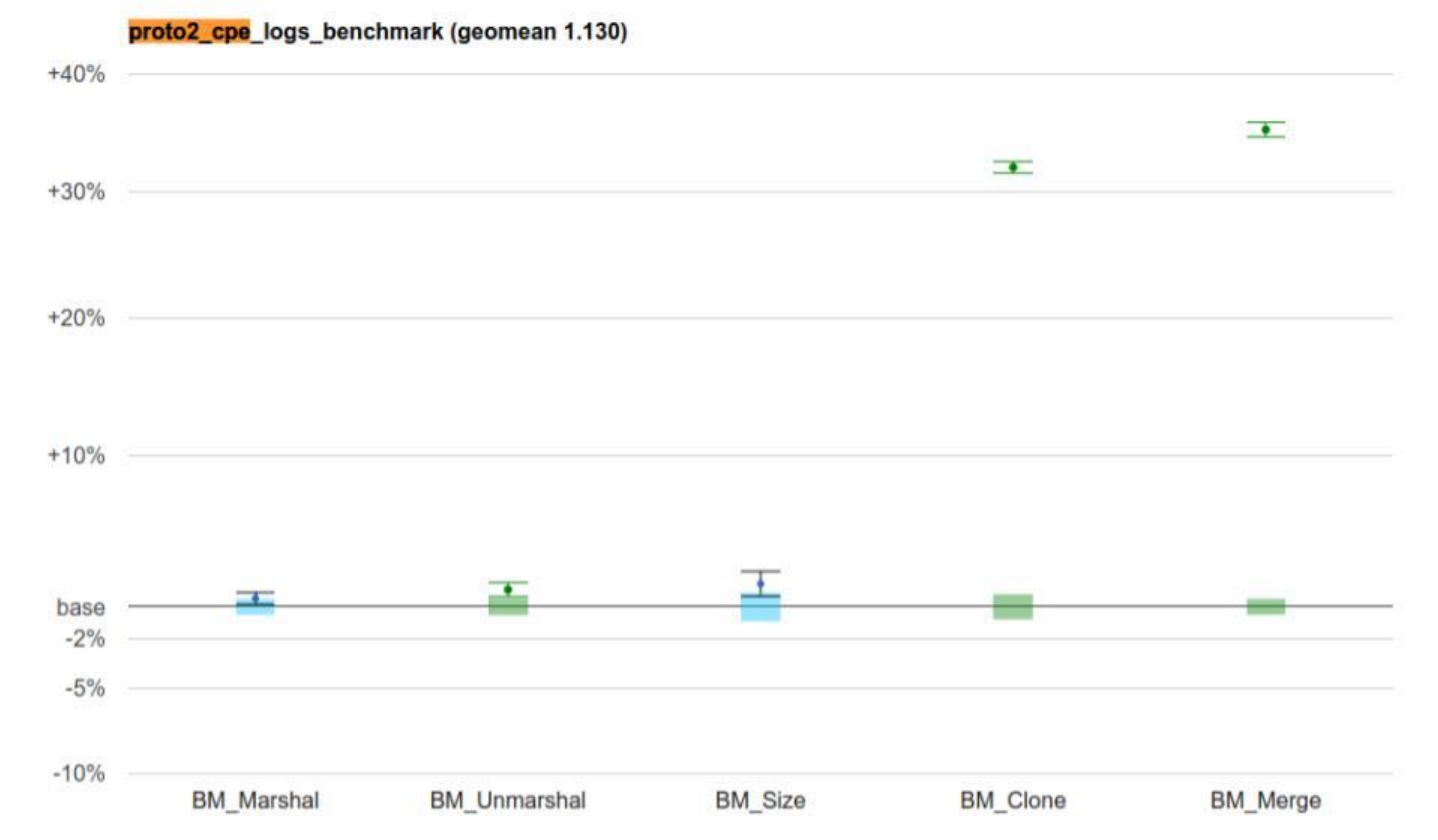
BENCHMARKS

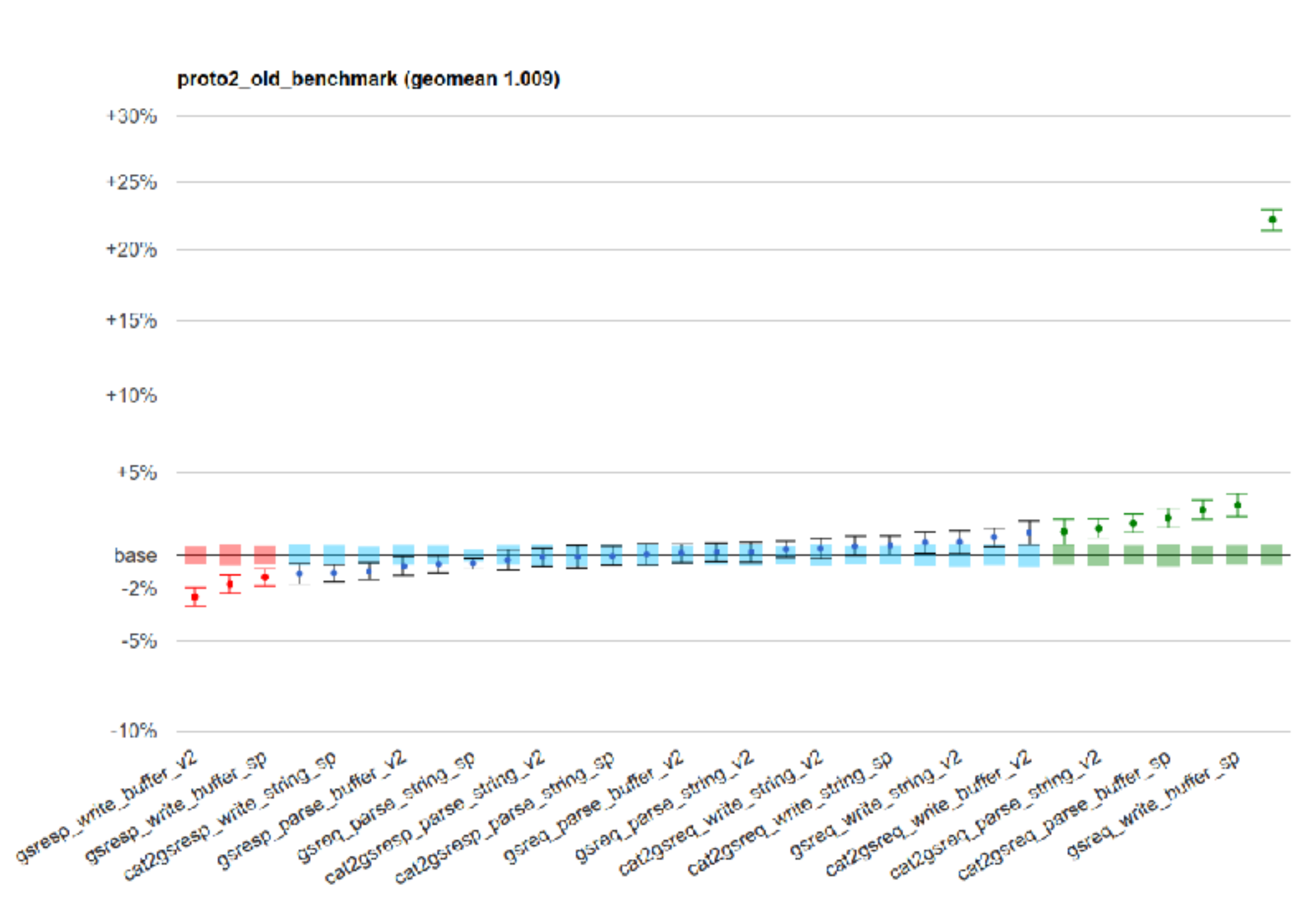

Distribution of metric averages in exp. run, relative to baseline (geomean 1.008)
+40%
+30%
+20%
+10%
base
-5%
-10%
-15%

BENCHMARKS



BENCHMARKS


proto2_cpe_logs_benchmark (geomean 1.130)
+40%
+30%
+20%
+10%
base
-2%
-5%
-10%
BM_Marshal
BM_Unmarshal
BM_Size
BM_Clone
BM_Merge


proto2_old_benchmark (geomean 1.009)
+30%
+25%
+20%
+15%
+10%
+5%
base
-2%
-5%
-10%
gsresp_write_buffer_v2
gsresp_write_buffer_sp
cat2gsresp_write_string_sp
gsresp_parse_buffer_v2
gsreq_parse_string_sp
cat2gsresp_parse_string_v2
cat2gsresp_parse_string_sp
gsreq_parse_buffer_v2
gsreq_parse_string_v2
cat2gsreq_write_string_v2
cat2gsreq_write_string_sp
gsreq_write_string_v2
cat2gsreq_write_buffer_v2
cat2gsreq_parse_string_v2
cat2gsreq_parse_buffer_sp
gsreq_write_buffer_sp

OTHER BENCHMARKS

- Google Search benchmarks showed 0.65% improvement (without FDO)
- Spec2006 didn't show any difference
- 7zip and zippy benchmarks showed 0.6% improvement before fixing the inliner
  - after fixing the inliner, there was no change for 7zip and zippy regressed
  - requires further investigation

WHEN ARE WE GETTING DEVIRTUALIZATION?

- We need a way to perform safe optimizations between modules compiled with and without devirtualization
    - RFC soon
- We hope the next release will have it turned on by default :)

FURTHER WORK

- Clang's new experimental flag -fforce-emit-vtables
- Calling one virtual function from another will not be devirtualized unless the latter is inlined or final
  - Emit a called-through-vtable specialization of every method (possibly duplicating it for derived types)
  - Perform explicit direct calls (`a->A::virt_meth()`) to virtual methods in the usual way