Item 203
git_comments:

1. Generate a kernel given a templated functor for base binary types. Generates a single kernel for binary/string and large binary / large string. If your kernel implementation needs access to the specific type at compile time, please use BaseBinarySpecific.
2. **comment:** NOTE: This code is not currently used by any kernels and has suboptimal performance because it's recomputing the validity bitmap that is already computed by the kernel execution layer. Consider writing a lower-level "output adapter" for base binary types. namespace codegen
   label: code-design
3. END of kernel generator-dispatchers --
4. BEGIN of kernel generator-dispatchers ("GD") These GD functions instantiate kernel functor templates and select one of the instantiated kernels dynamically based on the data type or Type::type id that is passed. This enables functions to be populated with kernels by looping over vectors of data types rather than using macros or other approaches. The kernel functor must be of the form: template <typename Type0, typename Type1, Args...> struct FUNCTOR { static void Exec(KernelContext* ctx, const ExecBatch& batch, Datum* out) { // IMPLEMENTATION } }; When you pass FUNCTOR to a GD function, you must pass at least one static type along with the functor -- this is often the fixed return type of the functor. This Type0 argument is passed as the first argument to the functor during instantiation. The 2nd type passed to the functor is the DataType subclass corresponding to the type passed as argument (not template type) to the function. For example, GenerateNumeric<FUNCTOR, Type0>(int32()) will select a kernel instantiated like FUNCTOR<Type0, Int32Type>. Any additional variadic template arguments will be passed as additional template arguments to the kernel template. Convenience so we can pass DataType or Type::type for the GD's GD for numeric types (integer and floating point)
5. **comment:** See BaseBinary documentation
   label: documentation
6. namespace
7. Time32 and Time64
8. Duration
9. Add timestamp kernels
10. Implement Less, LessEqual by flipping arguments to Greater, GreaterEqual
11. size_is_bytes=
12. Create view of data as the type (e.g. timestamp)

git_commits:

1. **summary:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   message: ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time. Closes #7461 from wesm/ARROW-8969 Lead-authored-by: Wes McKinney <wesm@apache.org> Co-authored-by: Antoine Pitrou <antoine@python.org> Signed-off-by: Wes McKinney <wesm@apache.org>
   label: code-design

github_issues:

github_issues_comments:

github_pulls:

1. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
2. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
3. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
4. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
5. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
   label: code-design
6. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string

data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

7. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
8. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
9. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
   label: code-design
10. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
11. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
   label: code-design
12. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
   body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On

the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

13. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
14. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
15. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
16. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
17. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
18. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for

primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

19. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
20. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
21. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
22. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
23. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
    label: code-design
24. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators

body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
label: code-design

25. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
26. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
27. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
    label: code-design
28. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
29. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string

data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

30. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
31. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
32. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
label: code-design
33. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
label: code-design
34. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
35. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On

the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

36. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
37. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
    label: code-design
38. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
39. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
40. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
    label: code-design
41. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators

body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

42. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
43. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
44. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
45. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.
    label: code-design
46. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

47. **title:** ARROW-8969: [C++] Reduce binary size of kernels/scalar_compare.cc.o by reusing more kernels between types, operators
    body: With clang-8 on Linux this unit is now 654KB down from 1257KB. A few strategies: * Use same binary code for Less/Greater and LessEqual/GreaterEqual with arguments flipped * Reuse kernels for primitive types represented as integers * Reuse kernels for binary/string and largebinary/largestring On the latter matter, I had to rewrite some of the codegen_internal.h routines where they assumed they would receive the specific logical type as the compile-time parameter (possibly causing a conflict when trying to put a BinaryArray box around a StringArray's ArrayData). This makes kernel execution against string data faster in the places where `ArrayIterator` is used. Will run benchmarks As an aside, in general these kernel code generators will likely have lower and lower level code over time.

github_pulls_comments:

1. @ursabot benchmark --benchmark_filter=Greater 18e559b
2. ``` no such option: --benchmark_filter ```
3. @ursabot benchmark --benchmark-filter=Greater 18e559b
4. https://issues.apache.org/jira/browse/ARROW-8969
5. [AMD64 Ubuntu 18.04 C++ Benchmark (#112653)](https://ci.ursalabs.org/#builders/73/builds/76) builder has been succeeded. Revision: 53671af32c338fcca1edc40732c4c5fd1ad7585e ```diff ====================================== ================== ================== ======== benchmark baseline contender change ====================================== ================== ================== ======== GreaterArrayArrayString/32768/2 53.657m items/sec 94.257m items/sec 75.665% - GreaterArrayArrayInt64/32768/2 363.591m items/sec 337.022m items/sec -7.307% GreaterArrayScalarInt64/32768/100 598.659m items/sec 701.841m items/sec 17.236% GreaterArrayScalarString/32768/0 341.707m items/sec 463.243m items/sec 35.567% GreaterArrayArrayString/32768/100 54.363m items/sec 75.156m items/sec 38.249% - GreaterArrayArrayInt64/32768/1 364.615m items/sec 337.861m items/sec -7.338% - GreaterArrayArrayInt64/32768/0 365.779m items/sec 338.697m items/sec -7.404% GreaterArrayArrayString/32768/10000 54.734m items/sec 75.788m items/sec 38.465% GreaterArrayScalarString/32768/1 364.271m items/sec 569.761m items/sec 56.411% GreaterArrayScalarInt64/32768/2 596.674m items/sec 681.189m items/sec 14.164% - GreaterArrayArrayInt64/32768/10000 364.433m items/sec 338.039m items/sec -7.242% GreaterArrayArrayString/32768/1 127.275m items/sec 195.144m items/sec 53.325% GreaterArrayScalarInt64/32768/10 614.042m items/sec 697.835m items/sec 13.646% GreaterArrayScalarInt64/32768/1 613.585m items/sec 692.598m items/sec 12.877% - GreaterArrayArrayInt64/32768/100 360.385m items/sec 337.343m items/sec -6.394% GreaterArrayScalarString/32768/100 331.392m items/sec 447.519m items/sec 35.042% GreaterArrayScalarInt64/32768/0 624.860m items/sec 705.244m items/sec 12.864% - GreaterArrayArrayInt64/32768/10 363.073m items/sec 337.156m items/sec -7.138% GreaterArrayScalarInt64/32768/10000 608.171m items/sec 678.010m items/sec 11.483% GreaterArrayScalarString/32768/10 256.765m items/sec 352.403m items/sec 37.247% GreaterArrayScalarString/32768/10000 335.759m items/sec 461.788m items/sec 37.535% GreaterArrayArrayString/32768/0 54.130m items/sec 75.873m items/sec 40.168% GreaterArrayScalarString/32768/2 130.823m items/sec 194.535m items/sec 48.701% GreaterArrayArrayString/32768/10 51.190m items/sec 71.220m items/sec 39.127% ====================================== ================== ================== ======== ```
6. **body:** Not sure why some of the Int64 benchmarks are slower (AFAIK nothing changed about these so it may be noise) but the string comparisons got a lot faster. I had to engage in some slight shenanigans to make temporal Scalar types have a common base type so that we can use e.g. `PrimitiveScalar<Int64Scalar>` to safely unbox a `TimestampScalar`. This involved adding a IMHO overdue `StorageType` attribute to several primitive type objects. Otherwise it's difficult to use an Int64 kernel to process Timestamp, Time64, Duration data, etc. Please take a look @bkietz or @pitrou
   label: code-design
7. There weren't any C++ unit tests for comparisons of primitive types so I addressed that, and also added comparisons for Time and Duration types (which on account of this patch are basically "free")
8. @ursabot benchmark --benchmark-filter=Greater 18e559b
9. [AMD64 Ubuntu 18.04 C++ Benchmark (#112703)](https://ci.ursalabs.org/#builders/73/builds/77) builder has been succeeded. Revision: 301ffa539e634f2c464ca072cd5c543f1407f1f7 ```diff

==================================== ================== ================== ======== benchmark baseline contender change ==================================== ================== ================== ======== GreaterArrayArrayString/32768/2 53.694m items/sec 93.892m items/sec 74.864% GreaterArrayScalarInt64/32768/2 616.025m items/sec 681.971m items/sec 10.705% GreaterArrayScalarString/32768/2 132.267m items/sec 216.365m items/sec 63.583% - GreaterArrayArrayInt64/32768/10000 365.572m items/sec 339.713m items/sec -7.074% GreaterArrayArrayString/32768/100 53.635m items/sec 74.222m items/sec 38.384% - GreaterArrayArrayInt64/32768/100 365.263m items/sec 339.865m items/sec -6.953% GreaterArrayScalarString/32768/10000 341.705m items/sec 547.337m items/sec 60.178% - GreaterArrayArrayInt64/32768/1 365.608m items/sec 340.598m items/sec -6.841% GreaterArrayArrayString/32768/0 55.046m items/sec 74.164m items/sec 34.733% GreaterArrayScalarInt64/32768/10000 551.861m items/sec 679.554m items/sec 23.139% GreaterArrayScalarString/32768/1 360.301m items/sec 724.313m items/sec 101.030% GreaterArrayScalarInt64/32768/10 618.148m items/sec 669.015m items/sec 8.229% - GreaterArrayArrayInt64/32768/2 365.552m items/sec 340.107m items/sec -6.961% GreaterArrayScalarString/32768/100 332.135m items/sec 529.815m items/sec 59.518% - GreaterArrayArrayInt64/32768/0 367.549m items/sec 341.912m items/sec -6.975% - GreaterArrayArrayInt64/32768/10 365.734m items/sec 339.651m items/sec -7.131% GreaterArrayScalarInt64/32768/100 604.015m items/sec 667.332m items/sec 10.483% GreaterArrayScalarString/32768/10 258.597m items/sec 411.304m items/sec 59.052% GreaterArrayScalarInt64/32768/0 622.436m items/sec 671.837m items/sec 7.937% GreaterArrayArrayString/32768/1 130.420m items/sec 196.548m items/sec 50.704% GreaterArrayScalarInt64/32768/1 622.602m items/sec 699.200m items/sec 12.303% GreaterArrayArrayString/32768/10 51.206m items/sec 70.119m items/sec 36.935% GreaterArrayScalarString/32768/0 337.556m items/sec 549.035m items/sec 62.650% GreaterArrayArrayString/32768/10000 54.218m items/sec 74.885m items/sec 38.117% ==================================== ================== ================== ======== ```

10. **body:** Ah! It's because Greater is now implemented using Less. Let me switch things around so things are based on Greater/GreaterEqual instead
    label: code-design
11. @ursabot benchmark --benchmark-filter=Greater 18e559b
12. **body:** Not to beat a dead horse about ARROW-9155, but the turnaround time for simple benchmarks isn't great
    label: code-design
13. To help follow along it would be handy if references to JIRA could be autolinked: https://help.github.com/en/github/administering-a-repository/configuring-autolinks-to-reference-external-resources That would save the lurker from having to search for it or the poster having to bother about pasting an actual URL.
14. Well we have these bot comments, is it not sufficient? https://github.com/apache/arrow/pull/7461#issuecomment-645086851
15. [AMD64 Ubuntu 18.04 C++ Benchmark (#112729)](https://ci.ursalabs.org/#builders/73/builds/78) builder has been succeeded. Revision: 74caaae25e3bd95d57f3f6d9b835c2610639ab41 ```diff ==================================== ================== ================== ======== benchmark baseline contender change ==================================== ================== ================== ======== GreaterArrayScalarString/32768/2 130.267m items/sec 224.418m items/sec 72.276% GreaterArrayScalarString/32768/10 261.021m items/sec 498.491m items/sec 90.977% - GreaterArrayArrayInt64/32768/2 361.493m items/sec 335.383m items/sec -7.223% GreaterArrayScalarInt64/32768/10 565.966m items/sec 690.931m items/sec 22.080% GreaterArrayScalarInt64/32768/100 555.331m items/sec 668.238m items/sec 20.331% - GreaterArrayArrayInt64/32768/10000 364.755m items/sec 338.493m items/sec -7.200% - GreaterArrayArrayInt64/32768/1 362.806m items/sec 336.071m items/sec -7.369% GreaterArrayScalarInt64/32768/1 573.242m items/sec 704.449m items/sec 22.889% GreaterArrayArrayString/32768/10000 54.921m items/sec 73.848m items/sec 34.463% GreaterArrayArrayString/32768/2 52.950m items/sec 92.643m items/sec 74.964% GreaterArrayScalarString/32768/1 361.437m items/sec 653.553m items/sec 80.821% GreaterArrayScalarInt64/32768/0 555.044m items/sec 701.556m items/sec 26.396% GreaterArrayScalarInt64/32768/10000 575.762m items/sec 682.164m items/sec 18.480% - GreaterArrayArrayInt64/32768/100 361.216m items/sec 338.015m items/sec -6.423%

GreaterArrayScalarString/32768/0 339.144m items/sec 707.998m items/sec 108.760% GreaterArrayArrayString/32768/1 127.597m items/sec 193.087m items/sec 51.326% GreaterArrayArrayString/32768/10 50.585m items/sec 69.612m items/sec 37.613% GreaterArrayArrayString/32768/100 54.563m items/sec 73.054m items/sec 33.889% GreaterArrayScalarInt64/32768/2 558.941m items/sec 697.482m items/sec 24.786% GreaterArrayScalarString/32768/100 331.214m items/sec 669.210m items/sec 102.048% - GreaterArrayArrayInt64/32768/10 361.791m items/sec 336.834m items/sec -6.898% GreaterArrayArrayString/32768/0 54.614m items/sec 73.456m items/sec 34.499% GreaterArrayScalarString/32768/10000 342.276m items/sec 705.190m items/sec 106.030% - GreaterArrayArrayInt64/32768/0 363.455m items/sec 336.738m items/sec -7.351% ===================================== ================== ================== ======== ```
16. Well my theory about greater/less didn't hold. The other relevant change was moving things into the anonymous namespace. It's possible that anonymous namespaces impact inlining somehow
17. > Not to beat a dead horse about ARROW-9155 The bot is fine - I guess it links to whatever JIRA is listed in the title. That doesn't help if someone mentions a JIRA in a comment. If the autolink was configured for this repo that reference would have *automatically* been converted to [`ARROW-9155`](https://issues.apache.org/jira/browse/ARROW-9155). It's just a *very* minor thing, but it does help the DX.
18. Can you bring it up somewhere else like on the mailing list? Someone can ask Apache INFRA to set this up
19. This can be reviewed, but I realized I can compress this further by reusing some equals/not_equal kernels, so I will quickly try to do that this morning
20. I'm revamping the documentation about these codegen functions which I'm dubbing "Generator-Dispatchers" (GDs) for short. I'll add "Generate" to their name. Stay tuned
21. +1. I'm going to merge this to help avoid conflicts caused by the stuff I just renamed. I welcome further comments and I will work to address them in follow ups
22. I gave up on trying to have e.g. a common "64-bit" kernel for Equals/NotEquals Int64/UInt64/Timestamp/etc. The sticking point is scalar unboxing. We might need to fashion a common internal base class for primitive types and give them a virtual method that returns their data as `const uint8_t*` that you can cast to whatever primitive type you want
23. I just requested JIRA reference autolinking https://issues.apache.org/jira/browse/INFRA-20450

github_pulls_reviews:

1. **body:** @pitrou anecdotally it appears this is meaningfully faster than the prior version which used `GetView`
   label: code-design
2. It would probably depend on the compiler? Logically, it's quite similar.
3. I'm not convinced why this is doing everything by hand instead of relying on the array builder as before.
4. **body:** What's this? How is it supposed to be used? `codegen_internal.h` quickly seems to become write-only: it's difficult to make sense of why those things exist and what they're good at.
   label: code-design
5. It doesn't make sense to try out a ton of different sizes (or probabilities, see below), IMHO.
6. I'll document it, sorry.
7. I'll remove this. I was copy and pasting what was already there
8. Well, the proof is in the benchmarks. I can start writing microbenchmarks for these internal utilities to help us avoid having arguments about them.
9. **body:** I don't think micro-benchmarks are useful here, as actual performance will depend on overall optimization of the calling function by the compiler. In any case, I don't dispute the numbers. But it also seems the improvements would be mostly compiler-dependent.
   label: code-design
10. **body:** Right well AFAICT this change is responsible for the 40-50% performance improvement. So I'm calling it out because there are other places where we use the GetView method that may therefore be performing suboptimally (even if only on some compilers). If it were only 5% then it would be inconclusive.
    label: code-design
11. FWIW, we have slight precedent for naming this "GenerateForPhysicalInteger": `type_traits.h::is_physical_integer_type`

12. ```suggestion // Implement Less, LessEqual by flipping arguments to Greater, GreaterEqual ```
13. ```suggestion ArrayKernelExec GenerateForBaseBinary(detail::GetTypeId get_id) { ```
14. **body:** It would improve readability if we formalized the `Op` concept and named them consistently. Even within this PR these are also called `Generator`s. It's not consistently made clear what functions they're expected to provide. https://issues.apache.org/jira/browse/ARROW-9161
    label: code-design
15. ```suggestion ```
16. ```suggestion using PhysicalType = Int64Type; ``` ?
17. **body:** I'm not sure this use of inheritance is kosher. This implies `TimestampScalar isa Int64Scalar`, which is a relationship not present in parallel hierarchies (Array, DataType, Builder) and is therefore likely to be confusing. Why not use `TemporalScalar : PrimitiveScalar<T>`?
    label: code-design
18. Int64Scalar and TimestampScalar need to have a common base type in order to both be unboxed by templated code that uses Int64Type in the kernel codegen. So `PrimitiveScalar<TimestampType>` won't work.
19. OK
20. Generator and Op are not the same thing in codegen_internal.h -- I agree with giving consistent names to isomorphic concepts, though.
21. The validity bitmap is already precomputed (zero-copied, in fact) by the executor so using the builder naively here is computationally wasteful
22. **body:** I'll move these "generator-dispatchers" to a `generate::` namespace for clarity. I thought that `codegen::` was clear enough but I guess not.
    label: code-design
23. I'm reverting this change since no kernel uses it and adding a comment explaining that it is suboptimal.

jira_issues:

1. **summary:** [C++] Reduce generated code in compute/kernels/scalar_compare.cc
   description: We are instantiating multiple versions of templates in this module for cases that, byte-wise, do the exact same comparison. For example: * For equals, not_equals, we can use the same 32-bit/64-bit comparison kernels for signed int / unsigned int / floating point types of the same byte width * TimestampType can reuse int64 kernels, similarly for other date/time types * BinaryType/StringType can share kernels etc.
2. **summary:** [C++] Reduce generated code in compute/kernels/scalar_compare.cc
   description: We are instantiating multiple versions of templates in this module for cases that, byte-wise, do the exact same comparison. For example: * For equals, not_equals, we can use the same 32-bit/64-bit comparison kernels for signed int / unsigned int / floating point types of the same byte width * TimestampType can reuse int64 kernels, similarly for other date/time types * BinaryType/StringType can share kernels etc.
3. **summary:** [C++] Reduce generated code in compute/kernels/scalar_compare.cc
   description: We are instantiating multiple versions of templates in this module for cases that, byte-wise, do the exact same comparison. For example: * For equals, not_equals, we can use the same 32-bit/64-bit comparison kernels for signed int / unsigned int / floating point types of the same byte width * TimestampType can reuse int64 kernels, similarly for other date/time types * BinaryType/StringType can share kernels etc.
   label: code-design

jira_issues_comments:

1. Issue resolved by pull request 7461 [https://github.com/apache/arrow/pull/7461]