Item 201
git_comments:

1.
2. This benchmarks acts as a reference to the native std::vector implementation. It appends kRounds chunks into a vector.
3. Insert an element late in the game that does not fit in the 8bit representation. This forces AdaptiveIntBuilder's to resize.
4. Make a vector out of `kDistinctElements` string values
5. Pattern matching that extracts the vector element type of Benchmark::Args() Benchmark changed its parameter type between releases from int to int64_t. As it doesn't have version macros, we need to apply C++ template magic.
6. Regression do not need to account for cache hierarchy, thus optimize for the best case.
7. ARROW_WITH_BENCHMARKS_REFERENCE
8. **comment:** Provides better regression stability with timings rather than bogus bandwidth.
   label: code-design
9. Linter stipulates: >> For a static/global string constant, use a C style string instead
10. ARROW_WITH_BENCHMARKS_REFERENCE
11. pretty
12. Trigger the slow path where the buffer is not byte aligned. NOLINT non-const reference
13. ARROW_WITH_BENCHMARKS_REFERENCE
14. ARROW_WITH_BENCHMARKS_REFERENCE
15. This benchmark simply provides a baseline indicating the raw cost of our workload depending on the workload size. Number of items / second in this (serial) benchmark can be compared to the numbers obtained in ThreadPoolSpawn.

git_commits:

1. **summary:** ARROW-5269: [C++][Archery] Mark relevant benchmarks as regression
   message: ARROW-5269: [C++][Archery] Mark relevant benchmarks as regression The goal of this change is to mark benchmarks candidate for automated regression checks. Some benchmarks were refactored for various reasons: - Code deduplication where needed - Uniform input size for benchmarks in the same suite, usually to minimize the effect of cache hierarchy. - Favor external repetitions over manual repetitions and mintime when possible. - Add cmake ARROW_BUILD_BENCHMARKS_REFERENCE to toggle reference benchmarks. - Remove default benchmark filter of `^Regression`. - Remove `BM_` prefix from benchmark names - Fixes and addons to `archery` to support the `--pdb` option for debugging and add support for benchmarks reported as `items_per_seconds`. - Adds `archery benchmark list` sub-command to list suites and benchmarks Author: François Saint-Jacques <fsaintjacques@gmail.com> Closes #4285 from fsaintjacques/ARROW-5269-regression-candidates and squashes the following commits: 83780304d <François Saint-Jacques> Address comments 4a570ab90 <François Saint-Jacques> Reformat a9ecffd98 <François Saint-Jacques> address comments d8e779aab <François Saint-Jacques> Reformat 8334c68f7 <François Saint-Jacques> Remove BM_ benchmark prefix 5cebe2c23 <François Saint-Jacques> Normalize benchmarks 908f05a5b <François Saint-Jacques> Change default repetitions to 10 instead of 20 ffe723f05 <François Saint-Jacques> Add `benchmark list` sub-command. e0876474b <François Saint-Jacques> Refactor JSON benchmarks a7b4f5f50 <François Saint-Jacques> Improve archery benchmark support bea76ed0e <François Saint-Jacques> Refactor csv benchmarks f207974c0 <François Saint-Jacques> Refactor CompareFilter benchmarks b7911df3d <François Saint-Jacques> Refactor Bitmap benchmarks 1b5135ebd <François Saint-Jacques> Refactor Builder benchmarks

github_issues:

github_issues_comments:

github_pulls:

1. **title:** ARROW-5269: [C++][Archery] Mark relevant benchmarks as regression
   body: The goal of this change is to mark benchmarks candidate for automated regression checks. Some benchmarks were refactored for various reasons: - Code deduplication where needed - Uniform input size

for benchmarks in the same suite, usually to minimize the effect of cache hierarchy. - Favor external repetitions over manual repetitions and mintime when possible. - Add cmake ARROW_BUILD_BENCHMARKS_REFERENCE to toggle reference benchmarks. - Remove default benchmark filter of `^Regression`. - Remove `BM_` prefix from benchmark names - Fixes and addons to `archery` to support the `--pdb` option for debugging and add support for benchmarks reported as `items_per_seconds`. - Adds `archery benchmark list` sub-command to list suites and benchmarks

github_pulls_comments:

1. **body:** I'm not sure why the "Regression" designation is necessary. It seems like we would want to monitor the performance of all of our benchmarks for significant changes.
   label: code-design
2. **body:** Multiple reasons: - some benchmarks are flaky. - some benchmarks exists only for point of reference (anything with Naive/Dummy in the name) for other benchmarks or hardware discover-ability, e.g. the memory latency and bandwidth benchmarks. - some benchmarks are heavyweight I think it's preferable to white list than include everything and possibly increase false positive regression alerts. It also lower the time to run benchmarks since archery is explicitly passing (by default) the `--benchmark_filter="^Regression"` options.
   label: test
3. By the way, I have no attachment to `Regression` it can be something else, but the function name (and hence the benchmark name) paired with a regex are the easiest way to label and filter.
4. **body:** If it's an issue with reporting on flaky benchmarks, we could create a "blacklist file" to instruct the reporter not to report regressions in known flakes.
   label: test
5. **body:** Some thoughts, for what they're worth: * A benefit of blacklisting vs. whitelisting (whatever the mechanism for indicating it) is that the default is to include benchmarks in regression testing, and I think that's the right default--if you're adding a benchmark, most likely the purpose is to make sure performance doesn't go down in the future. The bias should be in favor of this, and make developers have to opt out by blacklisting. * I'm generally of the opinion that flaky tests have negative value--you don't notice them when they pass, and when they fail, you don't believe them--and should either be fixed immediately or deleted (or skipped, if you're into that). My instinct is that the same is true for flaky benchmarks. * Maybe heavy benchmarks belong in a different place altogether since we aren't going to run them routinely.
   label: test
6. Ok, I'll just remove the filtering by default. Should we strip the `BM_` out of every benchmark names? I don't see the value in keeping this.
7. **body:** It's been several days and I'm still having a hard time swallowing the "Regression*" naming convention
   label: code-design
8. @ursabot benchmark
9. I've successfully started builds for this PR
10. [AMD64 Ubuntu 18.04 C++ Benchmark](https://ci.ursalabs.org/#builders/73/builds/19) Build failed.
11. # [Codecov](https://codecov.io/gh/apache/arrow/pull/4285?src=pr&el=h1) Report > Merging [#4285](https://codecov.io/gh/apache/arrow/pull/4285?src=pr&el=desc) into [master](https://codecov.io/gh/apache/arrow/commit/68329e728c5356f85c4a492c9749ea9125b06993?src=pr&el=desc) will **increase** coverage by `0.98%`. > The diff coverage is `n/a`. [![Impacted file tree graph](https://codecov.io/gh/apache/arrow/pull/4285/graphs/tree.svg?width=650&token=LpTCFbqVT1&height=150&src=pr)](https://codecov.io/gh/apache/arrow/pull/4285?src=pr&el=tree) ```diff @@ Coverage Diff @@ ## master #4285 +/- ## == + Coverage 88.3% 89.28% +0.98% == Files 780 636 -144 Lines 98400 87138 -11262 Branches 1251 0 -1251 == - Hits 86891 77801 -9090 + Misses 11273 9337 -1936 + Partials 236 0 -236 ``` | [Impacted Files](https://codecov.io/gh/apache/arrow/pull/4285?src=pr&el=tree) | Coverage Δ | | |---|---|---| | [cpp/src/plasma/thirdparty/ae/ae.c](https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Y3BwL3NyYy9wbGFzbWEvdGhpcmRwYXJ0eS9hZS9hZS5j) | `70.75% <0%> (-0.95%)` | :arrow_down: | | [cpp/src/plasma/test/external_store_tests.cc](https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Y3BwL3NyYy9wbGFzbWEvdGVzdC9leHRlcm5hbF9zdG9yZV90ZXN0cy5jYw==) | `100% <0%> (ø)` | :arrow_up: | | [cpp/src/plasma/test/client_tests.cc](https://codecov.io/gh/apache/arrow/pull/4285/diff?

src=pr&el=tree#diff-Y3BwL3NyYy9wbGFzbWEvdGVzdC9jbGllbnRfdGVzdHMuY2M=) | `100% <0%> (ø)` | :arrow_up: | | [cpp/src/plasma/test/serialization_tests.cc] (https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Y3BwL3NyYy9wbGFzbWEvdGVzdC9zZXJpYWxpemF0aW9uX3Rlc3RzLmNj) | `100% <0%> (ø)` | :arrow_up: | | [go/arrow/ipc/writer.go](https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Z28vYXJyb3cvaXBjL3dyaXRlci5nbw==) | | | | [go/arrow/math/uint64_amd64.go] (https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Z28vYXJyb3cvbWF0aC91aW50NjRfYW1kNjQuZ28=) | | | | [go/arrow/memory/memory_avx2_amd64.go](https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Z28vYXJyb3cvbWVtb3J5L21lbW9yeV9hdngyX2FtZDY0Lmdv) | | | | [go/arrow/ipc/file_reader.go](https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Z28vYXJyb3cvaXBjL2ZpbGVfcmVhZGVyLmdv) | | | | [js/src/enum.ts] (https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-anMvc3JjL2VudW0udHM=) | | | | [go/arrow/array/builder.go](https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree#diff-Z28vYXJyb3cvYXJyYXkvYnVpbGRlci5nbw==) | | | | ... and [144 more] (https://codecov.io/gh/apache/arrow/pull/4285/diff?src=pr&el=tree-more) | | ------ [Continue to review full report at Codecov](https://codecov.io/gh/apache/arrow/pull/4285?src=pr&el=continue). > **Legend** - [Click here to learn more](https://docs.codecov.io/docs/codecov-delta) > `Δ = absolute <relative> (impact)`, `ø = not affected`, `? = missing data` > Powered by [Codecov] (https://codecov.io/gh/apache/arrow/pull/4285?src=pr&el=footer). Last update [68329e7...27633c9] (https://codecov.io/gh/apache/arrow/pull/4285?src=pr&el=lastupdated). Read the [comment docs] (https://docs.codecov.io/docs/pull-request-comments).
12. @ursabot benchmark
13. @pitrou note that I added the cmake configuration flag `ARROW_BUILD_BENCHMARKS_REFERENCE` which toggles a C/C++ define `ARROW_WITH_BENCHMARKS_REFERENCE`. The goal is to avoid compiling (and running) benchmarks which are not used for monitoring purposes but only point of reference, e.g. memory latency/bandwidth, naive implementations of algorithms... Thus if you add a new benchmark and you know that it's only used for a point of reference of the true benchmark, you should probably wrap it with said `#ifdef`.
14. I've successfully started builds for this PR
15. [AMD64 Ubuntu 18.04 C++ Benchmark](https://ci.ursalabs.org/#builders/73/builds/20) Build failed with an exception.
16. @ursabot benchmark
17. I've successfully started builds for this PR
18. Why did We get empty output in https://ci.ursalabs.org/#builders/73/builds/20 ?
19. @kszucs because benchmarks were renamed and thus the intersection between 2 run is empty. See https://github.com/apache/arrow/blob/master/dev/archery/archery/benchmark/compare.py#L115-L116
20. [AMD64 Ubuntu 18.04 C++ Benchmark](https://ci.ursalabs.org/#builders/73/builds/21) Build failed with an exception.
21. @pitrou @kszucs I addressed the comments.

github_pulls_reviews:

1. `BuildIntArrayNoNulls`?
2. Is this equal to `kBytesProcessPerRound`?
3. **body:** I find `kBytesProcessPerRound` and `kBytesProcessed` a bit confusing, at first `round` and `iteration` seem synonyms, but they mean different loops.
   label: code-design
4. **body:** A bit more descriptive name?
   label: code-design
5. `click.echo`?
6. **body:** Really just a nit, but `ARROW_WITH_REFERENCE_BENCHMARKS` might be a better choice.
   label: code-design
7. Well, there's `ARROW_BUILD_BENCHMARKS` already. The convention seems to be `ARROW_BUILD_xxx` to enable the `xxx` target, and `ARROW_WITH_yyy` to enable support for the `yyy` external library.
8. What is this already? I can't find a reference for the git pseudo-revisions.
9. Or simply: ```python return self.bytes_per_second or self.items_per_second or self.time ``` ;-)

10. I'm curious, why is this a reference benchmark?
11. Shouldn't the reference benchmarks here be built conditionally?
12. **body:** I think these are pretty much reference benchmarks, unless we find out that compression performance is affected by our abstraction layer...
    label: code-design
13. Is it intended to force the number of repetitions?
14. Ditto here.
15. The "inlined trie" benchmarks are reference benchmarks (to compare against a hand-written specialized version of trie lookup).
16. I think this is a reference benchmark as well.
17. Are you suggesting changing the s/BUILD/WITH/ or the swap of BENCHMARKS/REFERENCE?
18. Correct, `iterations` is a google benchmark concept. Round is used to make a distinction between batch APIs versus point APIs.
19. I'll rename `kFinalSize` to `kRounds` for this.
20. it's not in git pseudo revisions, it's a magic keyword that only `archery` knows, it basically says, don't touch the git repository, use the workspace as-is (with dirty worktree). In `archery benchmark diff --help`: > The special token "WORKSPACE" is reserved to specify the current git workspace. This imply that no clone will be performed.
21. I decided not to include it yet because the n_proc inference isn't accounting for cpuset (cgroup). Thus I fear this benchmark is highly variable if it runs in cgroups, which I've prepared on dgx2 for benchmarks. Since we don't have a database, I'll revert this and leave it as a normal benchmark.
22. Hmm... if the baseline is the workspace, what is it compared against?
23. That's the run sub-command, so it's not comparing against anything. Baseline might be mis-leading here, it probably should be revision?
24. Yes, that would be better :)
25. Only this one, or all of them?
26. This one only.
27. **body:** Yes, wanted to minimize the time it takes to run and this one was in the heavy. I'll remove it since it doesn't change by that much.
    label: code-design
28. **body:** Hmm... why? You could use `const std::string`.
    label: code-design
29. **body:** Nit: "BuildCSVData".
    label: code-design
30. You're not enclosing this is in an `#ifdef ARROW_WITH_BENCHMARKS_REFERENCE`. Is it an oversight?
31. That's extremely small.
32. Due to linter, I'll add a comment.
33. **body:** Wanted to fit in L1, do you suggest a better size? Local tests shows not much difference by increasing this.
    label: code-design
34. At least 64 kiB. Fitting in L1 is not really a goal here.
35. yes.
36. I'm not saying that fitting in L1 is the goal, I meant that some benchmarks (e.g. the kernels), are more limited by the memory bandwidth so lowering the input does show the "maximal" rate. In this case, it doesn't change anything so I'll leave it to 64kb. Note that my process was to go through all benchmarks and lower the inputs to get a faster running suite. This change is a collateral damage.

jira_issues:

1. **summary:** [C++] Whitelist benchmarks candidates for regression checks
   description: Rename all benchmarks candidate for regression with the `Regression` prefix.

jira_issues_comments:

1. Issue resolved by pull request 4285 [https://github.com/apache/arrow/pull/4285]