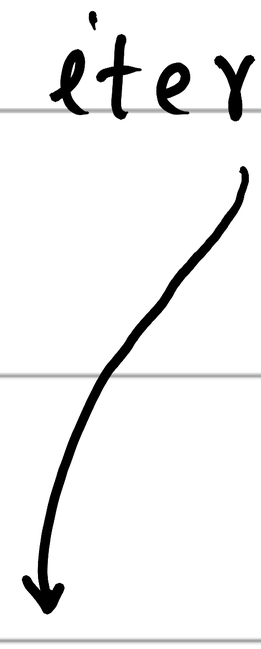
BenchmarkRunner needs to predict iterations for a BenchmarkInstance.

Prediction mainly involves identifying number of iterations to be done after current I iterations.

Predicting Iterations involves

① Picking multiplier for multiplying current iterations with.

② Deciding next iterations



maxOf (
currentIterations
* multiplier,
currentIterations
+ 1);

Steps involved in predicting iterations :

1. Picking multiplier
2. Refining multiplier
3. Ensuring new iterations increase atleast by 1.
4. Capping new iterations.

```cpp
struct IterationResults {
internal::ThreadManager::Result results;
 IterationCount iters;
  double seconds;
};
IterationCount BenchmarkRunner::PredictNumItersNeeded(
    const IterationResults& i) const {
  // See how much iterations should be increased by.
  // Note: Avoid division by zero with max(seconds, 1ns).
  double multiplier = GetMinTimeToApply() * 1.4 / std::max( a: i.seconds,  b: 1e-9);
  // If our last run was at least 10% of FLAGS_benchmark_min_time then we
  // use the multiplier directly.
  // Otherwise we use at most 10 times expansion.
  // NOTE: When the last run was at least 10% of the min time the max
  // expansion should be 14x.
  const bool is_significant = (i.seconds / GetMinTimeToApply()) > 0.1;
  multiplier = is_significant ? multiplier : 10.0;

  // So what seems to be the sufficiently-large iteration count? Round up.
  const IterationCount max_next_iters = static_cast<IterationCount>(
      std::llround( x: std::max( a: multiplier * static_cast<double>(i.iters),
                                 b: static_cast<double>(i.iters) + 1.0)));
  // But we do have *some* limits though..
  const IterationCount next_iters = std::min( a: max_next_iters,  b: kMaxIterations);

  BM_VLOG(3) << "Next iters: " << next_iters << ", " << multiplier << "\n";
  return next_iters;  // round up before conversion to integer.
}
```

picking multiplier

refining multiplier

new iterations & its cap.

① Picking multiplier: the idea is to multiply current iterations, such that the amount of time to execute those iterations is 1.4 times the minimum time specified by the user to run benchmark.

↳ new iterations

double multiplier =

(1.4 x bench_time) /

max(i.seconds / time for , $1e^{-9}$);

handle

division previous

iteration

by zero,

if i.seconds = 0

GetMinTimeToApply() Returns

bench_time.

(2)

Multiplier is further refined.

```
is_significant = (i.seconds /
                  bench_time)
                 > 0.1
```

If the last run already took 10% of the bench_time, we should pick the multiplier, else choose a fixed multiplier.

```
multiplier = is_significant ?
             multiplier : 10.0;
```

③

Next is ensuring that new iterations = atleast (current iterations + 1);

new_iterations = max (multiplier * i_iters, i_iters + 1);

if multiplier is too small, say 0.11, then multiplier * i_iters will be equal to

iters. We must ensure

that iterations increase by

1.

Last step is capping iterations.

```
new_iterations = min(
    new_iterations,
    kMaxIterations);
```