

Before a benchmark (or benchmark instance) can be run, it undergoes a warmup.

Warmup is all about doing N iterations of a benchmark instance. Warmup is done only ONCE per runner.

In this note

it was discussed that a single instance of benchmark runner

may repeat R times. But,
a it is important to note
that a single BenchmarkRunner
will do warm once only.

Up
warm also involves predicting
iterations, which is explained
here :

```
void BenchmarkRunner::DoOneRepetition() {
    assert(HasRepeatsRemaining() && "Already done all repetitions?");

    const bool is_the_first_repetition = num_repetitions_done == 0;

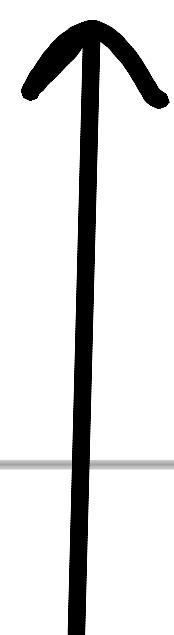
    // In case a warmup phase is requested by the benchmark, run it now.
    // After running the warmup phase the BenchmarkRunner should be in a state as
    // this warmup never happened except the fact that warmup_done is set. Every
    // other manipulation of the BenchmarkRunner instance would be a bug! Please
    // fix it.
    if (!warmup_done) {
        RunWarmUp();
    }
}
```



Warmup, if not done.

This code is a part of

BenchmarkRunner.



It is also important

to note that the same

instance of Runner will not

run in multiple threads.

```

void BenchmarkRunner::RunWarmUp() {
    // Use the same mechanisms for warming up the benchmark as used for actually
    // running and measuring the benchmark.
    IterationResults i_warmup;
    // Dont use the iterations determined in the warmup phase for the actual
    // measured benchmark phase. While this may be a good starting point for the
    // benchmark and it would therefore get rid of the need to figure out how many
    // iterations are needed if min_time is set again, this may also be a complete
    // wrong guess since the warmup loops might be considerably slower (e.g.
    // because of caching effects).
    const IterationCount i_backup = iters;
    for (;;) {
        b.Setup();
        i_warmup = DoNIterations();
        b.TearDown();

        const bool finish = ShouldReportIterationResults(i_warmup);

        if (finish) {
            FinishWarmUp(i_backup);
            break;
        }
    }
    // Although we are running "only" a warmup phase where running enough
    // iterations at once without measuring time isn't as important as it is for
    // the benchmarking phase, we still do it the same way as otherwise it is
    // very confusing for the user to know how to choose a proper value for
    // min_warmup_time if a different approach on running it is used.
    iters = PredictNumItersNeeded(i_warmup);
    assert(iters > i_warmup.iters &&
           "if we did more iterations than we want to do the next time, "
           "then we should have accepted the current iteration run.");
}

```

infinite loop :)

WarmUP involves:

1. Setup

2. N iterations

3. Teardown

4. Identifying if sufficient

iterations are done.

5. Else, predicting next iterations

Setup & Teardown involve creating

a state instance

& invoking Setup, tearDown on

BenchmarkInstance.

Quick note on : ShouldReportIterationResults.

Results .

```
bool BenchmarkRunner::ShouldReportIterationResults(  
    const IterationResults& i) const {  
    // Determine if this run should be reported;  
    // Either it has run for a sufficient amount of time  
    // or because an error was reported.  
    return (i.results.skipped_ != 0u) || FLAGS_benchmark_dry_run ||  
        i.iters >= kMaxIterations || // Too many iterations already.  
        i.seconds >=  
            GetMinTimeToApply() || // The elapsed time is large enough.  
            // CPU time is specified but the elapsed real time greatly exceeds  
            // the minimum time.  
            // Note that user provided timers are except from this test.  
            ((i.results.real_time_used >= 5 * GetMinTimeToApply()) &&  
             !b.use_manual_time());  
}
```

If the warmup
has crossed max ·
supported iterations

or warmup

For now, just consider time

① benchtime .

;(i.seconds)

② MaxIterations ,

; ≥ bench-

time .

DoNIterations

```
BenchmarkRunner::IterationResults BenchmarkRunner::DoNIterations() {
    BM_VLOG(2) << "Running " << b.name().str() << " for " << iters << "\n";

    std::unique_ptr<internal::ThreadManager> manager;
    manager.reset(new internal::ThreadManager(b.threads()));

    thread_runner->RunThreads( fn: [&](int thread_idx) -> void {
        RunInThread(&b, iters, thread_idx, manager.get(),
                    perf_counters_measurement_ptr, /*profiler_manager=*/nullptr);
    });
}

IterationResults i;
// Acquire the measurements/counters from the manager, UNDER THE LOCK!
{
    MutexLock l(m: [&] manager->GetBenchmarkMutex());
    i.results = manager->results;
}
```

→ Run N threads

```
void RunInThread(const BenchmarkInstance* b, IterationCount iters,
                  int thread_id, ThreadManager* manager,
                  PerfCountersMeasurement* perf_counters_measurement,
                  ProfilerManager* profiler_manager_) {
    internal::ThreadTimer timer(
        b->measure_process_cpu_time()
        ? internal::ThreadTimer::CreateProcessCpuTime()
        : internal::ThreadTimer::Create());
    State st = b->Run(iters, thread_id, &timer, manager,
                      perf_counters_measurement, profiler_manager_);
    if (!(st.skipped() || st.iterations() >= st.max_iterations)) {
        st.SkipWithError(
            msg: [&] "The benchmark didn't run, nor was it explicitly skipped. Please call "
            "'SkipWithXXX' in your benchmark as appropriate.");
    }
    {
        MutexLock l(m: [&] manager->GetBenchmarkMutex());
        internal::ThreadManager::Result& results = manager->results;
        results.iterations += st.iterations();
        results.cpu_time_used += timer.cpu_time_used();
        results.real_time_used += timer.real_time_used();
        results.manual_time_used += timer.manual_time_used();
        results.complexity_n += st.complexity_length_n();
        internal::Increment(&results.counters, st.counters);
    }
}
```

→ Run a benchmark instance.

DoNIterations uses a variable iters

which is at the level of

BenchmarkRunner. It comes from

BenchmarkInstance, or starts from 1.

Ignore
collecting
metrics for
now.

b. threads() = no. of threads configured

at Benchmark instance, default is

1.

Doubt : Assume, $\text{iters} = 100$, threads

= 5. It seems each thread is

doing 100 iterations. Is that right?