Microbenchmarking Using Google Benchmark

CS 165, Project in Algorithms and Data Structures UC Irvine
Spring 2020

Presented by Haleh Havvaei
Microbenchmarking is about measuring the time or performance of small to very small building blocks of real programs. This can be a common data access pattern, a sequence of operations or even a single instruction.

We can use Google Benchmark which is a library to support the benchmarking of functions.

Installation Guide
Installing Google Benchmark

• The library can be used with C++03. However, it requires C++11 to build, including compiler and standard library support.

• Installation Guid:
  • https://github.com/google/benchmark#installation
Sample Benchmark Function

```cpp
#include <benchmark/benchmark.h>

static void BM_StringCreation(benchmark::State& state) {
    for (auto _ : state)
        std::string empty_string;
}

// Register the function as a benchmark
BENCHMARK(BM_StringCreation);
BENCHMARK_MAIN();
```

• Build:
  
g++ test.cpp -std=c++11 -isystem benchmark/include \-Lbenchmark/build/src -lbenchmark -lpthread -o test
  
• Run:
  
  ./test
(base) Halehs-MacBook-Pro:test haleh$ ./test
2021-03-31T19:56:56-07:00
Running ./test
Run on (4 X 2300 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x2)
  L1 Instruction 32 KiB (x2)
  L2 Unified 256 KiB (x2)
  L3 Unified 4096 KiB (x1)
Load Average: 1.51, 1.66, 2.00

----------------------------------------------------------
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time</th>
<th>CPU</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM_StringCreation</td>
<td>31.7 ns</td>
<td>31.6 ns</td>
<td>22137257</td>
</tr>
</tbody>
</table>
----------------------------------------------------------

- CPU: It is the quantity of processor time taken by the process. This does not indicate duration.
- Time: elapsed time
- For more info check here
Sample Benchmark with Randomized Input

```cpp
std::vector<int> populateRandom(int n){ //Notice though that this function
  // does not generate uniformly distributed random numbers for the vector
  std::vector<int> v;
  for(int i = 0; i < n; i++){
    int a = rand() % (1 << 31);
    v.push_back(a);
  }
  return v;
}

static void Merge_sort_BM(benchmark::State& state) {
  while (state.KeepRunning())
  {
    state.PauseTiming();
    std::vector<int> v;
    v = populateRandom(state.range(0));
    state.ResumeTiming();
    mergeSort(v, 0, v.size() - 1);
  }
}

BENCHMARK(Merge_sort_BM)->Args({2000})->Complexity();
BENCHMARK(Merge_sort_BM)->Args({2000})->Unit(benchmark::kMillisecond);
BENCHMARK_MAIN();
```
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time</th>
<th>CPU</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge_sort_BM/2000</td>
<td>130535067 ns</td>
<td>129922600 ns</td>
<td>5</td>
</tr>
<tr>
<td>Merge_sort_BM/2000</td>
<td>131 ms</td>
<td>130 ms</td>
<td>5</td>
</tr>
</tbody>
</table>
References and Additional Info

- https://github.com/google/benchmark