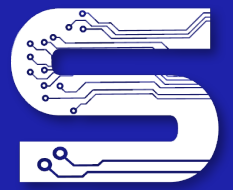


Syntacore™
Custom cores and tools

Разработка инструментария для исследования сравнительной производительности параллельных алгоритмов стандартной библиотеки C++

Студент:
Сидельников Станислав Игоревич

Научный руководитель:
Владимиров Константин Игоревич

Цель работы

- Исследование производительности параллельных алгоритмов стандартной библиотеки C++
- Задачи:
  - Исследование структуры кода для измерения производительности алгоритмов стандартной библиотеки
  - Разработка фреймворка для генерации бенчмарков и анализа результатов их выполнения

Актуальность

- На исследовании производительности параллельных алгоритмов влияет множество факторов
  - Библиотека анализа производительности
  - Входные данные бенчмарка
  - Алгоритм
  - Policy
  - Компилятор
- Для эффективного анализа производительности и поиска регрессий нужна некоторая инфраструктура для автоматической генерации и анализа результатов исполнения бенчмарков

Исследование структуры кода

- Типичный пример измеряемого кода:

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

#include <algorithm>
#include <execution>
#include <numeric>

static void any_of(benchmark::State& state) {
  std::vector<int> src_container(1000000);
  std::iota(src_container.begin(), src_container.end(), 0);

  for (auto _: state) {
    auto result = std::all_of(std::execution::par, src_container.begin(), src_container.end(), [](int i) { return i
== 0; });
    benchmark::DoNotOptimize(result);
  }

}


BENCHMARK(any_of);
BENCHMARK_MAIN()
```

Заголовочные файлы

Исследование структуры кода

- Типичный пример измеряемого кода:

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

#include <algorithm>
#include <execution>
#include <numeric>

static void any_of(benchmark::State& state) {
  std::vector<int> src_container(1000000);
  std::iota(src_container.begin(), src_container.end(), 0);

  for (auto _: state) {
    auto result = std::all_of(std::execution::par, src_container.begin(), src_container.end(), [](int i) { return i
== 0; });
    benchmark::DoNotOptimize(result);
  }

}


BENCHMARK(any_of);
BENCHMARK_MAIN()
```

Некоторый шаблон библиотеки бенчмаркинга

Исследование структуры кода

- Типичный пример измеряемого кода:

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

#include <algorithm>
#include <execution>
#include <numeric>

static void any_of(benchmark::State& state) {
  std::vector<int> src_container(1000000);
  std::iota(src_container.begin(), src_container.end(), 0);

  for (auto _: state) {
    auto result = std::all_of(std::execution::par, src_container.begin(), src_container.end(), [](int i) { return i
== 0; });
    benchmark::DoNotOptimize(result);
  }

}


BENCHMARK(any_of);
BENCHMARK_MAIN()
```

Инициализация контейнеров

Исследование структуры кода

- Типичный пример измеряемого кода:

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

#include <algorithm>
#include <execution>
#include <numeric>

static void any_of(benchmark::State& state) {
  std::vector<int> src_container(1000000);
  std::iota(src_container.begin(), src_container.end(), 0);

  for (auto _: state) {
    auto result = std::all_of(std::execution::par, src_container.begin(), src_container.end(), [](int i) { return i
== 0; });
    benchmark::DoNotOptimize(result);
  }

}


BENCHMARK(any_of);
BENCHMARK_MAIN()
```

Вызов бенчмарка

Наследование шаблонов

Базовое описание структуры бенчмарка

```
{% block includes %}
{% endblock %}

{% block benchmark_function %}

  {% block setup %}
    {% block init_container %}
    {% endblock %}
  {% endblock %}

  {% block loop %}
    {% block benchmark %}
    {% endblock %}
  {% endblock %}
{% endblock %}

{% block bench_register %}
{% endblock %}
```

Шаблон библиотеки бенчмаркинга

```
{% extends "base.jinja" %}
{% block includes %}
#include
<benchmark/benchmark.h>
{% endblock %}

{% block benchmark_function
%}
  static void {{name}}
(benchmark::State& state) {
  {% block setup %}
    {% block init_container %}
    {% endblock %}
  {% endblock %}
  {% block loop %}
  for (auto _: state) {
    {% block benchmark %}
    {% endblock %}
  }
  {% endblock %}
}
{% endblock %}

{% block bench_register %}
BENCHMARK({{name}});
BENCHMARK_MAIN();
{% endblock %}
```

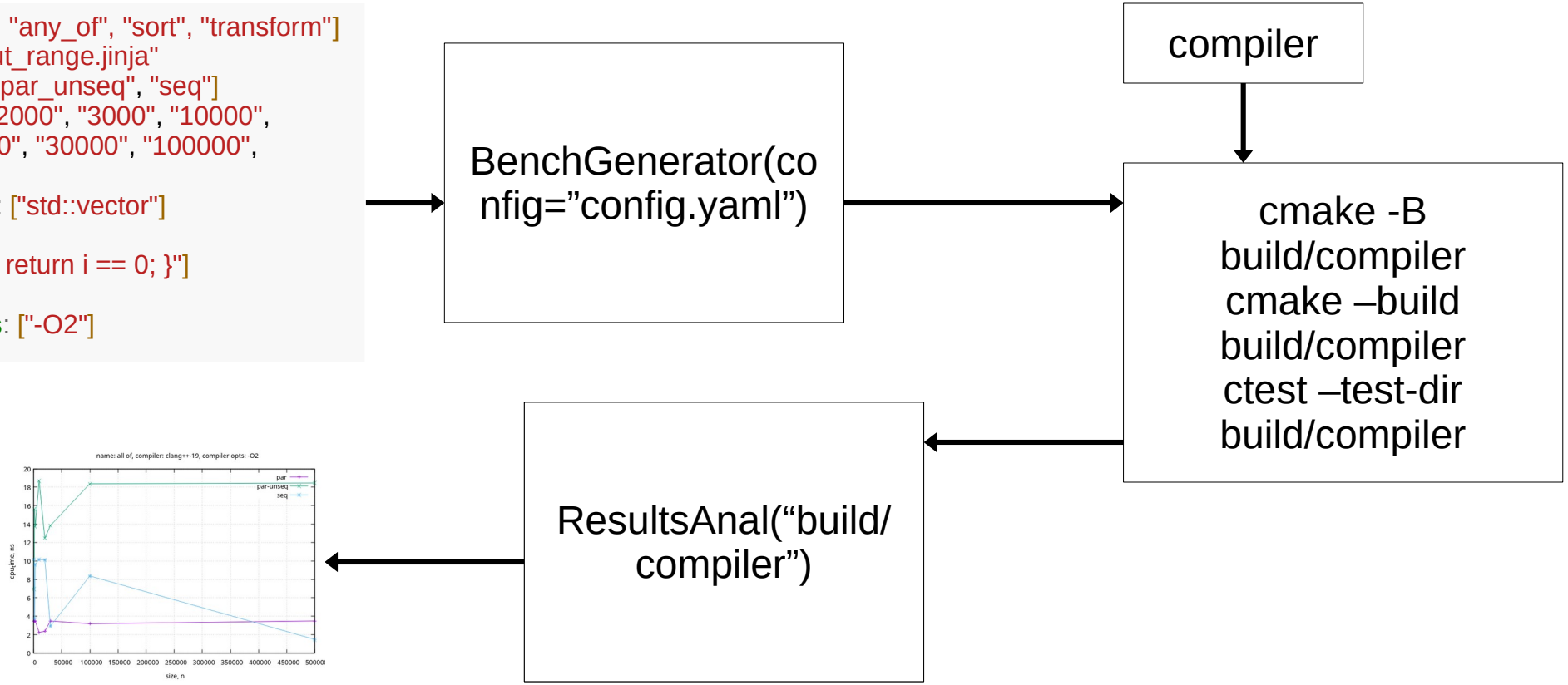
Шаблон некоторого подмножества алгоритмов

```
{% extends "googlebenchmark.jinja" %}
{% block includes %}
{{ super() }}

#include <algorithm>
#include <execution>
#include <numeric>
{% endblock %}

{% block benhcmark_function %}
{{ super () }}
  {% block setup %}
  {{ src_container }}<{{T}}> src_container({{ size }});
  {% block init_container %}
    {{ super () }}
    std::iota(src_container.begin(), src_container.end(), 0);
    {% endblock %}
  {% endblock %}

  {% block benchmark %}
  auto result = std::{{ name }}(std::execution::{{ policy }},
src_container.begin(), src_container.end(), {{ func }});
  benchmark::DoNotOptimize(result);
  {% endblock %}

{% endblock %}
```

Создание фреймворка для исследования производительнос

- Для того, чтобы работать с большим количеством бенчмарков, был разработана инфраструктура

```
name: ["all_of", "any_of", "sort", "transform"]
template: "input_range.jinja"
policy: ["par", "par_unseq", "seq"]
size: ["1000", "2000", "3000", "10000",
"10000", "20000", "30000", "100000",
"500000"]
src_container: ["std::vector"]
T: ["int"]
func: ["[](int i) { return i == 0; }"]

compiler_opts: ["-O2"]
```



Результаты исследования производительности

- Пример исследования:
  - компилятор: clang++-19
  - опции компиляции: -O2
  - policy: par, par_unseq, seq
  - алгоритм: all_of