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

Студент: Сидельников Станислав Игоревич
Научный консультант: Владимиров Константин Игоревич

Цель работы и задачи

Цель работы:

- Исследование производительности параллельных алгоритмов стандартной библиотеки C++

Задачи:

- Исследование типовой структуры бенчмарков параллельных алгоритмов
- Обобщение описания бенчмарков с использованием механизмов шаблонизации
- Разработка фреймворка для генерации бенчмарков и анализа результатов их выполнения
- Тестирование сгенерированных бенчмарков на архитектуре RISC-V

Архитектура RISC-V

- Архитектура RISC-V – популярная молодая открытая архитектура
  - **Открытость:** спецификация полностью открыта, позволяет коммерческое и некоммерческое использование
  - **Расширяемость:** архитектура позволяет комбинировать различные расширения (стандартные и нестандартные)

- Возникают задачи поддержки компилятора под конкретную архитектуру
- При разработке компилятора необходимо решать вопрос исследования производительности

Алгоритмы стандартной библиотеки

- Одной из областей исследования производительности компилятора – алгоритмы стандартной библиотеки (**#include <algorithm>**)
- В данной работе производительность данных алгоритмов рассматривается в контексте их распараллеливания с помощью **policy**
- В стандартной библиотеке порядка 105 алгоритмов, которые используются похожим образом

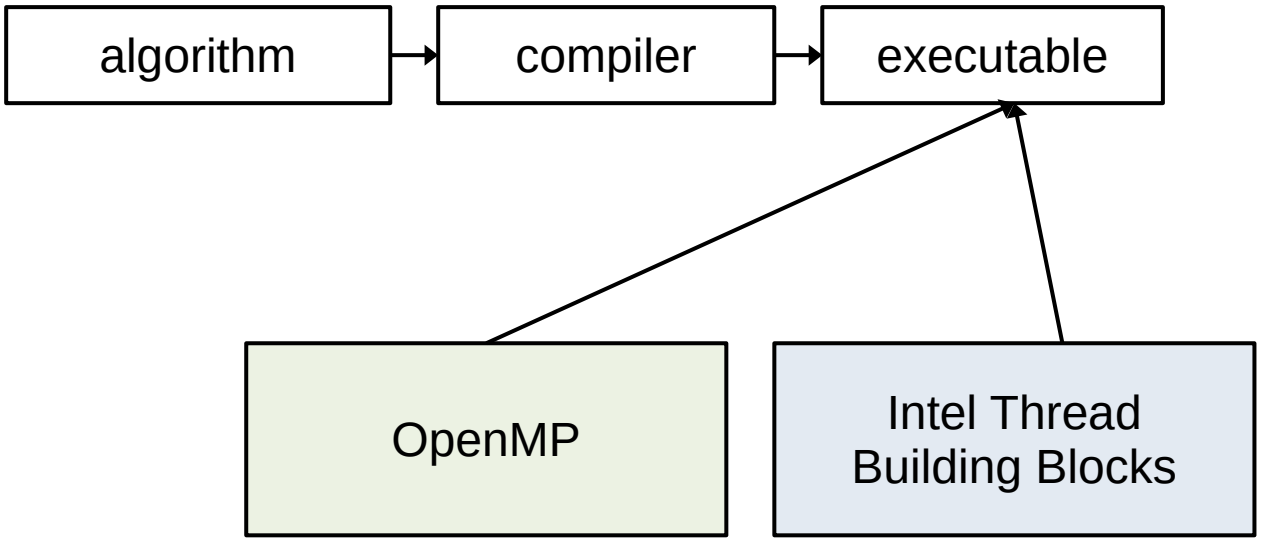
```
std::for_each(std::execution::par,
v.begin(), v.end(), [](int &n) { n++; })


std::all_of(std::execution::par_unseq,
v.cbegin(), v.cend(), [](int i)
{ return i % 2 == 0; }

std::find_end(std::execution::seq,
v.begin(), v.end(), x.begin(), x.end())
```

Параллельные алгоритмы

- С C++17 все алгоритмы получили возможность распараллеливания, тип распараллеливания регулируется **policy**

- **seq –** последовательное исполнение

- **par –** распараллеливание с использованием библиотек (**openmp, tbb)**
- **par_unseq –** распараллеливание с векторизацией (самая агрессивная политика распараллеливания)



Задача измерения производительности

- При решении задачи измерения приводительности надо иметь в виду несколько вещей:
  - Выбор библиотека бенчмаркинга: - например, **googlebenchmark** (использовалась как основная в работе), **celero, cppbenchmark, cpp, quickbench**
  - Написание самих бенчмарков
  - Сборка бенчмарков с различными компиляторами и опциями
  - Запуск бенчмарков, анализ результатов, графическая интерпретация

- Как правило, для эффективного измерения производительности, в закрытых решениях пишется некоторый слой автоматизации
- Данный слой автоматизации позволяет решить в частном виде задачу измерения производительгости для каких-то конкретных примеров (например, для параллельных алгоритмов)

Задача измерения производительности

- При решении задачи измерения приводительности надо иметь в виду несколько вещей:
  - Выбор библиотека бенчмаркинга: - например, **googlebenchmark** (использовалась как основная в работе), **celero, cppbenchmark, cpp, quickbench**
  - Написание самих бенчмарков
  - Сборка бенчмарков с различными компиляторами и опциями
  - Запуск бенчмарков, анализ результатов, графическая интерпретация

- Как правило, для эффективного измерения производительности, в закрытых решениях пишется некоторый слой автоматизации
- Данный слой автоматизации позволяет решить в частном виде задачу измерения производительгости для каких-то конкретных примеров (например, для параллельных алгоритмов)

Фреймворк измерения производительности stdbench

- Для решения задачи измерения производительности параллельных алгоритмов стандартной библиотеки был написан фреймворк **stdbench**

Создание различных конфигураций бенчмарков

```
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"]
compier: [“clang++-19”, “g++-14”]
```

Генерация бенчмарков

BenchGenerator(config=”config.yaml”)

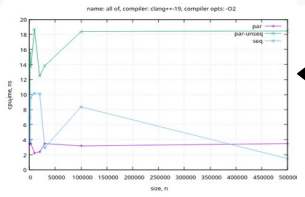
compiler

Сборка бенчмарков предоставленными компиляторами

```
cmake -B build/compiler
cmake –build build/compiler
ctest –test-dir build/compiler
```

Запуск бенчмарков и анализ результатов

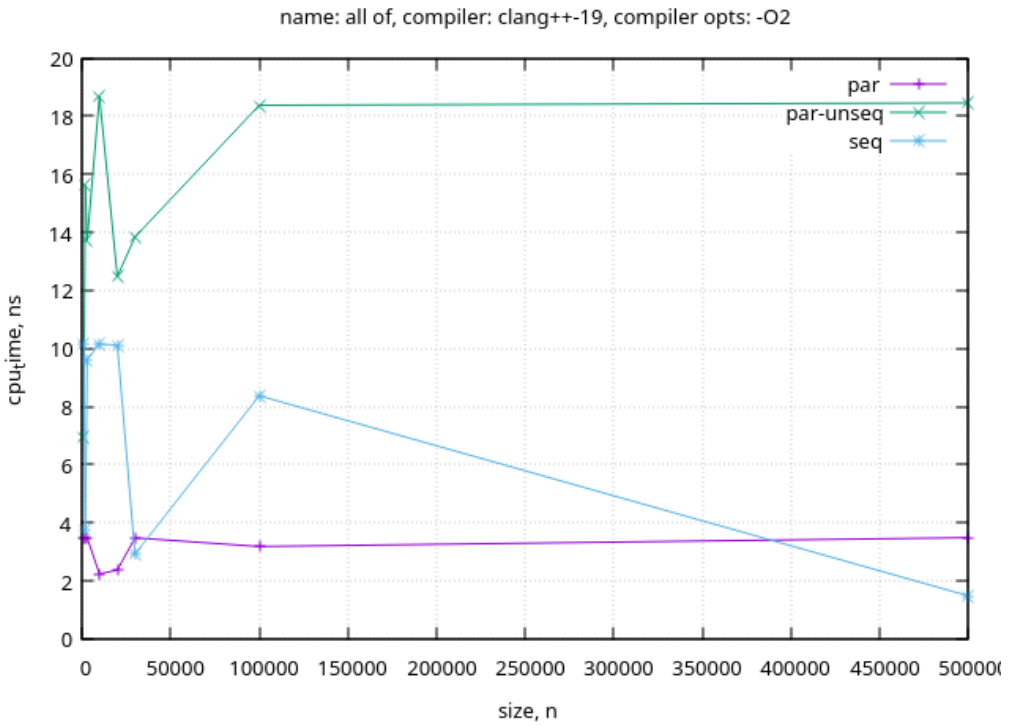
ResultsAnal(“build/compiler”)



Графическая интерпретация

Результаты исследования производительности

- Пример исследования
  - Компилятор: clang++-19
  - Опции компиляции: -O2
  - Policy: par, par_unseq, seq
  - Алгоритм: all_of



Backlog

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 %}
```

Исследование структуры 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

```
#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

```
#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

```
#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);
  }

}
```

Вызов бенчмарка

```
BENCHMARK(any_of);
BENCHMARK_MAIN()
```