УДК 004.432

ИНСТРУМЕНТЫ ПАРАЛЛЕЛИЗАЦИИ АЛГОРИТМОВ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++

Огородник И.В.
Научный руководитель – Прихожий А.А., д.т.н., профессор

С окончанием разработки и утверждением стандарта ISO/IEC 14882:2011, более известного как C++11, в язык программирования C++ была введена поддержка многопоточности на уровне ядра языка. Данная поддержка реализована двумя стандартными библиотеками: Thread support library и Atomic support library [1].

Вышеуказанные библиотеки предоставляют поддержку примитивов синхронизации и атомарных операций. Данные операции и примитивы ограничены в работе одним процессом, но при этом с ними может взаимодействовать неограниченное число потоков.

Таким образом, в C++ была добавлена возможность вручную разрабатывать многопоточные алгоритмы. В последующих редакциях стандарта, библиотеки дополнились новыми видами примитивов (семафор, барьер, общий мьютекс и т.п.).

Иное направление развития многопоточных алгоритмов появилось с выходом стандарта C++17. В данной версии языка была расширена библиотека стандартных алгоритмов Algorithm support library путем введения политик выполнения.

Политика выполнения – вид объекта в языке C++, описывающий то, каким образом может и должен выполняться некоторый код. В текущей версии языка (C++20) имеется четыре стандартных политики выполнения: sequenced policy, parallel policy, parallel unsequenced policy, unsequenced policy. Список политик может быть дополнен в различных компиляторах.

Политика последовательного выполнения (sequenced policy) указывает на то, что параллельную версию алгоритма запрещено использовать и операции над элементами массива должны выполняться последовательно.

Политика неупорядоченного выполнения (unsequenced policy) указывает, что выполнение функции должно происходить в вызывающем потоке, но при этом порядок обработки элементов не имеет значения и могут быть использованы инструкции векторизации (SIMD и т.п.).

Политика параллельного выполнения (parallel policy) означает, что алгоритм может выполняться на нескольких потоках, но в одном потоке элементы обрабатываются последовательно.

Политика параллельного неупорядоченного выполнения (parallel unsequenced policy) означает, что алгоритм может выполняться на нескольких

потоках. При этом в одном потоке элементы могут обрабатываться в произвольном порядке.

Стоит обратить внимание, что при использовании любой стандартной политики возникновение необработанной исключительной ситуации приведет к завершению работы приложения через вызов std::terminate. Кроме того, в случае невозможности параллелизовать или векторизовать алгоритм (например, при нехватке ресурсов), он будет выполнен последовательно.

Для сравнения времени выполнения стандартных алгоритмов при разных политиках выполнения были смоделированы задача трансформации (изменения) элементов массива и задача сортировки элементов в массиве.

Задача по изменению элементов массива (сложение с самим собой) была реализована через функцию std::transform, имеющую линейную сложность. Для каждого размера массива замеры повторялись 100 раз. Тестовая программа была скомпилирована при помощи компилятора MSVC v143 при уровне оптимизации O2. Выполнение происходило на четырех ядерном процессоре с архитектурой Zen 2, тактовой частотой 3.2 ГГц и поддержкой виртуализации. Зависимость среднего времени выполнения от размера массива показана на рисунке 1.

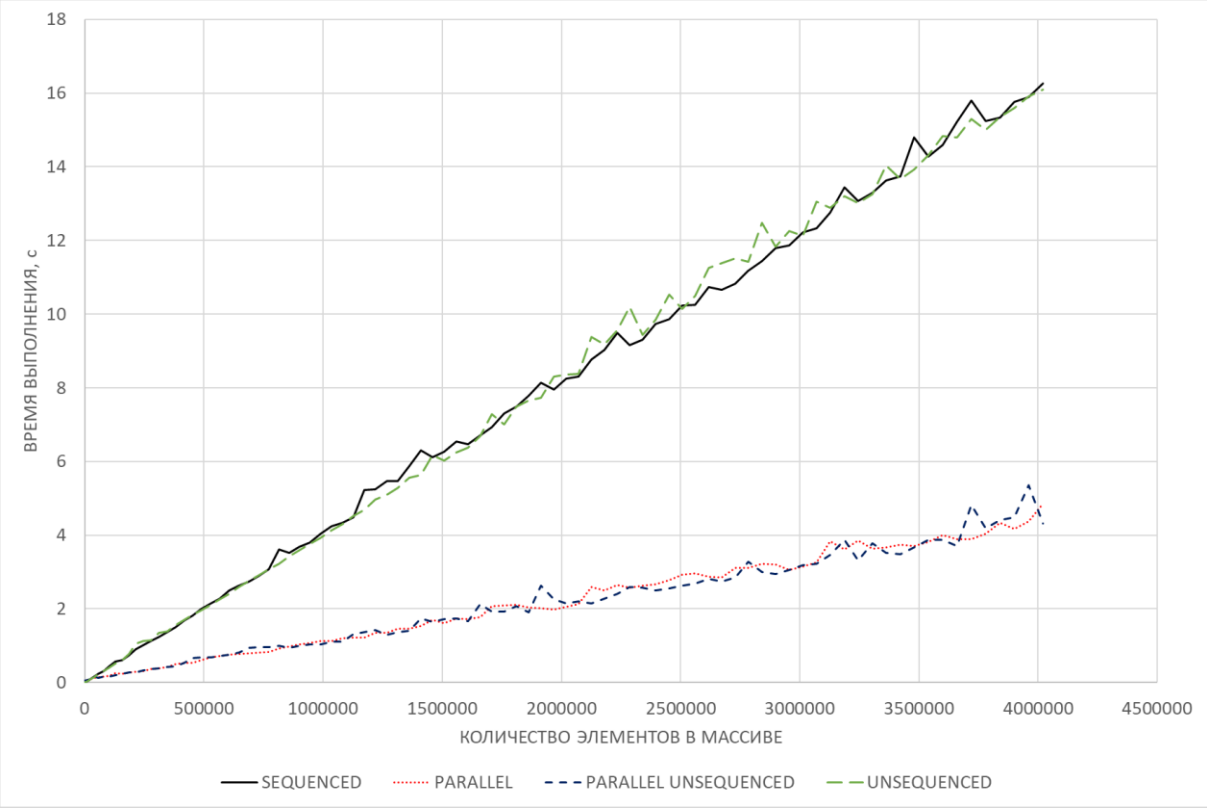

Рис. 1. Зависимость времени выполнения алгоритма трансформации от числа элементов массива на четырех политиках выполнения

Как видно из графика, последовательность (упорядоченность) выполнения практически не оказывает влияния на скорость трансформации. В среднем скорость выполнения при использовании параллельной политики выше, чем при использовании последовательной, в 3.2 раза. При этом для массивов размером менее 20000 элементов ускорение составило в среднем всего 0.22 раз, т.е. последовательный однопоточный вариант работал быстрее. При дальнейшем росте количества элементов до 600000 ускорение росло и в среднем составило 2.71 раз. После данного количества элементов ускорение оставалось примерно на одном уровне в 3.70 раз. Исходя из базового закона Амдала [2] и количества ядер процессора, можно сделать вывод о целесообразности использования параллельной версии данного алгоритма если размер массива более 600000 элементов.

Отметим, что на графиках отсутствует ускорение, вызванное переходом от политики sequenced к политике unsequenced. Это вызвано тем, что средства векторизации вычислений не были применены. Подобное обосновано использованным компилятором и отсутствием требований к обязательности векторизации со стороны стандарта языка [1].

Задача по сортировке массива была выполнена с использованием функции std::sort из библиотеки Algorithm support library. Данная функция реализует интроспективную сортировку, сложность которой в худшем случае и в среднем $n\ log\ (n)$, где n – количество элементов в массиве. Количество прогонов и остальные параметры программы и компьютера идентичны таковым в первой задаче. Зависимость среднего времени сортировки от размера массива показана на рисунке 2.

В данном случае упорядоченность выполнения также практически не оказывает влияния на скорость выполнения. При количестве элементов в массиве меньшем 3000 коэффициент ускорения в среднем составляет 0.42, т.е. параллельная версия алгоритма работает медленнее последовательной. При количестве элементов от 3 до 10 тысяч коэффициент ускорения больше единицы и в среднем составляет 2.41. При количестве элементов, превышающем 10 тысяч, коэффициент ускорения прекращает рост и в среднем составляет 3.39. Из этого следует, что использование параллелизации при сортировке стандартными методами C++ имеет смысл только при больших количествах (более 10000) элементов в сортируемых массивах.

Таким образом, в языке программирования C++ имеются инструменты как для написания своих параллельных алгоритмов, так и для параллелизации имеющихся в стандартной библиотеке. При этом параллелизация стандартных алгоритмов может быть достигнута простым применением политики выполнения, но при этом не гарантируется увеличение производительности.

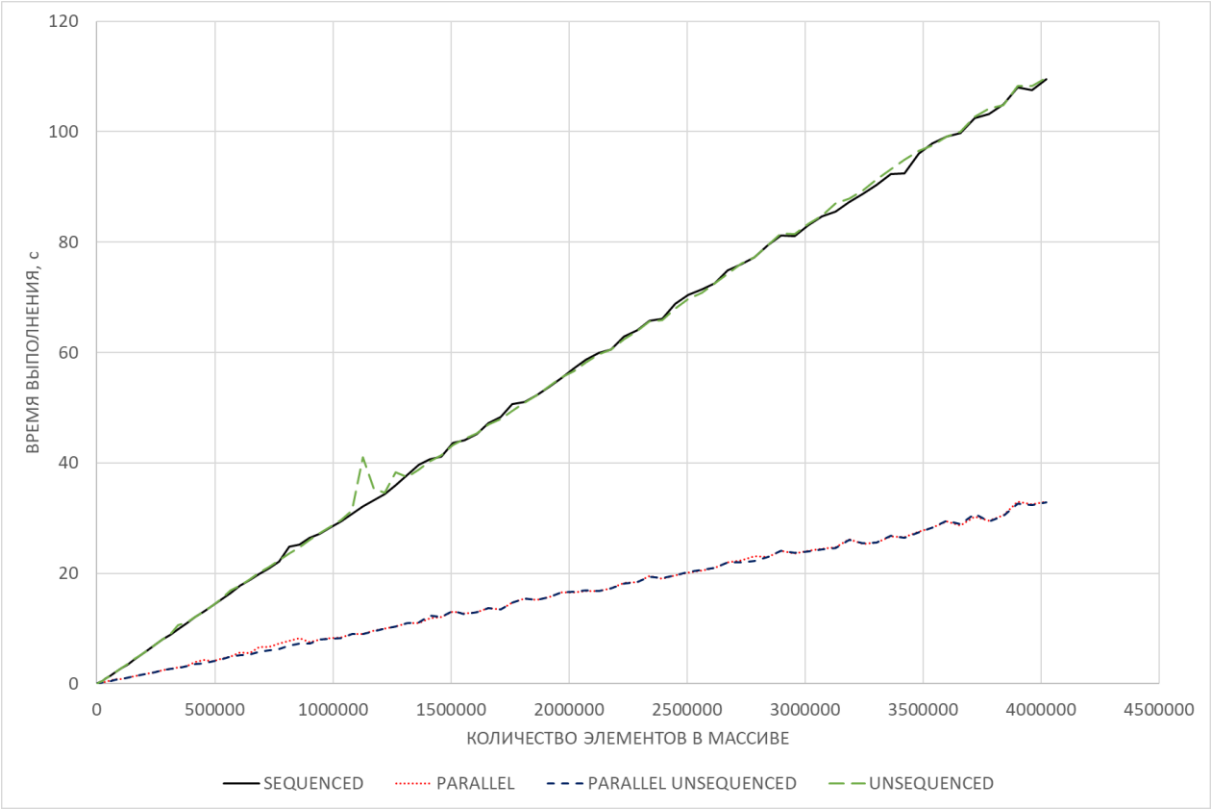

Рис. 2. Зависимость времени сортировки элементов массива от числа элементов на четырех политиках выполнения

Литература

1. Языки программирования. C++ : ISO/IEC 14882:2020. – Взамен ISO/IEC 14882:2017; введ. РБ 15.12.2020. – Минск : Белорус. гос. ин-т стандартизации и сертификации, 2020. – 1866 с.

2. Прихожий А.А.. Распределенная и параллельная обработка данных. – Минск : БНТУ, 2016. – 91 с.

УДК 004.942

ПРОЕКТ И КОМПОНЕНТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ОТКАЗОУСТОЙЧИВОЙ СЕТИ

Купченко А.И.
Научный руководитель – Белова С.В., старший преподаватель

В рамках данной работы была поставлена цель проектирования отказоустойчивой сети и системы защиты информации для РУП «Белтелеком», что включает в себя создание защитных мер, способных