ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ИСПОЛЬЗОВАНИЕ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ УСКОРЕНИЯ МАССОВЫХ ОПЕРАЦИЙ С ОДНОРОДНЫМИДАННЫМИ

А. П. Азявчиков

Белорусский государственный университет, пр. Независимости, 4, 220030, г. Минск, Беларусь, azyavchikov.alex@gmail.com
Научный руководитель — И. Д. Лукьянов, старший преподаватель

В статье описаны подходы по ускорению массовых операций с массивом однородных даных, использующие параллельные алгоритмы, то есть алгоритмы, которые могут быть реализованы по частям на множестве различных вычислительных устройств, а также приводятся описания реализованных многопоточных структур.

Ключевые слова: многопоточные алгоритмы; массовые операции с однородными данными; модификация данных на отрезке; дерево отрезков.

ВВЕДЕНИЕ

В современном информационном обществе, где объемы данных постоянно возрастают, эффективная обработка и анализ информации становятся все более важными задачами для разработчиков программного обеспечения. В этом контексте параллельные алгоритмы выделяются как один из методов ускорения операций с данными. В данной работе описана реализация на языке C++ многопоточной шаблонной структуры над массивом однородных данных, предоставляющей пользователю интерфейс для модификации данных на отрезке и получения состояния отрезка.

Определение 1. Работой многопоточного вычисления называется общее время выполнения всего вычисления на одном процессоре, исполняющим лишь один поток в каждый момент времени.

Определение 2. Интервалом многопоточного вычисления называется наибольшее время выполнения одного потока на идеальном процессоре, то есть процессоре, исполняющим потенциально бесконечно большое число потоков в каждый момент времени.

ПЕРВЫЙ ВАРИАНТ РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНОЙ СТРУКТУРЫ ДАННЫХ

Для удобной работы с несколькими потоками был реализован примитив синхронизации буферизированный канал, предоставляющий пользователю интерфейс отправки данных в канал и получения данных из канала. При попытке отправить данные в канал с заполненным буфером или получить данные из канала с пустым буфером поток будет заблокирован до того момента, пока это не станет возможно сделать, то есть пока другой поток не заберет данные из канала, или не положит их в канал соответственно, или пока канал не будет закрыт.

Пусть теперь задана политика модификации и массив однородных данных ($n \geqslant t$, здесь и далее: n – размер массива однородных данных, t – число используемых потоков). Разобьем исходный массив на подмассивы подряд идущих элементов размера $\lfloor n/t \rfloor$, а оставшиеся элементы положим в последний подмассив. Далее для каждого подмассива создадим дерево отрезков и поток, его обрабатывающий, а также создадим $\lfloor n/t \rfloor$ буферизированных каналов, хранящих в себе запросы модификации и состояния, каждый из которых будет отвечать за передачу запросов к одному подмассиву. Внутри одного потока будем ожидать появления новых запросов к подмассиву из канала, пока канал не будет закрыт. Получив новый запрос и исполнив его, увеличим глобальную атомарную переменную (счетчик числа потоков, выполнивших запрос). Таким образом, получив от пользователя запрос, нужно перенаправить запрос в соответствующие каналы и дождаться, пока значение счетчика не станет равным t.

Таким образом, работа запросов составит $O(t + t \cdot log(n/t))$, так как всего совершается t запросов к деревьям, каждый из которых работает за $O(log(n/t))$, а также выдается t запросов потокам (операции с каналом работают за $O(1)$). А интервал запросов составляет $O(t + log(n/t))$ так как каждый поток совершает запрос к одному дереву отрезков, а самый медленный ещё дожидается $O(t)$ времени выдачи ему запроса.

Для измерения времени работы использовались тесты, работающие с типом данных int и операциями присваивания и суммы в качестве операций модификации и состояния соответственно. Чтобы протестировать работу структуры в более медленном сценарии, была реализована политика модификации, замедляющая все операции с int на 500 нс (такие тесты в дальнейшем будем называть тестами на медленных данных).

Тестирование показало, что структура работает быстрее эталонной реализации (дерева отрезков) только на операциях модификации медленных данных (см. табл. 1, табл. 2), что объясняется медленной передачей запросов через каналы за счет совершения системных вызовов для уведомления потоков.

ВТОРОЙ ВАРИАНТ РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНОЙ СТРУКТУРЫ ДАННЫХ

В данной реализации мы избавились от долгих системных вызовов в буферизированном канале, заменив канал на неблокирующий примитив синхронизации: буферизированную очередь, обладающую аналогичным интерфейсом. Данная версия отличается также тем, что потоки после своего создания и до уведомления их о необходимости завершения работы постоянно проверяют соответствующую очередь на заполненность, и в момент, когда появится новый запрос, каждый поток начинает его исполнение. При этом теоретические временные оценки остаются теми же.

Тестирование показало, что данная структура работает быстрее эталонной реализации на операциях модификации быстрых данных и на медленных данных (см. табл. 1, табл. 2), а также видно, что данная реализация работает существенно быстрее первой.

ТРЕТИЙ ВАРИАНТ РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНОЙ СТРУКТУРЫ ДАННЫХ

В данной версии, в отличие от предыдущих, потоки обработчики сами определяют наличие нового запроса путем постоянного сканирования глобальной атомарной переменной, отвечающей за запрос, и, когда эта переменная изменяется, поток начинает исполнение нового запроса, а пользователь дожидается, пока все потоки не закончат исполнение.

Работа запросов составляет $O(t \cdot log(n/t))$, так как всего совершается t запросов к деревьям отрезков. А интервал составляет $O(log(n/t))$, так как каждый поток совершает запрос к одному дереву отрезков.

Тестирование показало, что данная реализация дает существенный выигрыш на операциях с медленными данными, а также работает лучше предыдущей на запросах состояния (см. табл. 1, табл. 2).

МОДИФИКАЦИЯ ТРЕТЬЕЙ РЕАЛИЗАЦИИ

В данной версии, в отличие от предыдущей, для каждого подмассива будет построено не однопоточное дерево отрезков, а третий вариант параллельной структуры данных. Далее будем обозначать t_1 –число потоков исходной структуры, t_2 – число потоков в третьей структуре.

Работа запросов составит $O(t_1 \cdot t_2 \cdot log(n \,/\, (t_1 \cdot t_2)))$, так как всего совершается $t_1 \cdot t_2$ запросов к деревьям отрезков над массивами размера $O(n \,/\, (t_1 \cdot t_2))$,. А интервал составит $log(n \,/\, (t_1 \cdot t_2)$, так как каждый поток совершает запрос к одному дереву отрезков.

Тестирование показало, что данная реализация дает существенный выигрыш на операциях с медленными данными, однако работает хуже предыдущей на всех типах операций (см. табл. 1, табл. 2).

ЧЕТВЕРТАЯ РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОЙ СТРУКТУРЫ ДАННЫХ

Данная реализация аналогично предыдущим версиям разбивает исходный массив на подмассивы, но использует всего 3 потока для их обработки. Идея в том, что если какой-то запрос затрагивает более 3 подмассивов, то теоретическую оценку в логарифм дают крайние подмассивы, а подмассивы между ними обрабатываются за $O(1)$ (по свойству дерева отрезков). Поэтому в данной реализации тяжелые задачи на левой и правой границе запроса выдаются двум потокам, а легкие промежуточные – оставшемуся потоку. Далее введем обозначение s – (число подмассивов, на которые разбит исходный массив) минус 2. То есть это то количество легких задач в худшем случае.

Работа запросов к структуре составляет $O(s + 2 \cdot log(n/(s+2)))$, так как всего совершается 2 долгих запроса к деревьям отрезков над массивами размера $O(n/(s+2))$ (каждый работает за $O(log(n/(s+2)))$, а также выдается s легких запросов одному потоку, работающих за $O(1)$. А интервал составляет $O(max(log(n/(s+2)), s))$, так как два потока отработают за $log(n/(s+2))$, а оставшийся – за $O(s)$.

Таблица 1

Относительная разница лучших временных показателей между эталоном и разными версиями на массиве быстрых данных размера 2 · 10^7

Версия	Запросы модификации	Запросы состояния
Первая (t = 2)	+49 %	+782 %
Вторая (t = 2)	-2 %	+419 %
Третья (t = 2)	+2 %	+17 %
Модификация третьей ($t_1 = 2, t_2 = 2$)	+8 %	+75 %
Четвертая (s = 2)	+12 %	+10 %
Четвертая (s = 3)	+2 %	+24 %

Таблица 2

Относительная разница лучших временных показателей между эталоном и разными версиями на массиве медленных данных размера 2 · 10^7

Версия	Запросы модификации	Запросы состояния
Первая (t = 5)	-26 %	+4 %
Вторая (t = 4)	-22 %	-1 %
Третья (t = 4)	-24 %	-20 %
Модификация третьей ($t_1 = 2, t_2 = 2$)	-19 %	-13 %
Четвертая (s = 2)	-20 %	-19 %
Четвертая (s = 3)	-26 %	-25 %

Тестирование показало, что параметры s = 3 или s = 2 – наилучшие. При s = 3 данная реализация работает на медленных данных лучше всех предыдущих (см. табл. 2) и использует при этом всего 3 потока.

Библиографические ссылки

1. Introduction to algorithms / Cormen T. H. [et al.], 2022.

2. C++ documentation [Electronic resource]. Mode of access: https://en.cppreference.com/w/. Date of access: 16.05.2024.

3. Segment tree [Electronic resource]. Mode of access: http://e-maxx.ru/algo/segment_tree. Date of access: 16.05.2024.

4. Тель Ж. Введение в распределенные алгоритмы, 2009.