УДК 004.8

Є.В. Томчук
В.О. Денисюк

ДОСІДЖЕННЯ ТА РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО АЛГОРИТМУ СОРТУВАННЯ TIM SORT

Вінницький національний технічний університет

Анотація

Розглянуто розробку паралельного алгоритму сортування на основі TimSort з використанням механізмів багатопоточності C#. Виконано аналіз існуючих методів сортування, обґрунтовано вибір алгоритму TimSort як базового завдяки його гібридній природі та високій ефективності на частково впорядкованих наборах даних. Розроблено програмну реалізацію паралельного варіанта TimSort із використанням бібліотеки Task Parallel Library. Наведено UML-діаграми програмного модуля, описано структуру розробленої системи та обґрунтовано вибір програмного середовища реалізації. Проведено тестування продуктивності розробленого паралельного алгоритму на різних наборах вхідних даних. Отримані результати демонструють покращення швидкодії під час обробки великих масивів та зростання продуктивності при збільшенні кількості потоків, що підтверджує ефективність використання паралельного TimSort в задачах, які потребують інтенсивної обробки даних.

Ключові слова: *TimSort, паралельне сортування, C#, TPL, продуктивність*.

Abstract

The development of a parallel sorting algorithm based on TimSort using C# multithreading mechanisms is considered. Existing sorting methods were analyzed, and the choice of TimSort as the baseline algorithm was justified due to its hybrid nature and high efficiency on partially ordered datasets. A software implementation of a parallel version of TimSort was developed using the Task Parallel Library. UML diagrams of the software module were created, the architecture was described, and the choice of the implementation environment was justified. Performance testing of the developed parallel algorithm was conducted on various input datasets. The results demonstrate performance improvement when processing large arrays and increasing throughput with a higher number of threads, confirming the effectiveness of parallel TimSort in data-intensive tasks.

Keywords: *TimSort, parallel sorting, C#, TPL, performance*.

Вступ

Актуальність реалізації паралельного алгоритму сортування Tim Sort полягає в тому, що сучасні обчислювальні системи оперують величезними масивами даних, для яких швидкість сортування є критично важливим фактором продуктивності. Таких сфер як – машинне навчання, аналітика великих даних, обробка сигналів, оптимізаційні завдання, системи реального часу – вимагають використання алгоритмів, здатних ефективно працювати як із випадковими, так і частково впорядкованими наборами даних.

Tim Sort є адаптивним гібридним алгоритмом, який поєднує сортування вставками та сортування злиттям, завдяки чому демонструє високу швидкодію на практичних наборах даних. Однак класична версія алгоритму реалізована послідовно та не використовує переваг багатоядерних процесорів, що обмежує його потенційні можливості прискорення.

Використання паралельних обчислювальних архітектур дозволяє одночасно обробляти незалежні підпослідовності масиву, виконувати паралельне злиття та значно зменшувати загальний час сортування. Це забезпечує кращу масштабованість і високу ефективність для розв’язання складних обчислювальних задач, у яких обробляються великі обсяги інформації.

Паралельна реалізація Tim Sort дозволяє розділяти дані між потоками, виконувати незалежні етапи сортування та об’єднувати проміжні результати в кінцеву відсортовану послідовність. Це особливо

корисно в сучасних програмних середовищах, де багатопотоковість є базовою функціональністю, а паралелізм суттєво покращує швидкодію та продуктивність програмних додатків.

Метою роботи є розробка паралельного алгоритму сортування Tim Sort мовою C# з використанням багатопоточних засобів .NET для підвищення швидкодії та продуктивності обчислювальних систем.

Постановка задачі дослідження

Задачі дослідження полягають у вирішенні наступних питань: розробка ефективної архітектури паралельного алгоритму Tim Sort для оптимального розподілу обчислень між потоками; визначення та мінімізація накладних витрат на створення потоків і синхронізацію при виконанні паралельного злиття; розробка програмної реалізації паралельного Tim Sort мовою C# із використанням багатопотокових механізмів .NET; проведення експериментального тестування програми на різних наборах даних і аналіз отриманих результатів за часовими та ефективнісними показниками.

Виклад основного матеріалу

Сортування є однією з базових операцій у програмуванні та обробці даних і широко використовується у базах даних, аналітичних системах, комп'ютерній графіці та алгоритмах машинного навчання [1, 2]. Серед алгоритмів сортування виділяють такі класичні методи, як Bubble Sort, Insertion Sort, Merge Sort, Quick Sort та гібридні алгоритми, зокрема Tim Sort. Tim Sort поєднує сортування вставками та злиття, що дозволяє адаптивно обробляти частково відсортовані масиви та забезпечує стабільність результату.

Складність алгоритму Tim Sort залежить від структури вхідних даних. Для середнього та випадку з великим обсягом елементів вона оцінюється як $O(n\,logn)$, що робить алгоритм ефективним для сортування великих масивів. У випадку вже відсортованих даних складність може бути майже лінійною $O(n)$, завдяки адаптивності алгоритму. Пам'яттєві витрати складають $O(n)$ через використання допоміжного буфера для злиття [3].

Для підвищення продуктивності при роботі з великими масивами застосовується паралельна реалізація Tim Sort. Паралелізація базується на природній структурі алгоритму, де масив розбивається на відсортовані підмасиви, так звані runs. Кожен run може бути оброблений незалежно, що дозволяє ефективно використовувати багатоядерні процесори.

Перший етап паралельного Tim Sort полягає у сортуванні окремих run [4]. Кожен підмасив обробляється методом вставок (Insertion Sort) окремим потоком. Час, необхідний для сортування всіх run у паралельному режимі, можна оцінити формулою:

$$T_{runs} = \frac{\sum_{i=1}^{m} |R_i|^2}{p}, \tag{1}$$

де T_{runs} — загальний час на паралельне сортування "runs", p — кількість паралельних потоків або ядер, $\sum_{i=1}^{m} |R_i|^2$ — сума квадратів довжин всіх підмасивів, що враховує роботу Insertion Sort.

Другий етап — злиття run у більші відсортовані підмасиви. Злиття проводиться паралельно, при цьому кожна пара run об'єднується незалежним потоком. Час одного раунду злиття оцінюється як:

$$T_{merge-round} \approx \frac{\sum_{i=1}^{\lceil \frac{m}{2} \rceil} (|R_{\{2i-1\}}| + |R_{\{2i\}}|)}{p} = \frac{n}{p} \tag{2}$$

де n — загальна кількість елементів у масиві.

Кількість раундів злиття визначається як $k = \lceil \log_2 m \rceil$, а загальний час злиття дорівнює:

$$T_{merge} \approx \frac{n \cdot \lceil \log_2 m \rceil}{p} \tag{3}$$

При паралельній реалізації особлива увага приділяється оптимізації пам’яті та динамічному керуванню потоками. Використання допоміжних буферів зменшує накладні витрати на копіювання даних, а кількість одночасних потоків обирається так, щоб не перевантажувати процесор..

Програмна реалізація паралельного Tim Sort на платформі .NET Framework мовою C# дозволяє розподіляти обчислення між потоками та досягати скорочення часу виконання для великих масивів.

Для оцінки ефективності алгоритму Parallel Tim Sort проведено експериментальне тестування на різних типах даних: випадкові масиви (Random), вже відсортовані (Sorted), зворотно відсортовані (Reversed) та частково відсортовані (PartiallySorted). У кожному випадку перевірялося виконання стандартного Array.Sort та Parallel Tim Sort із різною кількістю потоків (1, 12, 24). Результати наведено у таблицях 1–2.

Таблиця 1 – Random, Sorted

Розмір масиву	Random				Sorted			
	Потоків	Array. Sort (ms)	Parallel Tim Sort (ms)	Прискорення	Потоків	Array. Sort (ms)	Parallel Tim Sort (ms)	Прискорення
100,000	1	3.59	11.17	0.32	1	0.56	0.97	0.58
100,000	12	3.59	18.25	0.20	12	0.56	1.17	0.48
100,000	24	3.59	13.85	0.26	24	0.56	0.93	0.60
1,000,000	1	41.96	133.34	0.31	1	5.52	9.40	0.59
1,000,000	12	41.96	34.09	1.23	12	5.52	9.61	0.57
1,000,000	24	41.96	117.30	0.36	24	5.52	9.46	0.58
10,000,000	1	480.20	1563.81	0.31	1	64.73	94.51	0.68
10,000,000	12	480.20	332.62	1.44	12	64.73	94.70	0.68
10,000,000	24	480.20	326.24	1.47	24	64.73	94.83	0.68

Таблиця 3 – Reversed, PartiallySorted

Розмір масиву	Reversed				PartiallySorted			
	Потоків	Array. Sort (ms)	Parallel Tim Sort (ms)	Прискорення	Потоків	Array. Sort (ms)	Parallel Tim Sort (ms)	Прискорення
100,000	1	0.76	8.46	0.09	1	2.78	7.87	0.35
100,000	12	0.76	5.01	0.15	12	2.78	4.73	0.59
100,000	24	0.76	4.37	0.17	24	2.78	4.47	0.62
1,000,000	1	8.20	89.00	0.09	1	34.90	96.32	0.36
1,000,000	12	8.20	24.41	0.34	12	34.90	33.91	1.03
1,000,000	24	8.20	23.83	0.34	24	34.90	30.63	1.14
10,000,000	1	92.55	1001.11	0.09	1	408.85	1193.57	0.34
10,000,000	12	92.55	190.99	0.48	12	408.85	293.89	1.39
10,000,000	24	92.55	189.56	0.49	24	408.85	274.64	1.49

Проаналізувавши отримані експериментальні результати (табл. 1–2), можна зробити наступні висновки.

- Random. Алгоритм Parallel Tim Sort добре масштабувався для великих масивів, демонструючи максимальне прискорення при 24 потоках для масиву з 10 млн елементів (прискорення 1.47). Для малих масивів додаткові потоки не давали виграшу у швидкодії через накладні витрати на управління потоками та синхронізацію.
- Sorted. Для вже відсортованих даних паралельна версія практично не забезпечувала прискорення, оскільки Tim Sort ефективно обробляє готові run, а додаткові потоки лише підвищують накладні витрати (прискорення близько 0.57–0.60).
- Reversed. Це найскладніший сценарій для паралельного сортування, особливо для малих масивів, де прискорення було мінімальним (до 0.09). Для великих масивів із застосуванням 12–24 потоків спостерігалося часткове прискорення (до 0.49).

- PartiallySorted. Паралельний Tim Sort показав середню ефективність, при цьому для великих масивів і великої кількості потоків досягалося суттєве прискорення (до 1.49), що свідчить про адаптивність алгоритму до частково впорядкованих даних.

Отже, ефективність Parallel Tim Sort значною мірою залежить від розміру масиву, характеру його впорядкування (випадковий, відсортований, зворотно відсортований або частково відсортований) та кількості потоків. Алгоритм демонструє високу продуктивність для великих масивів із випадковою або частково відсортованою структурою даних, тоді як для малих або вже відсортованих масивів додаткові потоки можуть знижувати швидкодію через накладні витрати на синхронізацію та управління потоками [5–8].

Висновки

Досліджено алгоритм Tim Sort як ефективний метод сортування великих масивів даних, визначено його основні сфери застосування у програмуванні та аналітичних системах. Розглянуто класичну послідовну версію Tim Sort та її паралельну модифікацію, описано принципи їх реалізації та порівняно ефективність у різних умовах.

На основі сучасних підходів проаналізовано механізми паралельного виконання алгоритмів сортування у багатопоточному середовищі .NET, визначено можливості розподілу обчислень для оптимізації часу виконання та використання апаратних ресурсів.

Реалізовано паралельний Tim Sort на мові C#, описано структуру програми та її компоненти, наведено UML-діаграми для наочності. Проведено тестування алгоритму на різних типах даних та обсягах, проаналізовано результати, а також розраховано коефіцієнти прискорення та ефективності.

Встановлено, що збільшення кількості потоків дозволяє значно скоротити час сортування для великих масивів, проте надмірна кількість потоків знижує ефективність через накладні витрати на синхронізацію та управління потоками.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Source Wikipedia, LLC Books. Sorting Algorithms: Sorting Algorithm, Merge Sort, Radix Sort, Insertion Sort, Heapsort, Selection Sort, Shell Sort, Bucket Sort. Нью-Йорк : General Books LLC, 2010. 238 с.
2. Бхаргава А. Grokking Algorithms: An illustrated guide for programmers and other curious people. Нью-Йорк : Simon and Schuster, 2016. 256 с.
3. Wikipedia. TimSort Algorithm. URL: https://uk.wikipedia.org/wiki/Timsort/.
4. GitHub. TimSort Algorithm in C#. URL: https://gist.github.com/indication/1299013/.
5. GeeksforGeeks. TimSort Algorithm. URL: https://www.geeksforgeeks.org/dsa/timsort/.
6. Kirupa. All About Timsort. URL: https://www.kirupa.com/sorts/timsort.htm.
7. Medium. Tim Sort: Powering Through Real-World Data Sorting. URL: https://medium.com/@serene_mulberry_tiger_125/tim-sort-powering-through-real-world-data-sorting-c4940fd7c363.
8. Parallel programming in .NET: A guide to the documentation. URL: https://learn.microsoft.com/uk-ua/dotnet/standard/parallel-programming/.

Томчук Євген Віталійович – студент кафедри комп`ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: 2kn23b.zhenya@gmail.com;

Денисюк Валерій Олександрович – канд. техн. наук, доцент, доцент кафедри комп'ютерних наук, Вінницький національний технічний університет, м.Вінниця, e-mail: vad64@i.ua.

Tomchuk Yevhen Vitaliyovych – student of Computer Science Department, Faculty of Intelligent Information Technologies and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: 2kn23b.zhenya@gmail.com;

Denysiuk Valerii Olexandrovich – Ph.D., Assistant Professor, Assistant Professor of the Chair of Computer Science, Vinnytsia National Technical University, Vinnytsia, e-mail: vad64@i.ua