Федеральное государственное
автономное образовательное
учреждение высшего образования
«Национальный исследовательский
университет ИТМО»

Факультет программной инженерии и компьютерной техники

Дисциплина: Высокопроизводительные системы

Лабораторная работа 1

Вариант 19

Выполнила: Зырянова Анастасия Денисовна
Группа: P34012

г. Санкт-Петербург
2021

Цели

Реализовать алгоритмы сортировки в соответствии с вариантом задания. Параллелизовать реализованные алгоритмы с помощью OpenMP., применяя для каждого общий набор из разных подходов для распределения частей работы и объединения результатов. Собрать статистику о времени работы реализованных алгоритмов в целом и отдельных этапов в реализациях для различных наборов входных данных в сочетании с различными степенями параллелизма. Сравнить все полученные реализации: разные алгоритмы попарно для каждого подхода и отдельно все реализации каждого алгоритма между собой. Объяснить результаты.

Задачи

Вариант 19	Алгоритм 1 (1)	Алгоритм 2 (3)
	Timsort	Быстрая сортировка

1. Изучить алгоритмы сортировки.
2. Изучить директивы OpenMP.
3. Реализовать алгоритмы.
4. Подготовить тестовые данные.
5. Запуск и анализ результатов.

Описание работы и аспекты реализации

Первым алгоритмом сортировки является timsort.
Timsort— гибридный алгоритм сортировки, сочетающий сортировку вставками и сортировку слиянием. Очень коротко суть алгоритма можно объяснить так:

1. Разделяем массив на подмассивы.
2. Сортируем каждый подмассив сортировкой вставками.
3. Собираем отсортированные подмассивы в единый массив с помощью сортировки слиянием.

Подмассивы должны быть достаточно короткими, чтобы их было выгодно сортировать вставками. А еще чтобы пары последовательностей были примерно одинаковой длины для сортировки слиянием.
В своей работе я реализую алгоритм обычный timsort. Для распараллеливания я использую директивы OpenMP для того же алгоритма.

Вторым алгоритмом сортировки является quicksort.
Суть алгоритма быстрой сортировки выражается в двух шагах:

1. разделить массив на 2 новых — в первый входят элементы меньше заданного, во второй — остальные;

2. рекурсивно отсортировать каждый из новых массивов.

Изначально при тестировании простого и параллельного алгоритма quicksort, параллельная реализация занимала больше времени, чем последовательная. Это было связано с тем, что я использовала директиву task, при которой создаваемые задачи помещаются в пул задач, а свободный поток может подхватить эту задачу, когда освободиться. Но в пул могут помещаться мелкие задачи, а переключение потока к задаче — достаточно сложная операция. Например, наша реализация может поместить в пул задачу для сортировки массива из двух элементов.
Оптимизированный вариант сортировки отличается от обычного лишь тем, что при небольшом (x < 50000, в моем случае) размере массива сортировка выполняется последовательно.
Функция быстрой сортировки вызывается из *параллельной области*. Обрабатываемый массив является общим для всех потоков. Функция должна быть вызвана только один раз — поэтому вызов помещен в область *omp single*.

В ходе реализации были использованы следующие директивы OpenMP:

- Parallel
  Она создает некоторое количество потоков, выполняющих один и тот же код. Параллельный код помещается в фигурные скобки. В конце параллельной области происходит неявная барьерная синхронизация, т.е. все потоки останавливаются до тех пор, пока последний поток не выполнит код параллельной области целиком.
- Single
  Позволяет указать, что раздел кода должен выполняться в одном потоке, а не в главном потоке.
- Task
  Помещается внутрь параллельной области, и задает блок операторов, который может выполняться в отдельном потоке. Задача не создает новый поток. Задача помещается в пул, из которого ее может взять один из свободных потоков для выполнения. Имеет необязательную опцию if, задающую условие. Если условие истинно — будет создана задача и добавлена в пул, если же условие ложно — задача создана не будет, а ассоциированный с ней блок операторов будет немедленно выполнен в текущем потоке.
- Taskwait
  Используется для синхронизации задач. Поток не будет выполнять код, размещенный после taskwait до тех пор, пока не будут выполнены все созданные этим потоком задачи.

Результаты

Исходный код:
https://github.com/Nastennn/hps-1

Кол-во элементов	TimSort, с	Parallel Timsort, с	Разница, %	Quicksort, с	Parallel Quicksort, с	Разница, %
1000000	2,231	1,591	28,68	1,233	0,529	57,11
5000000	12,088	8,675	28,23	6,760	3,208	52,54
10000000	25,021	18,188	27,31	14,173	6,801	52,02
50000000	137,145	103,880	24,26	77,959	47,729	38,78
100000000	284,587	240,056	15,65	163,403	81,386	50,19

Timsort

Generating 50000000 number for timsort
Starting 10 sorts of 50000000 elements. Sort type: timsort
1 sort is done for 13.79s
2 sort is done for 13.67s
3 sort is done for 13.70s
4 sort is done for 13.70s
5 sort is done for 13.68s
6 sort is done for 13.72s
7 sort is done for 13.70s
8 sort is done for 13.72s
9 sort is done for 13.72s
10 sort is done for 13.75s

Mean time is: 13.714533 seconds
Time varies between:
  min_time: 13.665653 seconds
  max_time: 13.794584 seconds
10 sorts of 50000000 elements took 137.145326s

Параллельный Timsort

Generating 50000000 number for parallel_timsort
Starting 10 sorts of 50000000 elements. Sort type: parallel_timsort
1 sort is done for 9.98s
2 sort is done for 9.91s
3 sort is done for 9.96s
4 sort is done for 10.00s
5 sort is done for 10.17s

6 sort is done for 10.46s
7 sort is done for 10.64s
8 sort is done for 10.85s
9 sort is done for 10.94s
10 sort is done for 10.97s

Mean time is: 10.388045 seconds
Time varies between:
  min_time: 9.910046 seconds
  max_time: 10.966363 seconds
10 sorts of 50000000 elements took 103.880446s

Quicksort

Generating 50000000 number for quicksort
Starting 10 sorts of 50000000 elements. Sort type: quicksort
1 sort is done for 7.80s
2 sort is done for 7.79s
3 sort is done for 7.79s
4 sort is done for 7.79s
5 sort is done for 7.79s
6 sort is done for 7.81s
7 sort is done for 7.81s
8 sort is done for 7.79s
9 sort is done for 7.79s
10 sort is done for 7.79s

Mean time is: 7.795944 seconds
Time varies between:
  min_time: 7.790910 seconds
  max_time: 7.812820 seconds
10 sorts of 50000000 elements took 77.959443s

Параллельный Quicksort

Generating 50000000 number for parallel_quicksort
Starting 10 sorts of 50000000 elements. Sort type: parallel_quicksort
1 sort is done for 4.77s
2 sort is done for 4.77s
3 sort is done for 4.77s
4 sort is done for 4.76s
5 sort is done for 4.76s

6 sort is done for 4.78s
7 sort is done for 4.77s
8 sort is done for 4.77s
9 sort is done for 4.77s
10 sort is done for 4.80s

Mean time is: 4.772926 seconds
Time varies between:
  min_time: 4.763506 seconds
   max_time: 4.802837 seconds
10 sorts of 50000000 elements took 47.729263s

Выводы

В ходе выполнения лабораторной работы я реализовала два алгоритма сортировки - timsort и quicksort. Затем я использовала возможности OpenMP для распараллеливания выполнения этих алгоритмов.

При сравнительном анализе простого и параллельного timsort мы можем видеть, что параллельная реализация действительно быстрее. Причем, на разных размерах массива разница в скорости обработки примерно одинаковая. Выделяется только последний тест с размером массива 100000000.

При сравнении обычного и параллельного quicksort изначально у меня не получилась сделать так, чтобы параллельная реализация была быстрее. Однако после некоторого времени изучения материалов в интернете, я нашла способ это изменить. При небольшом размере подмассива сортировка производится последовательно. Это помогло существенно поменять ситуацию - параллельная сортировка практически всегда в 2 раза быстрее.

Таким образом, я поняла, что при правильной реализации параллельная сортировка действительно быстрее последовательной, но это лучше всего это использовать для больших массивов данных, поскольку переключение потока к задаче - очень сложная операция.