МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий математики и механики

Кафедра: Математическое обеспечение и суперкомпьютерные технологии

Направление подготовки: «Фундаментальная информатика и информационные технологии»

ОТЧЁТ

по лабораторной работе

по дисциплине «Параллельное программирование»

на тему:

«Поразрядная сортировка для целых чисел с чётно-нечётным слияние Бэтчера»

Выполнила: студентка гр. 381606-3

Трубина Анастасия Владимировна

Проверил: ассистент кафедры математического обеспечения и суперкомпьютерных технологий, Волокитин Валентин Дмитриевич

Нижний Новгород

2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Поразрядная сортировка для целых чисел

Поразрядная сортировка имеет две модификации:

- Most Significant Digit, MSD (Нисходящая поразрядная сортировка, от старшего разряда к младшему)
- Least Significant Digit. LSD (Восходящая поразрядная сортировка, от младшего разряда к старшему)

В данной работе был применен алгоритм LSD как наиболее эффективный (и подходящий).

Идея поразрядной восходящей сортировки (Least Significant Digit (LSD) Radix Sort) заключается в том, что выполняется последовательная сортировка чисел по разрядам (от младшего разряда к старшему).

Эффективность алгоритма O(k * (N + m)), где k – число разрядов, N – число элементов, m – число возможных значений (число элементов системы счисления)

Т.е. эффективность при правильно реализованном алгоритме линейная.

Общая идея алгоритма быстрой сортировки состоит в следующем:

- На первой итерации сортировки выполняется размещение элементов в отсортированном порядке по младшему разряду чисел
- На следующей итерации сортируются элементы по второму разряду
- Далее выполняется упорядочивание элементов по третьему разряду
- –//–
- На последнем шаге выполняется сортировка по старшему разряду

Поразрядная сортировка массива будет работать только в том случае, если сортировка, выполняющаяся по разряду, является устойчивой (элементы равных разрядов не будут менять взаимного расположения при сортировке по очередному разряду)

Чтобы решить проблему с сортировкой чисел разного знака, в данной работе сделано разделение массива на отрицательный и неотрицательный.

Разбиение «Четно-нечетное слияние Бэтчера»

Чётно-нечётное слияние Бэтчера заключается в том, что два упорядоченных массива, которые необходимо слить, разделяются на чётные и нечётные элементы. Такое слияние может быть выполнено параллельно.

Чтобы массив стал окончательно отсортированным, достаточно сравнить пары элементов, стоящие на нечётной и чётной позициях. Первый и последний элементы массива проверять не надо, т.к. они являются минимальным и максимальным элементов массивов.

Чётно-нечётное слияние Бэтчера позволяет задействовать 2 потока при слиянии двух упорядоченных массивов. В этом случае слияние N массивов могут выполнять N параллельных потоков. На следующем шаге слияние N/2 полученных массивов будут выполнять N/2 потоков и т.д. На последнем шаге два массива будут сливать 2 потока

Рис. 1. Чётно-нечётное слияние Бэтчера

Характеристики компьютера, на котором составлялась и выполнялась программа:

Процессор: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992 МГц, ядер: 4, логических процессоров: 8

RAM: 16 ГБ,

ОС: Windows 10 PRO

Версия: 10.0.17134 Сборка 17134

ТЕСТОВАЯ ВЕРСИЯ

Solution.h

***Описание**: Реализация восходящей поразрядной сортировки, по десятичным разрядам чисел файлы Solution.cpp и main.cpp*

Реализация(*Solution.cpp***):** Вызываемая функция сортировки называется *radix_sort_sol(vector<int>& array_for_sort, const int left_border, const int right_border)*, в которой первый параметр – сортируемый вектор, второй-третий параметры границ сортируемого вектора. В функции вектор делится на положительную часть и отрицательную. Далее используются две дополнительные функции.

Функция *int get_max(vector<int>& array_for_sort, const int left_border, const int right_border)* используется для нахождения в сортируемом фрагменте массива максимальное значение для определения глубины сортировки (кол-во разрядов)

А также функция *void count_sort(vector<int>& array_for_sort, const int left_border, const int right_border, int digit)* которая непосредственно сортирует по определенному разряду.

ОБЩАЯ ИДЕЯ И СТРУКТУРА ПАРАЛЛЕЛЬНЫХ ВЕРСИЙ

Реализованные OpenMP и TBB версии имеют одинаковую структуру, поэтому их можно представить в виде одной схемы:

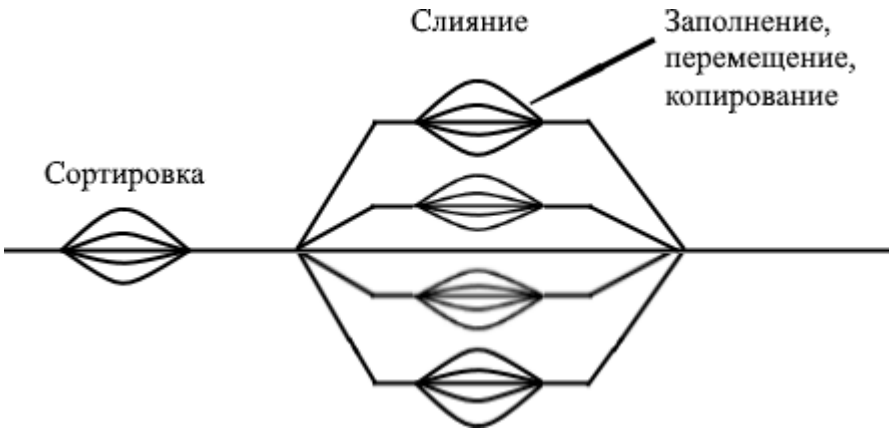


Рис. 2. Схема распараллеливания.

*На этапе «сортировка» в OpenMP сразу массив делится на куски в количестве равном количеству потоков, в TBB при создании дочерних «заданий» когда размер «подмассива» соответствует необходимому разделению на потоки и в обоих случаях «подмассивы» сортируются последовательной версией сортировки (**Solution.h**). После этой части весь массив элементов состоит из отсортированных кусков-«подмассивов».*

На участке «слияние» происходит соответственное последовательное слияние подмассивов (1-2,3-4, и т.д.).

«Слияние» в данном случае подразумевает слияние четных и нечетных элементов подмассивов отдельно и их соответственное (четно-нечетное) слияние.

Схема слияния:

Первый подмассив	-1	10	25	31	48	50	52	64	Второй подмассив	-8	-5	0	2	7	10	19	60
Массив нечетных элементов	-8	-1	0	7	19	25	48	52	Массив четных элементов	-5	2	10	10	31	50	60	64
Почти отсортированный массив	-8	-5	-1	2	0	10	7	10		19	31	25	50	48	60	52	64
Отсортированный массив	-8	-5	-1	0	2	7	10	10		19	25	31	48	50	52	60	64

Рис.3. Схема слияния

Параллельная версия OpenMP

***Описание**: Реализация поразрядной сортировки для целых чисел с четно-нечетным слиянием Бэтчера с использованием средств OpenMP (Parallel.cpp и main.cpp)*

Реализация(Parallel.cpp***):** Вызываемая функция сортировки называется* void parallel_sort(std::vector<int>& vector_for_sorting, int _size, int _threads, double& _time)*, в которой первый параметр – сортируемый вектор, второй – размер сортируемого вектора, третий – количество потоков, четвертый – время.*

Общая структура: сортируем последовательной версией столько частей сколько потоков, далее делаем слияние, пока шагов будет меньше потоков. Шаг рассчитывается специальной формулой. Внутри слияния делаем все по Рис.3.

P.s. алгоритм сделан так чтобы не создавать отдельно каждый проход/шаг векторы для каждого из подмассивов, вместо этого один раз идет конвертация из вектора в массив перед обработкой и из массива в вектор после обработки, это позволяет сэкономить память. Последовательная версия используется из Solution.h

Функция *void MergeAndSort(const std::vector<int> vec_one, const std::vector<int> vec_two, int* output_sorted_array)* используется для простого слияния массивов четных и нечетных элементов и финальное «причесывание» выходного массива

Функция *void select_splitter(elemType type, const int* array_one, int size_one, const int* array_two, int size_two, std::vector<int>& result)* используется для выделения из двух массивов либо четных либо нечетных элементов в упорядоченном виде по enum EVEN или ODD

ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ TBB

***Описание**: Реализация поразрядной сортировки для целых чисел с четно-нечетным слиянием Бэтчера с использованием средств TBB (ParallelTBB.cpp)*

Реализация(ParallelTBB.cpp**):** Вызываемая функция сортировки называется *void Start_TBB_Sorting(int *arr, int size, int threads)*, в которой первый параметр – сортируемый массив, второй – размер сортируемого массива, третий – количество потоков.

Общая структура: Запускаем корневое задание, которое, по сути, рекурсивно вызовет столько дочерних задний сколько есть потоков и когда заданная порция (которая зависит от кол-ва потоков) будет больше числа элементов в подмассиве запустится последовательная версия сортировки из solver. Далее созданные задания также, по сути, делают слияния аналогично пункту ***«Параллельная версия OpenMP»***

В алгоритме используется функция-конвертер которая преобразует массив в вектор и наоборот. Она необходима т.к. получилось реализовать программу только используя массив, а Solution.h версия рассчитана на вектор.

Последовательная версия используется из **Solution.h.h**

Класс *используемый в работе* class SortingBody *необходим для запуска самого тела сортировки: таких же заданий для следующих уровней, выделения массивов четных и нечетных элементов и их распределения в массив, а также запуск последовательной сортировки для кусков необходимого размера.*

Классы class OddSelector *и* class EvenSelector *каждый из которых выбирает из массивов нечетные и четные элементы соответственно в упорядоченном виде и сразу же распределяет их по нечетным и четным местам массива соответственно.*

Функция *void transformed_sort(int* arr, int size)* производящая конвертирование и сортировку последовательной версией.

Класс class Comparator *необходимый для пробега по массиву и проверки что массив отсортирован, если соседние элементы стоят неправильно он их обменивает*

СТАТИСТИКА:

Сортировка проводилась для массивов размерами:

1) 100000; 2) 1000000; 3) 10000000;

элементов для последовательной версии, параллельной версии OMP и параллельной версии TBB для:

1) 1 потока; 2) 2 потоков; 3) 4 потоков; 4) 8 потоков.

Результаты работы представляется в виде значения времени работы сортировки и сравнивались по отношению со временем работы последовательной версии при соответствующих значениях размера массива, а так же внутри самой параллельной версии относительно времени работы при различных значениях количества потоков.

Ниже представлен результат работы последовательной версии. При величине массива < 16 представляется возможность проконтролировать работу сортировки.

```
C:\CLionProjects\Batcher_sort\cmake-build-debug>Batcher_sort
Nice to meet you!
Please, add size of array: 10
Array is creating...
The following array was created:
-273 253 158 704 -853 -893 -906 -486 -850 740
Positive vector:
253 158 704 740
Negative vector:
-273 -853 -893 -906 -486 -850
Sort positive vector:
740 253 704 158
704 740 253 158
158 253 704 740
Sort negative vector:
-850 -273 -853 -893 -906 -486
-906 -850 -853 -273 -486 -893
-273 -486 -850 -853 -893 -906
Reverse negative vector:
-906 -893 -853 -850 -486 -273
Result array is:
-906 -893 -853 -850 -486 -273 158 253 704 740
Time: 0.03c
Metod is true.

C:\CLionProjects\Batcher_sort\cmake-build-debug>
```

1. Последовательная версия.

```
C:\CLionProjects\Batcher_sort\cmake-build-debug>Batcher_sort
Nice to meet you!
Please, add size of array: 100000
Array is creating...
Time: 0.128c
Metod is true.

C:\CLionProjects\Batcher_sort\cmake-build-debug>Batcher_sort
Nice to meet you!
Please, add size of array: 1000000
Array is creating...
Time: 1.214c
Metod is true.

C:\CLionProjects\Batcher_sort\cmake-build-debug>Batcher_sort
Nice to meet you!
Please, add size of array: 10000000
Array is creating...
Time: 12.105c
Metod is true.

C:\CLionProjects\Batcher_sort\cmake-build-debug>Batcher_sort
Nice to meet you!
Please, add size of array: 100000000
Array is creating...
Time: 162.374c
Metod is true.
```

2.1. Параллельная версия OpenMP. N = 100000

```
C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 1
Please, add size of array: 100000
Array is creating...
Time: 0.135702c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 2
Please, add size of array: 100000
Array is creating...
Time: 0.0955372c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 4
Please, add size of array: 100000
Array is creating...
Time: 0.0954683c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 8
Please, add size of array: 100000
Array is creating...
Time: 0.136116c
Metod is true.
```

2.2. Параллельная версия OpenMP. N = 1000000

```
C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 1
Please, add size of array: 1000000
Array is creating...
Time: 1.27185c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 2
Please, add size of array: 1000000
Array is creating...
Time: 0.85045c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 4
Please, add size of array: 1000000
Array is creating...
Time: 0.893872c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 8
Please, add size of array: 1000000
Array is creating...
Time: 1.25761c
Metod is true.
```

2.3. Параллельная версия OpenMP. N = 1000000

```
C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 1
Please, add size of array: 10000000
Array is creating...
Time: 17.8469c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 2
Please, add size of array: 10000000
Array is creating...
Time: 11.7098c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 4
Please, add size of array: 10000000
Array is creating...
Time: 11.0266c
Metod is true.

C:\CLionProjects\Batcher_sort_omp\cmake-build-debug>Batcher_sort_omp
Nice to meet you!
Please, enter number of threads: 8
Please, add size of array: 10000000
Array is creating...
Time: 14.2918c
Metod is true.
```

3.1. Параллельная версия TBB. N = 100000

```
C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 1
Please, add size of array: 100000
Array is creating...
TBB sorting time: 0.006758
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 2
Please, add size of array: 100000
Array is creating...
TBB sorting time: 0.007320
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 4
Please, add size of array: 100000
Array is creating...
TBB sorting time: 0.005708
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 8
Please, add size of array: 100000
Array is creating...
TBB sorting time: 0.005241
Metod is true.
```

3.2. Параллельная версия TBB. N = 1000000

```
C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 1
Please, add size of array: 1000000
Array is creating...
TBB sorting time: 0.058328
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 2
Please, add size of array: 1000000
Array is creating...
TBB sorting time: 0.030372
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 4
Please, add size of array: 1000000
Array is creating...
TBB sorting time: 0.021766
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 8
Please, add size of array: 1000000
Array is creating...
TBB sorting time: 0.024120
Metod is true.
```

3.3. Параллельная версия TBB. N = 1000000

```
C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 1
Please, add size of array: 10000000
Array is creating...
TBB sorting time: 0.554049
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 2
Please, add size of array: 10000000
Array is creating...
TBB sorting time: 0.306026
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 4
Please, add size of array: 10000000
Array is creating...
TBB sorting time: 0.210407
Metod is true.

C:\VisualStudioProjects\Batcher_sort_tbb_vs\x64\Release>Batcher_sort_tbb_vs
Nice to meet you!
Please, enter number of threads: 8
Please, add size of array: 10000000
Array is creating...
TBB sorting time: 0.153966
Metod is true.
```

Таким образом, результаты экспериментов представляются в виде таблицы со значениями времени работы в зависимости от количества потоков и размера массива N (в секундах):

N = 100000			
Потоков	*Последовательная*	*Параллельная OMP*	*Параллельная TBB*
1	*0.128*	*0.135702*	*0.006758*
2		*0.0955372*	*0.007320*
4		*0.0954683*	*0.005708*
8		*0.136116*	*0.005241*
N = 1000000			
1	*1.214*	*1.27185*	*0.058328*
2		*0.85045*	*0.030372*
4		*0.893872*	*0.021766*
8		*1.25761*	*0.024120*
N = 10000000			
1	*12.105*	*12.567*	*0.554049*
2		*8.52097*	*0.306026*
4		*8.65546*	*0.210407*
8		*11.7156*	*0.153966*

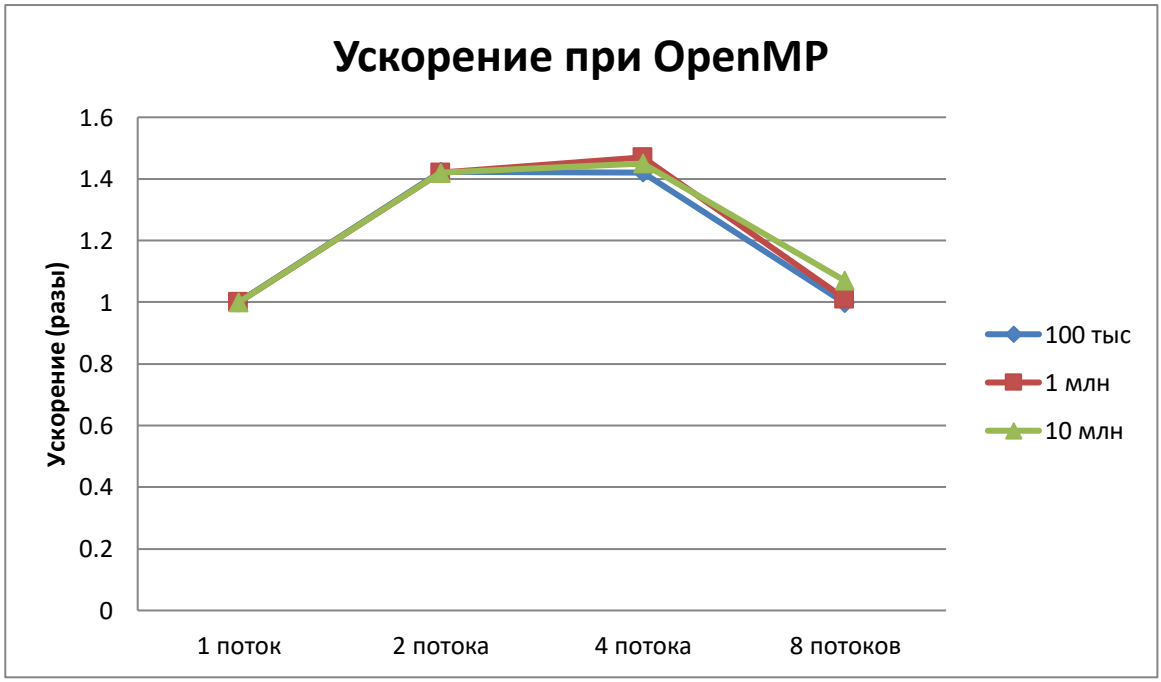
Таким образом ускорение по отношению к последовательной версии при соответствующем значении размера массива N (в разах):

N = 100000		
Потоков	*Параллельная OMP*	*Параллельная TBB*
1	*0,943243*	*18,94051*
2	*1,339792*	*17,48634*
4	*1,340759*	*22,42467*
8	*0,940374*	*24,42282*
N = 1000000		
1	*0,954515*	*20,81333*
2	*1,42748*	*39,97103*
4	*1,358136*	*55,77506*
8	*0,965323*	*50,33167*
N = 10000000		
1	*0,963237*	*21,84825*
2	*1,420613*	*39,55546*
4	*1,398539*	*57,53136*
8	*1,033238*	*78,62125*

Ускорение при данной последовательной версии относительно её работы на 1 потоке представлено в виде таблицы:

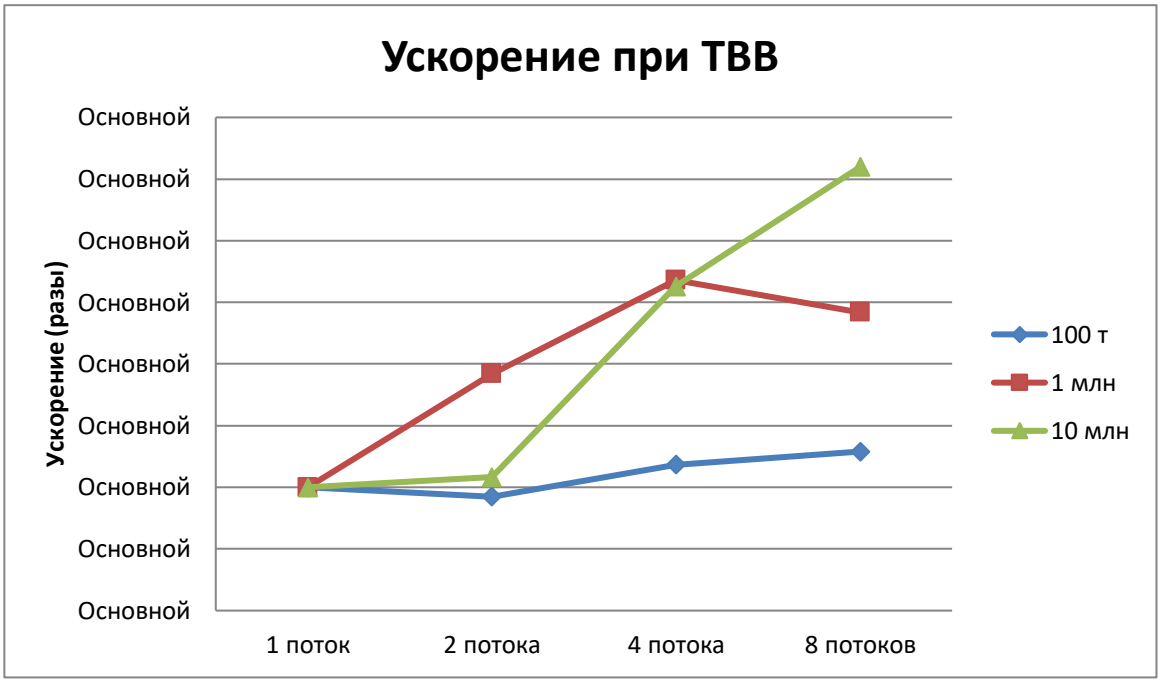
N = 100000		
Потоков	*Параллельная OMP*	*Параллельная TBB*
1	*1*	*1*
2	*1,423*	*0,9232*
4	*1,421*	*1,184*
8	*0,997*	*1,29*
N = 1000000		
1	*1*	*1*
2	*1,42*	*1,92*
4	*1,47*	*2,68*
8	*1,011*	*2,42*
N = 10000000		
1	*1*	*1*
2	*1,421*	*1,081*
4	*1,45*	*2,63*
8	*1,07*	*3,598*

По графикам ниже можно отследить изменение ускорения по каждой из сортировки:



Как видно из графика, у OpenMP наблюдается отсутствие сильной зависимости от размера массива (возможно дело в ограничениях «железа» или в «малом» количестве элементов). Однако, в целом само поведение ускорения здесь более спокойное чем на следующем графике работы TBB. Оптимальным количеством потоков для наилучших показателей ускорения у OpenMP в среднем является 4 потока, причина может быть, в аппаратных составляющих (4-х ядерный

процессор). Хорошо прослеживается тенденция увеличения ускорения при увеличении количества потоков в 2 раза от последовательного исполнения, но приближаясь к значению 4-х потоков, ускорение снижается, а при 8 потоках и вовсе становится схожим со значением при 1 потоке. Причиной тому скорее всего являются накладные расходы, которые увеличиваются при увеличении числа потоков, так как реализованный алгоритм в OpenMP версии не являлся наиболее оптимальным из возможно реализуемых.



У средств TBB похожая ситуация, хотя, здесь отрыв 4 потоков не так очевиден, однако он все же есть. Так же можно заметить, что более высокое ускорение при 4-8 потоках хотя и выдает хорошие результаты, весьма нестабильно и может выдавать как хорошие показатели, так и не очень. В целом же ускорение весьма приличное для проводимого опыта.

Также интересно выяснить какие из средств параллельного программирования – TBB или OpenMP оказались на высоте в данном алгоритме. Посмотрев на значения таблицы и представления графиков сразу становится очевидно, что во всех случаях более высокие показатели у средств TBB, и даже если сравнивать на разных размерах массивов, картина не меняется. И на 2-х, и на 4-х, и на 8-ми потоках TBB более эффективно выполняли сортировку и можно заметить, что при увеличении числа потоков ускорение у TBB становится больше. Это может объясняться различием в реализации распараллеливания – у TBB более эффективно – а так же самим видоизменением алгоритма.

ВЫВОД

В ходе работы была реализована **Поразрядная сортировка для целых чисел с четно-нечетным слиянием Бэтчера** в последовательной и 2-х параллельных версиях, изучены и опробованы такие средства параллельной разработки как ***Open Multi-Processing(OpenMP)*** и ***Intel Threading Building Blocks(TBB)***.

По самой сортировке видно, что ее эффективность вполне соответствует теоретическим ожиданиям, а также видно, что сама последовательная сортировка (Восходящая поразрядная) очень устойчивая и генерация исходных данных массивов не влияют на результат (нет худшего случая при случайном распределении данных)

Также сама параллельная часть т.е. слияние тоже является очень неплохим для варианта слияний.

Масштабируемость по данным:

Сложно оценить, т.к. не хватает данных, и при текущих условиях сложно более глубоко проверить масштабируемость. По текущим же данным при росте размера массива ускорение остается в целом стабильным.

Масштабируемость по потокам:

По результатам эксперимента видно, что при увеличении числа потоков для данной задачи, до 4 ускорение алгоритма быстро растет, при увеличении до 8 либо немного падает, либо остается на том же уровне. Скорее всего причина в том, как именно аппаратная часть реализует работу при восьми потоках на 4 ядрах.

В результате работы была написана программа (набор программ), позволяющая создавать случайные массивы, сортировать их, как последовательным, так и параллельным алгоритмом, а также вести обработку полученных по ним данных. Для любого количества данных и потоков алгоритм является корректным. Проведены серии экспериментов с анализом масштабируемости, показывающие ускорение параллельного алгоритма, достоинства и важность изучения методов параллельных вычислений.

СПИСОК ЛИТЕРАТУРЫ

1. Учебный курс «Технологии параллельного программирования» Лекционные материалы Сиднев А.А., Сысоев А.В., Мееров И.Б: https://software.intel.com/sites/default/files/m/d/4/1/d/8/TBB_Review_doc.pdf
2. Оптимизация вычислительно трудоемкого программного модуля для архитектуры Intel Xeon Phi. Линейные сортировки. Сиднев А.А., Сысоев А.В., Мееров И.Б. http://hpc-education.unn.ru/files/courses/XeonPhi/Lab07.pdf
3. Методическое пособие OpenMP http://ccfit.nsu.ru/arom/data/openmp.pdf
4. Википедия – свободная энциклопедия [Электронный ресурс],- https://ru.wikipedia.org/wiki/Сортировка слиянием Бэтчера статья в интернете.
5. Википедия – свободная энциклопедия [Электронный ресурс],- https://ru.wikipedia.org/wiki/Поразрядная_сортировка
6. Методическое пособие «Параллельное программирование на OpenMP» [Электронный ресурс], - http://ccfit.nsu.ru/arom/data/openmp.pdf.