1. Informations about computer system used:

RAM SIZE: 16 GB (SO-DIMM DDR4, 2666MHz)

CPU: Intel® Core™ i5-8300H

NUMBER OF PHYSICAL CORRES: 4

NUMBER OF LOGICAL CORRES: 8

CLOCK RATE: 2300 - 4000 MHz

CPU CACHE: 8 MB

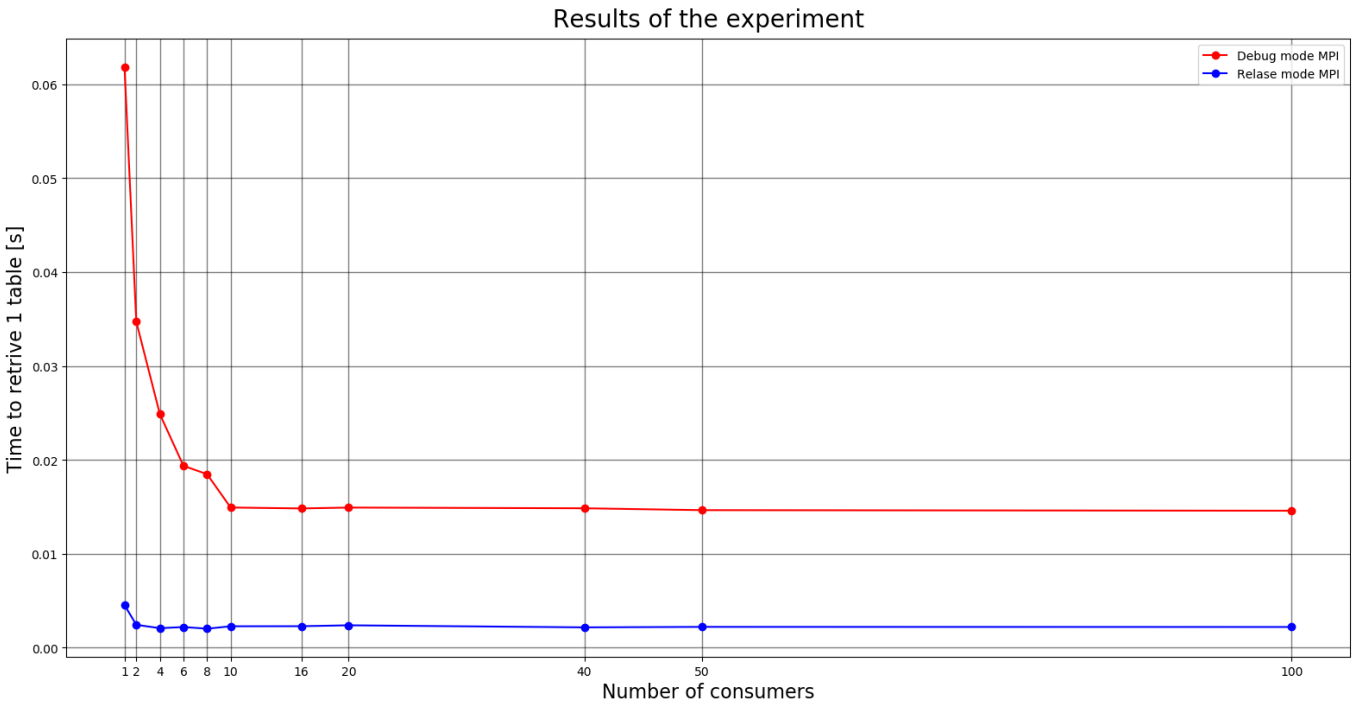
2. Parameters

Compiler parameters			Experiment parameters		
Version	**Mode**	**Architecture**	**Number of sorted arrays**	**Array size**	**Queue size**
Default C++14 for Visual Studio Community 2019 16.5.4	Debug/Relase	X64	5000	100000	1000

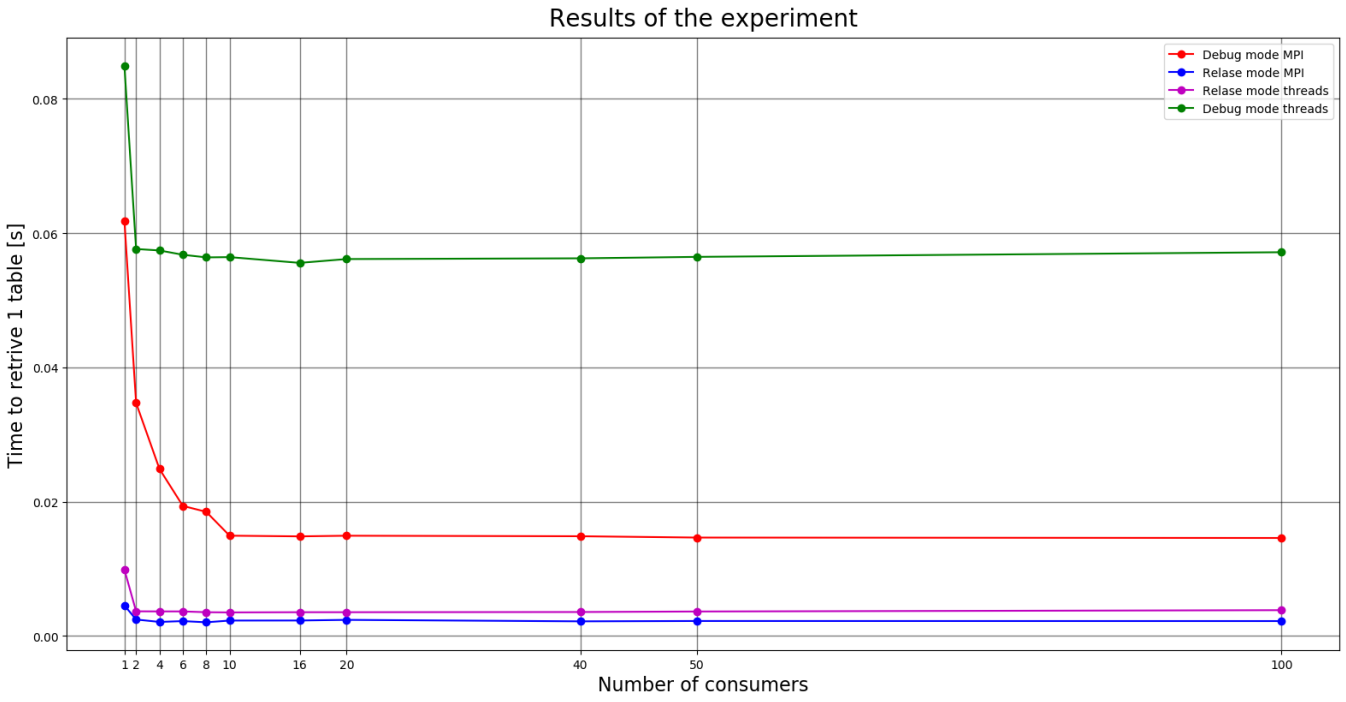
3. Comparison of the speed of retrieving arrays from the queue

Number of consumers	Execution time[s] Debug	Time/1 table[s] Debug	Execution time[s] Relase	Time/1 table[s] Relase	Execution time[s] Threads/ Debug	Time/1 table[s] Threads/ Debug	Execution time[s] Threads/ Relase	Time/1 table[s] Threads/ Relase
1	309.117	0.0618	22.6973	0.0045	424,623	0.0849	49.3699	0.0099
2	173.5	0.0347	12.1875	0.0024	288.186	0.0576	18.2917	0.0037
4	124.336	0.0248	10.3926	0.0021	287.097	0.0574	18.2174	0.0036
6	96.8164	0.0194	11.0098	0.0022	283.889	0.0568	18.2196	00036
8	92.3984	0.0185	10.1172	0.0020	281.936	0.0564	17.6176	0.0035
10	74.6367	0.0149	11.4102	0.0022	282.136,	0.0564	17.4771	0.0035
16	74.125	0.0148	11.4492	0.0023	277.84	0.0556	17.6291	0.0035
20	74.6016	0.0149	11.9258	0.0023	280.688	0.0561	17.6306	0.0035
40	74.2188	0.0148	10.8320	0.0022	281.228	0.0562	17.7723	0.0035
50	73.2305	0.0146	11.0859	0.0022	282.314	0.056	18.1577	0.0036
100	72.9258	0.0146	11.0273	0.0022	285.768	0.0571	19.1523	0.0038

Results of the experiment for MPI approach.



Comparison of times for all experiments



4. Commentary

- The presented results were obtained by the average of five measurements for given number of consumers.
- The best time to sort single table has been achieved for 100 consumers in debug mode and for 8 consumers in relase mode.
- By optimizing the execution of the program in the case of relase mode, best time was obtained for 8 consumers, which corresponds to the number of logical cores.
- From 10 consumers in debug mode and 8 in relase mode, adding more consumers does not bring about a significant improvement in the execution time of a program.
- For both compiler modes execution of program is much worse for one consumer, than any other multiple consumer scenario.
- Comparing the results obtained for a program with multiple threads, the processing speed of one array is significantly lower for MPI implementation.

5. Example of single output

Queue size is : 0

Check sum is: 49.5084

Check sum is: 49.4809

Check sum is: 49.5936

Check sum is: 49.5468

Check sum is: 49.4376

…

…

…

Consumer: 1 sorted: 5000

Execution time: 22.3594

Consumer was destructed

Producer has produced: 5000

Producer was destructed

Queue size is : 0

Queue was destructed

6. Source code

Output of source code was presented above

```
#include <iostream>
#include <mpi.h>
#include <math.h>
#include <thread>
#include <vector>
#include <queue>
#include <array>
```

```
#define ARRSIZE 100000

template< typename T >

class Queue : public std::queue< T >
{
public:
    Queue(int size) : length(size)
    {
        std::cout << "Queue size is : " << std::queue< T >::size() << std::endl;
    }
    ~Queue()
    {
        std::cout << "Queue size is : " << std::queue< T >::size() << std::endl;
        std::cout << "Queue was destructed" << std::endl;
    }

    void push(T& array)
    {
        if (current_length == length)
        {
            std::cout << "Queue is full!" << std::endl;
        }
        else
        {
            std::queue< T >::push(array);
            current_length++;
        }
    }

    void pop()
    {
        if (!std::queue< T >::empty())
        {
            std::queue< T >::pop();
            --current_length;
        }
    }

    const int length;
    int current_length = 0;
};

class Producer
{
public:
    Producer(int n, Queue<std::array<int, ARRSIZE>>& head) : num_of_arrays(n), queue(head)
    {
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    }
    ~Producer()
    {
        std::cout << "Producer has produced: " << push << std::endl;
        std::cout << "Producer was destructed" << std::endl;
    }

    void MakeArrays()
    {
        while (push < num_of_arrays)
        {
            MPI_Status status;
            MPI_Recv(&flag, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            MPI_Send(&finish, 1, MPI_C_BOOL, status.MPI_SOURCE, 1, MPI_COMM_WORLD);
            for (int i = 0; i < arrsize; i++)
            {
                arr[i] = rand() % 100;
            }
            queue.push(arr);
```

```
            push++;
            MPI_Send(&queue.front(), ARRSIZE, MPI_INT, status.MPI_SOURCE, 0, MPI_COMM_WORLD);
            queue.pop();
        }
        while (true)
        {
            finish = 1;
            MPI_Status status;
            MPI_Recv(&flag, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            MPI_Send(&finish, 1, MPI_C_BOOL, status.MPI_SOURCE, 1, MPI_COMM_WORLD);
            send++;
            if (send == size - 1)
            {
                break;
            }
        }
    }

private:
    Queue<std::array<int, ARRSIZE>>& queue;
    std::array<int, ARRSIZE> arr;
    const int num_of_arrays = 0;
    const int arrsize = ARRSIZE;
    int finish = 0;
    int push = 0;
    int flag = 0;
    int send = 0;
    int size = 0;
};

class Consumer
{
public:
    Consumer()
    {
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    }
    ~Consumer()
    {
        std::cout << "Consumer was destructed" << std::endl;
    }
    void FetchArrays()
    {
        while (true)
        {
            MPI_Send(&flag, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            MPI_Recv(&finish, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            if (finish == 1)
            {
                break;
            }
            std::array<int, ARRSIZE> queuefront;
            MPI_Recv(&queuefront, ARRSIZE, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            sum = 0;
            checksum = 0;
            std::sort(queuefront.begin(), queuefront.end());
            for (int i = 0; i < arrsize; i++)
            {
                sum += queuefront[i];
            }
            checksum = sum / arrsize;
            std::cout << "Check sum is: " << checksum << std::endl;
            tables_sorted++;
        }
        std::cout << "Consumer: " << rank <<" sorted: " << tables_sorted << std::endl;
    }

private:
```

```
    const int arrsize = ARRSIZE;
    int tables_sorted = 0;
    float checksum = 0;
    float sum = 0;
    int flag = 0;
    int finish = 0;
    int rank;
};

void mpi_producer_consumer(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    int size, rank;
    float start_time, end_time;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    start_time = MPI_Wtime();
    if (rank == 0)
    {
        Queue<std::array<int, ARRSIZE>> que(1000);
        Producer obj(5000,que);
        obj.MakeArrays();
        end_time = MPI_Wtime();
        std::cout << "Execution time: " << end_time - start_time << std::endl;;
    }
    if (rank > 0)
    {
        Consumer obj1;
        obj1.FetchArrays();
    }

    MPI_Finalize();
}

int main(int argc, char* argv[])
{
    mpi_producer_consumer(argc, argv);
    return  0;
}
```