**Лабораторная работа №4: Асинхронные операции в MPI на C++. Сравнение синхронного и асинхронного обмена данными**

**Теоретическая часть**

В MPI для асинхронного обмена данными существуют функции, которые позволяют начинать передачу и получать управление программой до завершения передачи данных. Они возвращают объект типа MPI\_Request, который используется с функцией MPI\_Wait для завершения операции. Эти функции помогают разгрузить процессоры, позволяя выполнять другие вычисления или готовиться к следующей операции обмена.

1. **MPI\_Iscatter** и **MPI\_Igather** — асинхронные версии функций для распределения и сбора данных. С помощью них можно инициировать операцию, но программа не будет ожидать её завершения, позволяя выполнять другие задачи, что сокращает общее время выполнения программы.
2. **MPI\_Wait** — эта функция завершает асинхронную операцию, блокируя выполнение, пока передача не завершится. Она необходима, если результаты операции требуются для дальнейших вычислений.

Асинхронные операции особенно полезны в случаях, когда можно совмещать передачу данных с вычислениями. В этом задании вы сравните синхронный и асинхронный обмен данными, чтобы оценить, как каждый из них влияет на производительность.

**Задание**

1. **Процесс с рангом 0** создаёт массив из N элементов и заполняет его значениями от 1 до N.
2. Все процессы используют синхронную функцию MPI\_Scatter для распределения данных. Затем каждый процесс умножает свои элементы на свой ранг.
3. Процессы собирают результат в массив с помощью MPI\_Gather.
4. Процесс 0 выводит результаты и время выполнения синхронного обмена.
5. Все процессы синхронизируются, затем начинают асинхронный обмен с использованием MPI\_Iscatter и MPI\_Igather. По завершении операции процесс 0 также выводит массив и время выполнения.
6. Сравните время выполнения синхронного и асинхронного обмена данными.

**Пример вывода:**

Synchronous data exchange took: 0.0008938 sec.

Resulting array of data (synchronous): 0 0 0 0 5 6 7 8 18 20 22 24 39 42 45 48

Asynchronous data exchange took: 0.000143 sec.

Resulting array of data (asynchronous): 0 0 0 0 5 6 7 8 18 20 22 24 39 42 45 48