Также данный класс располагает методом *Invoke*, который позволяет выполнять один или несколько методов, указываемых в виде его аргументов. Он также масштабирует исполнение кода, используя доступные процессоры, если имеется такая возможность. Данный метод сначала инициирует выполнение, а затем ожидает завершения всех передаваемых ему методов. Это, в частности, избавляет от необходимости вызывать метод *Wait*. И хотя это не гарантирует, что методы будут действительно выполняться параллельно, тем не менее, именно такое их выполнение предполагается, если система поддерживает несколько процессоров. Кроме того, в данном методе отсутствует возможность указать порядок выполнения методов от первого и до последнего, и этот порядок не может быть таким же, как и в списке аргументов.

1.4.3 Класс PLINQ и его методы

Также для параллельной обработки данных коллекций существует такой фреймворк, как PLINQ. Он позволяет работать в распределённом режиме с некоторой коллекцией данных (например, List).

По умолчанию все элементы коллекции в LINQ обрабатываются последовательно, но начиная с .NET 4.0 в пространство имен *System.Linq* был добавлен класс *ParallelEnumerable*, который инкапсулирует функциональность PLINQ (Parallel LINQ) и позволяет выполнять обращения к коллекции в параллельном режиме.

При обработке коллекции PLINQ использует возможности всех процессоров в системе. Источник данных разделяется на сегменты, и каждый сегмент обрабатывается в отдельном потоке. Это позволяет произвести запрос на многоядерных машинах намного быстрее.

В то же время по умолчанию PLINQ выбирает последовательную обработку данных. Переход к параллельной обработке осуществляется в том случае, если это приведет к ускорению работы. Однако, как правило, при параллельных операциях возрастают дополнительные издержки. Поэтому если параллельная обработка потенциально требует больших затрат ресурсов, то PLINK в этом случае может выбрать последовательную обработку, если она не требует больших затрат ресурсов.

Поэтому смысл применения PLINQ имеется преимущественно на больших коллекциях или при сложных операциях, где действительно выгода от распараллеливания запросов может перекрыть возникающие при этом издержки.

Также следует учитывать, что при доступе к общему разделяемому состоянию в параллельных операциях будет неявно использоваться синхронизация, чтобы избежать взаимоблокировки доступа к этим общим ресурсам. Затраты на синхронизацию ведут к снижению производительности, поэтому желательно избегать или ограничивать применения в параллельных операциях разделяемых ресурсов.

Параллельная обработка запросов выполняется фреймворком PLINQ в три этапа, как показано на рисунке 4. Сначала PLINQ решает, сколько потоков следует использовать для выполнения запроса. Затем рабочие потоки извлекают свои фрагменты из исходной коллекции, под защитой блокировок. Все потоки выполняют свои задания независимо и помещают результаты в свои локальные очереди. В заключение, локальные результаты объединяются в единую коллекцию, которая подается в цикл foreach [10].

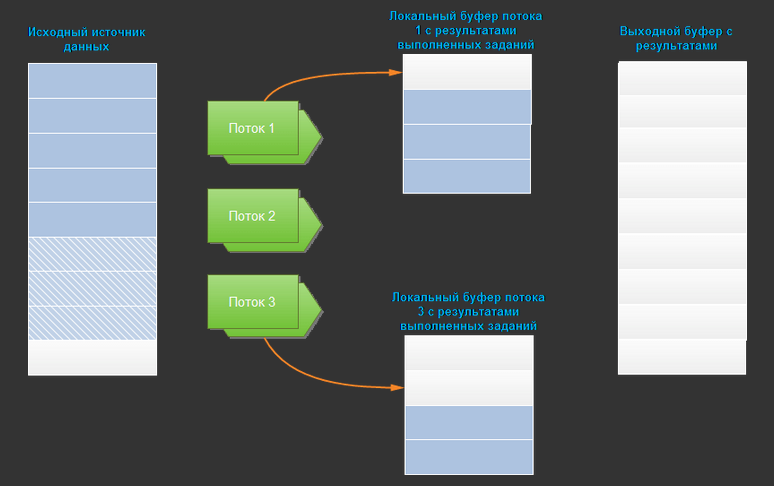


Рисунок 4 – Принцип работы фреймворка PLINQ

Метод *AsParallel* позволяет распараллелить запрос к источнику данных. Он реализован как метод расширения LINQ у массивов и коллекций. При вызове данного метода источник данных разделяется на части (если это возможно) и над каждой частью отдельно производятся операции.

Код также можно дополнительно оптимизировать с точки зрения параллелизации в случаях, когда для вывода результата параллельной операции используется цикл *foreach*. Его использование приводит к увеличению издержек ввиду того, что необходимо склеить полученные в разных потоках данные в один набор и затем их перебрать в цикле. Более оптимально в данном случае будет использовать метод *ForAll*, который выводит данные в том же потоке, в котором они обрабатываются.