Data Parallelism

Data parallelism refers to the application of some common operation over an aggregate of data either to produce a new data aggregate or to reduce the aggregate to a scalar value. The parallelism comes from doing the same logical operation to each element independent of the surrounding elements. There have been many languages with various levels of support for aggregate operations, but by far the most successful has been the one used with databases—SQL. LINQ provides direct support in both C# and Visual Basic for SQL-style operators, and the queries expressed with LINQ can be handed off to a data provider, such as ADO.NET, or can be evaluated against in-memory collections of objects or even XML documents.

Part of Parallel Extensions to .NET is an implementation of LINQ to Objects and LINQ to XML that includes parallel evaluation of the query. This implementation is called PLINQ and can be used to work conveniently with data aggregates. The following example illustrates the kernel of the standard K-means algorithm for statistical clustering: at each step you have K points in space that are your candidate cluster centers. Map every point to the nearest cluster and then, for all points mapped to the same cluster, recompute the center of that cluster by averaging the location of points in the cluster. This continues until a convergence condition is met where the positions of the cluster centers become stable. The central loop of this algorithm description is translated fairly directly into PLINQ, as shown in **Figure 4**.

Figure 4 Find Center

Copy Code

```
// C# using PLINQ
var q = from p in points.AsParallel()
        let center = nearestCenter(p, clusters)
                     // "index" of nearest cluster to p
        group p by center into g
        select new
        {
             index = g.Key,
             count = g.Count(),
             position = g.Aggregate(new Vector(0, 0, 0),
                    (accumulated, element) => accumulated + element,
                    (accumulated1, accumulated2) =>
                                  accumulated1 + accumulated2,
                    (accumulated) => accumulated
                    ) / g.Count()
        };
var newclusters = q.ToList();
```

The difference between LINQ and PLINQ is the AsParallel method on the data-collection points. This example also illustrates that LINQ includes the core map-reduce pattern but with clean integration into mainstream languages. One subtle point in this example is the behavior of the Aggregate operator. The third parameter is a delegate that provides a mechanism to combine partial sums. When this method is provided, the implementation is done in a parallel style by blocking the input into chunks, reducing each chunk in parallel, and then combining the partial results.

The great strength of data parallelism is that suitable algorithms are typically expressed much more cleanly and readably than if I had applied a structured multithreading approach where data structure assumptions can become entangled. Furthermore, the more abstract description allows the system greater opportunity to optimize in ways that would completely obscure the algorithm if done by hand. Finally, this high-level representation allows greater flexibility with the execution target: multicore CPU, GPU, or scale out to cluster. And as long as the leaf functions, for example, nearestCenter has no side effects—you don't face any of the concerns of data races or deadlocks of thread-oriented programming.