Introducción a la velocidad en PLINQ

PLINQ debe crear particiones del origen de datos y programar el trabajo en los subprocesos y generalmente tiene que combinar los resultados cuando finaliza la consulta. Todas estas operaciones aumentan el costo computacional de la paralelización; a estos costos derivados de agregar la paralelización se les denomina **sobrecarga**. Para lograr un rendimiento óptimo de una consulta PLINQ, el objetivo es maximizar las partes que son perfectamente paralelas y minimizar las partes que requieren sobrecarga.

Factores que afectan al rendimiento de las consultas PLINQ

En las siguientes secciones se enumeran algunos de los factores más importantes que afectan al rendimiento de las consultas paralelas. Estas son las instrucciones generales que por sí mismas no son suficientes para predecir el rendimiento de las consultas en todos los casos. Como siempre, es importante medir el rendimiento real de consultas específicas en equipos con una variedad de cargas y configuraciones representativas.

1. Costo computacional del trabajo total.

   Para conseguir velocidad, una consulta PLINQ debe tener suficiente trabajo perfectamente paralelo para compensar la sobrecarga. El trabajo se puede expresar como el costo computacional de cada delegado multiplicado por el número de elementos de la colección de origen. Suponiendo que una operación se puede paralelizar, mientras más costoso resulte desde el punto de vista computacional, mayor será la posibilidad de aumentar la velocidad. Por ejemplo, si una función tarda un milisegundo en ejecutarse, una consulta secuencial de más de 1000 elementos tardará un segundo en realizar esa operación, mientras que una consulta paralela en un equipo con cuatro núcleos puede tardar solo 250 milisegundos. Esto da como resultado una velocidad de 750 milisegundos. Si la función requiere un segundo para ejecutar cada elemento, la velocidad sería de 750 segundos. Si el delegado es muy caro, PLINQ podría proporcionar un aumento significativo de la velocidad con solo unos pocos elementos de la colección de origen. Por el contrario, las colecciones de origen pequeñas con delegados triviales generalmente no son buenas candidatas para PLINQ.

   En el ejemplo siguiente, queryA es probablemente una buena candidata para PLINQ, suponiendo que su función Select implica mucho trabajo. queryB probablemente no es una buena candidata porque no hay suficiente trabajo en la instrucción Select, y la sobrecarga de la paralelización compensará la mayoría o la totalidad de la velocidad.

```
Dim queryA = From num In numberList.AsParallel()
             Select ExpensiveFunction(num); 'good for PLINQ

Dim queryB = From num In numberList.AsParallel()
             Where num Mod 2 > 0
             Select num; 'not as good for PLINQ
```

```
var queryA = from num in numberList.AsParallel()
             select ExpensiveFunction(num); //good for PLINQ
```

```
var queryB = from num in numberList.AsParallel()
             where num % 2 > 0
             select num; //not as good for PLINQ
```

2. El número de núcleos lógicos del sistema (grado de paralelismo).

   Este punto es un corolario obvio de la sección anterior, las consultas que son perfectamente paralelas se ejecutan más rápido en equipos con varios núcleos porque se puede dividir el trabajo entre más subprocesos simultáneos. La cantidad total de velocidad depende de qué porcentaje del trabajo total de la consulta se puede paralelizar. Sin embargo, no se da por supuesto que todas las consultas se ejecutarán dos veces más rápido en un equipo de ocho núcleos que un equipo de cuatro núcleos. Al optimizar las consultas para un rendimiento óptimo, es importante medir los resultados reales en equipos con varios números de núcleos. Este punto se relaciona con el punto 1: los conjuntos de datos más grandes deben aprovechar la ventaja de una cantidad mayor de recursos informáticos.

3. El número y tipo de operaciones.

   PLINQ proporciona el operador AsOrdered para situaciones en las que es necesario mantener el orden de los elementos de la secuencia de origen. Hay un costo asociado con la ordenación, pero suele ser moderado. Las operaciones GroupBy y Join también incurren en sobrecarga. PLINQ funciona mejor si se permite procesar elementos en la colección de origen en cualquier orden y pasarlos al operador siguiente en cuanto estén listos. Para más información, consulte cómo conservar el orden en PLINQ.

4. La forma de ejecución de consultas.

   Si va a almacenar los resultados de una consulta llamando a ToArray o ToList, los resultados de todos los subprocesos paralelos deben combinarse en la estructura de datos única. Esto implica un costo computacional inevitable. Asimismo, si se recorren en iteración los resultados mediante el uso de un bucle foreach (For Each en Visual Basic), los resultados de los subprocesos de trabajo deben serializarse en el subproceso del enumerador. Pero si desea realizar alguna acción en función del resultado de cada subproceso, puede utilizar el método ForAll para realizar este trabajo en varios subprocesos.

5. El tipo de opciones de combinación.

   PLINQ puede configurarse para almacenar en buffer su salida y generarla en fragmentos o a la vez después de que el conjunto de resultados completo se genere, o bien para transmitir secuencias de los resultados individuales a medida que se generan. El primero da como resultado una reducción del tiempo de ejecución total y el último da como resultado una reducción de la latencia entre los elementos producidos. Aunque las opciones de combinación no siempre tienen una repercusión importante en el rendimiento general de las consultas, pueden afectar al rendimiento percibido porque controlan el tiempo que un usuario debe esperar para ver los resultados. Para más información, consulte las opciones de combinación en PLINQ.

6. El tipo de partición.

   En algunos casos, una consulta PLINQ sobre una colección de origen indexable puede producir una carga de trabajo desequilibrada. Cuando esto ocurre, es posible que pueda aumentar el rendimiento de

las consultas con la creación de un particionador personalizado. Para más información, consulte Particionadores personalizados para PLINQ y TPL.

Cuando PLINQ elige el modo secuencial

PLINQ siempre intentará ejecutar una consulta al menos tan rápido como se ejecutaría de forma secuencial. Aunque PLINQ no se fija en lo caros que son los delegados de usuario desde el punto de vista computacional o en lo grande que es el origen de entrada, sí busca determinadas "formas" de consulta. En concreto, busca operadores de consulta o combinaciones de operadores que normalmente provocan que una consulta se ejecute más lentamente en modo paralelo. Cuando encuentra esas formas, PLINQ vuelve al modo secuencial de forma predeterminada.

Sin embargo, después de medir el rendimiento de una consulta concreta, puede determinar que realmente se ejecute más rápido en modo paralelo. En tales casos puede usar la marca ParallelExecutionMode.ForceParallelism> con el método WithExecutionMode para indicar a PLINQ que paralelice la consulta. Para obtener más información, vea Cómo: Especificar el modo de ejecución en PLINQ.

En la lista siguiente se describen las formas de consulta PLINQ que de forma predeterminada se ejecutarán en modo secuencial:

- Las consultas que contienen una instrucción Select, Where indexada, SelectMany indexada o una cláusula ElementAt después de un operador de ordenación o filtrado que ha quitado o reorganizado los índices originales.
- Las consultas que contienen un operador Take, TakeWhile, Skip o SkipWhile y donde los índices de la secuencia de origen no están en el orden original.
- Las consultas que contienen Zip o SequenceEquals, a menos que uno de los orígenes de datos tenga un índice ordenado inicialmente y el otro origen de datos sea indexable, es decir, una matriz o IList(T).
- Las consultas que contienen Concat, a menos que se aplique a los orígenes de datos indexables.
- Las consultas que contienen Reverse, a menos que se aplique a un origen de datos indexable.