Problemas potenciales en el paralelismo de datos y tareas

En muchos casos, Parallel.For y Parallel.ForEach pueden proporcionar importantes mejoras de rendimiento con respecto a los bucles secuenciales normales. Sin embargo, el trabajo de paralelizar el bucle aporta una complejidad que puede conducir a problemas que, en código secuencial, no son tan comunes o no se producen en ningún caso. En este tema se indican algunas prácticas que se deben evitar al escribir bucles paralelos.

No suponer que la ejecución en paralelo siempre es más rápida

En ciertos casos, un bucle paralelo podría ejecutarse más lentamente que su equivalente secuencial. La regla básica es que es poco probable que los bucles en paralelo que tienen pocas iteraciones y delegados de usuario rápidos aumenten la velocidad en gran medida. Sin embargo, dado que hay muchos factores que afectan al rendimiento, recomendamos medir siempre los resultados reales.

Evitar la escritura en ubicaciones de memoria compartida

En código secuencial, no es raro leer o escribir en variables estáticas o campos de clase. Sin embargo, cada vez que varios subprocesos tienen acceso simultáneamente a estas variables, hay grandes posibilidades de que se produzcan condiciones de carrera. Aunque se pueden usar bloqueos para sincronizar el acceso a la variable, el costo de la sincronización puede afectar negativamente al rendimiento. Por tanto, se recomienda evitar, o al menos limitar, el acceso al estado compartido en un bucle en paralelo en la medida de lo posible. La mejor manera de hacerlo es utilizar las sobrecargas de Parallel.For y Parallel.ForEach que usan una variable System.Threading.ThreadLocal para almacenar el estado local de subproceso durante la ejecución del bucle. Para obtener más información, vea Cómo: Escribir un bucle Parallel.For con variables locales de subproceso y Cómo: Escribir un bucle Parallel.ForEach con variables locales de partición.

Evitar la paralelización excesiva

Si usa bucles en paralelo, incurrirá en costos de sobrecarga al crear particiones de la colección de origen y sincronizar los subprocesos de trabajo. El número de procesadores del equipo reduce también las ventajas de la paralelización. Si se ejecutan varios subprocesos enlazados a cálculos en un único procesador, no se gana en velocidad. Por tanto, debe tener cuidado para no paralelizar en exceso un bucle.

El escenario más común en el que se puede producir un exceso de paralelización son los bucles anidados. En la mayoría de los casos, es mejor paralelizar únicamente el bucle exterior, a menos que se cumplan una o varias de las siguientes condiciones:

- Se sabe que el bucle interno es muy largo.
- Se realiza un cálculo costoso en cada pedido (la operación que se muestra en el ejemplo no es costosa).
- Se sabe que el sistema de destino tiene suficientes procesadores como para controlar el número de subprocesos que se producirán al paralelizar la consulta de `cust.Orders`.

En todos los casos, la mejor manera de determinar la forma óptima de la consulta es mediante la prueba y la medición.

Evitar llamadas a métodos que no son seguros para subprocesos

La escritura en métodos de instancia que no son seguros para subprocesos de un bucle en paralelo puede producir daños en los datos, que pueden pasar o no inadvertidos para el programa. También puede dar lugar a excepciones. En el siguiente ejemplo, varios subprocesos estarían intentando llamar simultáneamente al método FileStream.WriteByte, lo que no se admite en la clase.

```
FileStream fs = File.OpenWrite(path);
byte[] bytes = new Byte[10000000];
// ...
Parallel.For(0, bytes.Length, (i) => fs.WriteByte(bytes[i]));
```

Limitar las llamadas a métodos seguros para subprocesos

La mayoría de los métodos estáticos de .NET Framework son seguros para subprocesos y se les puede llamar simultáneamente desde varios subprocesos. Sin embargo, incluso en estos casos, la sincronización que esto supone puede conducir a una ralentización importante en la consulta.

Tener en cuenta los problemas de afinidad de los subprocesos

Algunas tecnologías, como la interoperabilidad COM para componentes de contenedor uniproceso (STA), Windows Forms y Windows Presentation Foundation (WPF), imponen restricciones de afinidad de subprocesos que exigen que el código se ejecute en un subproceso determinado. En este caso es mejor usar Task o Thread para separarnos del hilo de la interfaz y después usamos Parallel para lo que contiene el otro hilo

Tener precaución cuando se espera en delegados a los que llama Parallel.Invoke

En determinadas circunstancias, la biblioteca TPL incluirá una tarea, lo que significa que se ejecuta en la tarea del subproceso que se está ejecutando actualmente. (Para más información, consulte Clase TaskScheduler). Esta optimización de rendimiento puede provocar un interbloqueo en ciertos casos. Por ejemplo, dos tareas podrían ejecutar el mismo código de delegado, que señala cuándo se genera un evento, y después esperar a que la otra tarea señale. Si la segunda tarea está alineada en el mismo subproceso que la primera y la primera entra en un estado de espera, la segunda tarea nunca podrá señalar su evento. Para evitar que esto suceda, puede especificar un tiempo de espera en la operación de espera o utilizar constructores de subproceso explícitos para ayudar a garantizar que una tarea no pueda bloquear a la otra.

No suponer que las iteraciones de ForEach, For y ForAll siempre se ejecutan en paralelo

Es importante tener en cuenta que las iteraciones individuales de un bucle Parallel.For, Parallel.ForEach o ForAll pueden ejecutarse en paralelo, pero no tiene que ser así necesariamente. Por consiguiente, se debe

evitar escribir código cuya exactitud dependa de la ejecución en paralelo de las iteraciones o de la ejecución de las iteraciones en algún orden concreto. Por ejemplo, es probable que este código lleve a un interbloqueo:

```
ManualResetEventSlim mre = new ManualResetEventSlim();
Enumerable.Range(0, Environment.ProcessorCount * 100)
.AsParallel()
.ForAll((j) => {
        if (j == Environment.ProcessorCount) {
            Console.WriteLine("Set on {0} with value of {1}",
                Thread.CurrentThread.ManagedThreadId, j);
            mre.Set();
        } else {
            Console.WriteLine("Waiting on {0} with value of {1}",
                Thread.CurrentThread.ManagedThreadId, j);
            mre.Wait();
        }
    }); //deadlocks
```

En este ejemplo, una iteración establece un evento y el resto de las iteraciones esperan el evento. Ninguna de las iteraciones que esperan puede completarse hasta que se haya completado la iteración del valor de evento. Sin embargo, es posible que las iteraciones que esperan bloqueen todos los subprocesos que se utilizan para ejecutar el bucle paralelo antes de que la iteración del valor de evento haya tenido oportunidad de ejecutarse. Esto produce un interbloqueo: la iteración del valor de evento nunca se ejecutará y las iteraciones que esperan nunca se activarán.

En concreto, una iteración de un bucle paralelo no debe esperar nunca otra iteración del bucle para progresar. Si el bucle paralelo decide programar las iteraciones secuencialmente pero en el orden contrario, se producirá un interbloqueo.

Evitar la ejecución de bucles en paralelo en el subproceso de la interfaz de usuario

Es importante que la interfaz de usuario (IU) de la aplicación siga respondiendo. Si una operación contiene bastante trabajo para garantizar la paralelización, no se debería ejecutar en el subproceso de la interfaz de usuario. Conviene descargarla para que se ejecute en un subproceso en segundo plano. Por ejemplo, si desea utilizar un bucle en paralelo para calcular datos que después se presentarán en un control de IU, considere ejecutar el bucle dentro de una instancia de la tarea en lugar de directamente en un controlador de eventos de IU. Solo si el cálculo básico se ha completado se debería serializar la actualización de nuevo en el subproceso de la interfaz de usuario.

Si ejecuta bucles en paralelo en el subproceso de la interfaz de usuario, tenga cuidado de evitar la actualización de los controles de la interfaz de usuario desde el interior del bucle. Si se intenta actualizar los controles de la interfaz de usuario desde dentro de un bucle en paralelo que se está ejecutando en el subproceso de la interfaz de usuario, se pueden provocar daños en el estado, excepciones, retrasos en las actualizaciones e incluso interbloqueos, dependiendo de cómo se invoque la actualización de la interfaz de usuario. En el siguiente ejemplo, el bucle en paralelo bloquea el subproceso de la interfaz de usuario en el que se está ejecutando hasta que todas las iteraciones se completan. Sin embargo, si se está ejecutando una iteración del bucle en un subproceso en segundo plano (como puede hacer Parallel.For), la llamada a Invoke

produce que se envíe un mensaje al subproceso de la interfaz de usuario, que se bloquea mientras espera a que ese mensaje se procese. Puesto que se bloquea el subproceso de la interfaz de usuario cuando se ejecuta Parallel.For, el mensaje puede no procesarse nunca y el subproceso de la interfaz de usuario se interbloquea.

```
private void button1_Click(object sender, EventArgs e) {
  Parallel.For(0, N, i => {
      // do work for i
      button1.Invoke((Action)delegate { DisplayProgress(i); });
  });
}
```

En el siguiente ejemplo se muestra cómo evitar el interbloqueo mediante la ejecución del bucle dentro de una instancia de la tarea. El bucle no bloquea el subproceso de la interfaz de usuario y se puede procesar el mensaje.

```
private void button1_Click(object sender, EventArgs e) {
    Task.Factory.StartNew(() =>
        Parallel.For(0, N, i => {
            // do work for i
            button1.Invoke((Action)delegate { DisplayProgress(i); });
        })
    );
}
```