



Parallel Programming con Visual Studio 2010

Marco Russo
<http://blogs.devleap.com/marco>
http://sqlblog.com/blogs/marco_russo

msdn Microsoft | TechNet

Agenda

- Introduzione alla scalabilità multi-core
- Task Parallel Library (TPL) in .NET 4
- Da LINQ a Parallel LINQ (PLINQ)
- Strumenti in Visual Studio 2010

msdn Microsoft | TechNet

Come ottenere scalabilità su più 8 o più core

INTRODUZIONE MULTI-CORE

msdn Microsoft | TechNet

Come guadagnare in prestazioni

- **Multithreading su elaborazioni**
 - Dove il costo di CPU è significativo
 - Per operazioni di I/O bastano operazioni asincrone
 - Per parallelizzare algoritmi lavoro più complesso
 - Bisogna anche ripartire il carico tra più processori possibili
- **Strade possibili**
 - Classe Thread
 - Begin/End Invoke
 - Task

Non solo dual core

- **Sfruttare CPU dual core è semplice**
 - Basta creare operazioni asincrone
 - Spesso è sufficiente usare le tecniche adottate per un solo core
 - Conflitti limitati
- **Ma con 4, 8 e più core è diverso**
 - HyperThreading richiede di sfruttare tutti i processori logici per sfruttare al massimo la CPU
- **Problemi**
 - Condivisione risorse
 - Definizione algoritmo
 - Overhead multithreading

Conflitto accesso memoria

- **Può penalizzare prestazioni, specialmente in scrittura**
 - Richiesti lock per accesso concorrente
- **Anche in lettura**
 - Conflitto accesso a bus memoria
 - Facile osservare 5% di guadagno solo per quello

Considerazioni

- Cambiare modo di programmare
 - Specialmente per algoritmi e operazioni CPU intensive
- Cambiare linguaggio?
 - Esistono linguaggi dedicati (es. Axum)
 - Programmazione più dichiarativa è un aiuto
 - Es. PLINQ

Task Parallel Library (TPL) in .NET 4

TPL IN .NET 4

msdn

Microsoft | TechNet

Agenda

- Classe Parallel
- Task
- Scheduler
- CancellationToken

Parallel Framework in .NET 4.0

- Libreria per semplificare parallelizzazione codice
 - Gestisce thread e sincronizzazione
- Non è una libreria di programmazione asincrona
 - Serve per velocizzare il software
 - Multithread non garantito – solo se migliora tempi di esecuzione
- Comprende:
 - TPL – Task Parallel Library
 - PLINQ – Parallel LINQ

Task Parallel Library (TPL)

- Assembly: incluso in mscorlib
- Namespace
 - System.Threading.Tasks
 - Parallel
 - Task
 - TaskManager
 - System
 - AggregateException

Parallel.Invoke method

- Array di delegate eseguiti senza vincoli di ordine
 - Array Action[]

```
Operation(1);
Operation(2);
Operation(3);
Operation(4);
```

```
Parallel.Invoke(
    () => Operation(1),
    () => Operation(2),
    () => Operation(3),
    () => Operation(4) );
```

Parallel.For

- Esegue un ciclo for in parallelo

```
for (int index = 0; index < 100; index++) {
    ProcessData(index);
}
```

```
Parallel.For(0, 100, (index) => {
    ProcessData(index);
});
```

Parallel.ForEach

- Esegue un ciclo foreach in parallelo

```
foreach (int value in data) {
    ProcessData(value);
}
```

```
Parallel.ForEach(data, (value) => {
    ProcessData(value);
});
```

Task

- I metodi di Parallel internamente usano istanze della classe Task
- Wrapper di un delegate che rappresenta una Unit of Work
- Metodi e proprietà di istanza:
 - Cancel, IsCompleted, IsCanceled
 - Esecuzione parte alla creazione dell'oggetto

Classe Task

- Usare metodi statici di Task
 - WaitAll
 - WaitAny
- Creazione via Task.Factory

```
Task t1 = Task.Factory.StartNew( delegate { Operation(1); });
Task t2 = Task.Factory.StartNew( delegate { Operation(2); });
Task t3 = Task.Factory.StartNew( () => { Operation(3); });
Task t4 = Task.Factory.StartNew( () => { Operation(4); });
Task.WaitAll( new Task[] { t1, t2, t3, t4 } );
```

Task<T>

- Deriva da Task
- Aggiunge semantica (.Result) per leggere il valore restituito da un'operazione sincrona, senza dover scrivere codice di sincronizzazione tra thread

```
var f1 = Task.Factory.StartNew ( () => Operation(1) );
var f2 = Task.Factory.StartNew ( () => Operation(2) );
var f3 = Task.Factory.StartNew ( () => Operation(3) );
var f4 = Task.Factory.StartNew ( () => Operation(4) );
int result = f1.Result + f2.Result + f3.Result + f4.Result;
```

Annullamento Task

- Pattern cooperativo
- CancellationTokenSource
 - Controlla e propaga cancellazione
 - Tipo reference (classe sealed)
 - Metodi Cancel, IsCancellationRequested
 - Proprietà Token – restituisce CancellationToken
- CancellationToken
 - Polling o notifica su cancellazione
 - Tipo value
 - Proprietà pubblica WaitHandle
 - Metodi IsCancellationRequested, Register

Classe TaskFactory

- Definisce default per:
 - CancellationToken
 - Scheduler
 - TaskCreationOptions
 - Default su creazione Task
 - TaskContinuationOptions
 - Default su chiamata a ContinueWith (extension method su Task definito in TaskFactory)

Distributed Task Queue

- Esiste una coda di task per ogni core
- Sistema più efficiente per gestire il parallelismo
 - Riduce o evita sincronizzazione
 - Importante per le prestazioni
- Migliori risultati con codice che non necessita di sincronizzazione tra thread diversi

TaskScheduler

- Classe astratta – consente specializzazioni
- Disponibile supporto a sincronizzazione con UI

```
public void Button1_Click(...) {
    var ui = TaskScheduler.FromCurrentSynchronizationContext();

    Task.Factory.StartNew(() => {
        return LoadAndProcessImage(); // compute the image
    }).ContinueWith(t => {
        pictureBox1.Image = t.Result; // display it
    }, ui);
}
```

Problemi di Concorrenza

- Operazioni parallele su dati condivisi
 - Sincronizzazione limita scalabilità e prestazioni
 - Evitare dati condivisi

```
int sum = 0;
Parallel.For(0, 10, (index) => {
    int local = sum;
    local += Operation(index);
    sum = local;
});

object sync = new object();
int sum = 0;
Parallel.For(0, 10, (index) => {
    lock (sync) {
        int local = sum;
        local += Operation(index);
        sum = local;
    }
});

int[] results = new int[10];
Parallel.For(0, 10, (index) => {
    results[index] = Operation(index);
});
int sum = results.Sum();
```

Problemi di Concorrenza

- Accesso a counter loop in lambda expression assegnate a Task
 - Copiare dati su variabile locale

```
for (int i = 0; i < array.length; i++) {
    Task.Factory.StartNew(
        () => Calc( array[i] ) );
}

for (int i = 0; i < array.length; i++) {
    var item = array[i];
    Task.Factory.StartNew(
        () => Calc( item ) );
}
```

Gestione eccezioni

- Un'eccezione non ferma subito altri thread
- Le eccezioni vengono catturate e aggregate
- Classe **AggregateException**
 - Collection InnerExceptions con elenco eccezioni

```
try {
    Parallel.Do(
        () => { throw new ArgumentException(); },
        () => { throw new NullReferenceException(); }
    );
} catch (AggregateException ex) {
    Console.WriteLine(ex.InnerExceptions.Count);
}
```

Parallel LINQ (PLINQ)

- Implementazione di LINQ to Object
- Parallellizza esecuzione query
 - Si applicano extension methods di PLINQ usando AsParallel su una collection esistente
- Consente di controllare
 - Grado di parallelismo
 - Vincoli di ordinamento sull'esecuzione

AsParallel

- Consente di parallelizzare query esistenti
 - Query che usano LINQ to Objects

```
int[] numbers = ...
var oddNumbers =
    from i in numbers.AsParallel()
    where i % 2 == 1
    select i;
```

Processing mode

- Modalità di esecuzione (**ParallelMergeOptions**)
 - Pipelined
 - Default, un pool di thread esegue le operazioni e un thread separato riceve i risultati
 - **WithMergeOptions(ParallelMergeOptions.NotBuffered)**
 - **WithMergeOptions(ParallelMergeOptions.AutoBuffered)**
 - Stop-and-Go
 - I dati vengono consumati dopo che tutti i worker thread sono finiti
 - Migliori tempi complessivi, maggiore latenza iniziale
 - **WithMergeOptions(ParallelMergeOptions.FullyBuffered)**
 - Inverted Enumeration
 - Risultati consumati nello stesso thread che li produce
 - Usare metodo **ForAll** su query

www.devleap.it

Effetti collaterali

- Ordine dei risultati
 - Non viene mantenuto per default
 - **orderby** nella query forza l'ordinamento del risultato
 - Uso implicito di stop-and-go
 - Usare **AsParallel().AsOrdered()**
 - Mantiene ordinamento iniziale
 - Impatto prestazionale, conviene solo in alcuni casi
- Race condition (modifica durante lettura)
 - Usare funzioni "pure", senza effetti collaterali

```
int accumulator = 0;
var query =
    from i in data.AsParallel(ParallelQueryOptions.PreserveOrdering)
    select new { Aggregated = accumulator += i, Value = i };
```

Restituzione dati

- Classi .NET non sono thread-safe per default
- Attenzione a usare classi condivise nell'elaborazione dei risultati di una query PLINQ

```
List<int> result = new
List<int>();
var query =
    from i in data.AsParallel()
    where i % 2 == 0
    select i;
foreach (var number in query) {
    result.Add(number);
}
```

```
List<int> result = new
List<int>();
var query =
    from i in data.AsParallel()
    where i % 2 == 0
    select i;
query.ForAll((number) => {
    result.Add(number);
});
```

```
var query =
    from i in data.AsParallel()
    where i % 2 == 0
    select i;
List<int> result = query.ToList();
```

Collection thread-safe

- Classi collection thread-safe in namespace **System.Collections.Concurrent**
 - **BlockingCollection<T>**
 - **ConcurrentBag<T>**
 - **ConcurrentDictionary<TKey,TValue>**
 - **ConcurrentQueue<T>**
 - **ConcurrentStack<T>**

In sintesi

- Parallelismo applicazioni è una tappa obbligata
- Unica strada per migliorare prestazioni con macchine multicore
- Usare i Task
- Pattern annullamento – CancellationSourceToken
- Gestione errori
- Usare System.Collections.Concurrent

Strumenti in Visual Studio 2010 per il debug e l'ottimizzazione di applicazioni multithreading e multicore

TOOL IN VS2010

msdn

Microsoft | TechNet

Agenda

- Performance Analysis in Visual Studio 2010
 - Timing / CPU
 - Memoria
 - Concorrenza
- In questa sessione vediamo solo concorrenza

Concorrenza

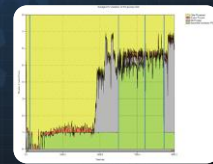
- Modalità acquisizione
 - Resource Contention
 - Opzione «Collect resource contention data»
 - Informazioni su contese nell'accesso a una risorsa
 - Thread Execution
 - Opzione «Visualize the behavior of a multithread application»
 - Abilita uso di Concurrency Visualizer
 - Richiede diritti amministratore

Concurrency Visualizer

- Interfaccia visualizzazione delle informazioni acquisite su
 - Uso complessivo CPU su Logical Core
 - Thread
 - Allocazione lavoro sui singoli Core

CPU Utilization

- Area verde: numero di Core in uso
- Area grigia: Idle – Core non impegnati
- Area rossa: System – es. kernel, servizi
- Area gialla: altri processi (o idle su s.o.)



Threads View

- Attività per Thread
- Attività su dischi
- Informazioni sul codice sottostante un blocco-colore, in particolare:
 - Synchronization
 - I/O
 - Esecuzione



Cores View

- Attività dei thread sui singoli core
- Spazi bianchi: non c'è attività
- Cambi colore frequenti: alto context-switch
 - Operazione costosa
- Colore uniforme e costante = esecuzione più efficiente



Microsoft
Your potential. Our passion.™

© 2010 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names, logos or icons are either registered trademarks or trademarks in the U.S. and/or other countries. The information in this document is for informational purposes and does not constitute a contract. Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS PRESENTATION.

msdn

Microsoft | TechNet