



Parallel LinQ (P LinQ)

. AsParallel

TH: Ko dùng AsParallel sẽ là 1 thread

```

Program Main()
References
static void Main()
{
    // Lấy thông tin về tiến trình hiện tại
    Process currentProcess = Process.GetCurrentProcess();

    // Lấy số lượng thread trước khi sử dụng Parallel
    int threadCountBefore = currentProcess.Threads.Count;

    // Tạo một mảng số nguyên
    int[] numbers = Enumerable.Range(0, 10).ToArray();

    // Sử dụng vòng lặp thông thường để xử lý các phần tử của mảng numbers
    foreach (var number in numbers)
    {
        // Có thể thay đổi logic xử lý ở đây
        Console.WriteLine($"Đang xử lý số {number} trong thread {System.Threading.Thread.CurrentThread.ManagedThreadId}");
    }

    // Lấy số lượng thread sau khi sử dụng Parallel
    int threadCountAfter = currentProcess.Threads.Count;

    // In ra số lượng thread trước và sau khi sử dụng Parallel
    Console.WriteLine($"Số lượng thread trước khi sử dụng Parallel: {threadCountBefore}");
    Console.WriteLine($"Số lượng thread sau khi sử dụng Parallel: {threadCountAfter}");

    Console.ReadLine(); // Dừng chương trình để bạn có thể xem kết quả trước khi đóng cửa sổ console
}

```

```

C:\Users\hoang\source\repos\Test Parallel\Test ...
Đang xử lý số 0 trong thread 1
Đang xử lý số 1 trong thread 1
Đang xử lý số 2 trong thread 1
Đang xử lý số 3 trong thread 1
Đang xử lý số 4 trong thread 1
Đang xử lý số 5 trong thread 1
Đang xử lý số 6 trong thread 1
Đang xử lý số 7 trong thread 1
Đang xử lý số 8 trong thread 1
Đang xử lý số 9 trong thread 1
Số lượng thread trước khi sử dụng Parallel: 17
Số lượng thread sau khi sử dụng Parallel: 17

```

TH2 : Trường hợp có AsParallel

```
0 references
static void Main()
{
    // Lấy thông tin về tiến trình hiện tại
    Process currentProcess = Process.GetCurrentProcess();

    // Lấy số lượng thread trước khi sử dụng Parallel
    int threadCountBefore = currentProcess.Threads.Count;

    // Tạo một mảng số nguyên
    int[] numbers = Enumerable.Range(0, 10).ToArray();

    // Sử dụng vòng lặp thông thường để xử lý các phần tử của mảng number
    numbers.AsParallel().ForAll(number =>
    {
        // Có thể thay đổi logic xử lý ở đây
        Console.WriteLine($"Đang xử lý số {number} trong thread {System.Threading.Thread.CurrentThread.ManagedThreadId}");
    });

    // Lấy số lượng thread sau khi sử dụng Parallel
    int threadCountAfter = currentProcess.Threads.Count;

    // In ra số lượng thread trước và sau khi sử dụng Parallel
    Console.WriteLine($"Số lượng thread trước khi sử dụng Parallel: {threadCountBefore}");
    Console.WriteLine($"Số lượng thread sau khi sử dụng Parallel: {threadCountAfter}");

    Console.ReadLine(); // Dừng chương trình để bạn có thể xem kết quả trước khi đóng cửa sổ console
}
```

```
C:\Users\hoang\source\repos\Test Parallel\Test ...
Đang xử lý số 1 trong thread 6
Đang xử lý số 6 trong thread 13
Đang xử lý số 0 trong thread 4
Đang xử lý số 2 trong thread 9
Đang xử lý số 4 trong thread 10
Đang xử lý số 3 trong thread 11
Đang xử lý số 5 trong thread 12
Đang xử lý số 7 trong thread 16
Đang xử lý số 9 trong thread 15
Đang xử lý số 8 trong thread 14
Số lượng thread trước khi sử dụng Parallel: 16
Số lượng thread sau khi sử dụng Parallel: 16
```

. WithDegreeOfParallelism

Lưu ý : ở đây nó đã tập trung vào **2 thread** như ta đã yêu cầu

```
0 references
static void Main()
{
    // Lấy thông tin về tiến trình hiện tại
    Process currentProcess = Process.GetCurrentProcess();

    // Lấy số lượng thread trước khi sử dụng Parallel
    int threadCountBefore = currentProcess.Threads.Count;

    // Tạo một mảng số nguyên
    int[] numbers = Enumerable.Range(0, 10).ToArray();

    // Sử dụng vòng lặp thông thường để xử lý các phần tử của mảng numbers
    numbers.AsParallel().WithDegreeOfParallelism(2).ForAll(number =>
    {
        // Có thể thay đổi logic xử lý ở đây
        Console.WriteLine($"Đang xử lý số {number} trong thread {System.Threading.Thread.CurrentThread.ManagedThreadId}");
    });

    // Lấy số lượng thread sau khi sử dụng Parallel
    int threadCountAfter = currentProcess.Threads.Count;

    // In ra số lượng thread trước và sau khi sử dụng Parallel
    Console.WriteLine($"Số lượng thread trước khi sử dụng Parallel: {threadCountBefore}");
    Console.WriteLine($"Số lượng thread sau khi sử dụng Parallel: {threadCountAfter}");

    Console.ReadLine(); // Dừng chương trình để bạn có thể xem kết quả trước khi đóng cửa sổ console
}
```

```
C:\Users\hoang\source\repos\Test Parallel\Test ...
Đang xử lý số 0 trong thread 6
Đang xử lý số 5 trong thread 1
Đang xử lý số 6 trong thread 1
Đang xử lý số 7 trong thread 1
Đang xử lý số 8 trong thread 1
Đang xử lý số 9 trong thread 1
Đang xử lý số 1 trong thread 6
Đang xử lý số 2 trong thread 6
Đang xử lý số 3 trong thread 6
Đang xử lý số 4 trong thread 6
Số lượng thread trước khi sử dụng Parallel: 16
Số lượng thread sau khi sử dụng Parallel: 16
```

AsOrdered And AsUnordered

Khác biệt AsOrdered va AsUnordered:

- AsOrdered khi trả về kết quả sẽ được **sắp xếp thứ tự y chang kết quả đầu vào**
- AsUnordered khi trả về kết quả sẽ **ko theo bất kỳ quy tắc nào cả** và điều đó giúp **tăng tốc câu lệnh**

The screenshot shows a Visual Studio IDE with a C# file named Program.cs. The code defines a class Program with a static Main method. Inside Main, a string array 'strings' is defined with values: "apple", "banana", "grape", "orange", "kiwi", "pear", "strawberry". The code then demonstrates two parallel queries. The first query, 'query1', uses 'AsParallel().AsUnordered().Where(str => str.Length > 0);'. It iterates over 'query1' and prints each string. The second query, 'query2', uses 'AsParallel().AsOrdered().Where(str => str.Length > 0);'. It iterates over 'query2' and prints each string. The output in the 'Microsoft Visual Studio Debug Console' shows the results of these queries. For 'Using AsUnordered():', the strings are printed in the order: banana, grape, orange, kiwi, pear, strawberry, apple. For 'Using AsOrdered():', the strings are printed in the order: apple, banana, grape, orange, kiwi, pear, strawberry. The console also shows the file path and some debugging instructions.

```

using System;
using System.Linq;

class Program
{
    static void Main()
    {
        string[] strings = { "apple", "banana", "grape", "orange", "kiwi", "pear", "strawberry" };

        Console.WriteLine("Using AsUnordered():");
        var query1 = strings
            .AsParallel().AsUnordered()
            .Where(str => str.Length > 0);

        foreach (var str in query1)
        {
            Console.WriteLine(str);
        }

        Console.WriteLine("\nUsing AsOrdered():");
        var query2 = strings
            .AsParallel().AsOrdered()
            .Where(str => str.Length > 0);

        foreach (var str in query2)
        {
            Console.WriteLine(str);
        }
    }
}

```

Microsoft Visual Studio Debug Console

```

Using AsUnordered():
banana
grape
orange
kiwi
pear
strawberry
apple

Using AsOrdered():
apple
banana
grape
orange
kiwi
pear
strawberry

C:\Users\hoang\source\repos\Test Parallel\Test Parallel\bin\Debug\net6.0\Test Parallel.exe
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Close console when debugging stops.
Press any key to close this window . . .

```

. ParallelEnumerable.Range vs ParallelEnumerable.Repeat vs ParallelEnumerable.Empty

```
#region[ Khác biệt ( ParallelEnumerable ) Range - Repeat - Empty : ]
0 references
static void Main()
{
    // Tạo một mảng số từ 1 đến 10
    int[] numbers = Enumerable.Range(1, 10).ToArray();

    // Sử dụng ParallelEnumerable.Range để thực hiện các phép tác song song trên mảng numbers
    Console.WriteLine("Using ParallelEnumerable.Range:");
    var range = ParallelEnumerable.Range(0, numbers.Length);
    foreach (var index in range)
    {
        Console.WriteLine($"Element at index {index}: {numbers[index]}");
    }

    // Sử dụng ParallelEnumerable.Repeat để lặp lại một giá trị từ mảng numbers
    Console.WriteLine("\nUsing ParallelEnumerable.Repeat:");
    var repeat = ParallelEnumerable.Repeat(numbers[0], 3);
    foreach (var num in repeat)
    {
        Console.WriteLine(num);
    }

    // Sử dụng ParallelEnumerable.Empty để tạo ra một tập hợp rỗng
    Console.WriteLine("\nUsing ParallelEnumerable.Empty:");
    var empty = ParallelEnumerable.Empty<int>();
    Console.WriteLine($"Count of empty set: {empty.Count()}"); // In ra 0 vì tập hợp rỗng
}
#endregion
```

Test Parallel

- Dependencies
- Program.cs

Microsoft Visual Studio Debug Console

Using ParallelEnumerable.Range:

Element at index 0: 1
Element at index 1: 2
Element at index 2: 3
Element at index 3: 4
Element at index 4: 5
Element at index 5: 6
Element at index 6: 7
Element at index 7: 8
Element at index 8: 9
Element at index 9: 10

Using ParallelEnumerable.Repeat:

1
1
1

Using ParallelEnumerable.Empty:
Count of empty set: 0

C:\Users\hoang\source\repos\Test Parallel\ode 0.
To automatically close the console
le when debugging stops.
Press any key to close this window

. AsSequential vs WithCancellation vs WithMergeOptions

- AsSequential ở đây là Chuyển đổi PLINQ (Parallel LINQ) sang LINQ tuần tự (trái ngược với (PLINQ)) Có thể kết hợp với AsParallel
- WithCancellation ở đây là sẽ cho phép quăng ra lỗi trước khi xuất ra đáp án
- WithMergeOptions

```
// Sử dụng AsSequential() để thực hiện các phép tác tuần tự trên mảng numbers
Console.WriteLine("Using AsSequential()");
var sequentialQuery = numbers.AsParallel().AsSequential().Where(num => num % 2 == 0);
foreach (var num in sequentialQuery)
{
    Console.WriteLine(num);
}

// Sử dụng WithCancellation() để hủy bỏ thực hiện các phép tác song song
Console.WriteLine("\nUsing WithCancellation()");
CancellationTokenSource cancellationTokenSource = new CancellationTokenSource();
var cancellationQuery = numbers.AsParallel().WithCancellation(cancellationTokenSource.Token).Where(num => num % 2 == 0);
try
{
    foreach (var num in cancellationQuery)
    {
        Console.WriteLine(num);
        if (num == 4) // Giả định một điều kiện để hủy bỏ
        {
            cancellationTokenSource.Cancel();
            break;
        }
    }
}
catch (OperationCanceledException)
{
    Console.WriteLine("Query was cancelled.");
}

// Sử dụng WithMergeOptions() để cấu hình cách hợp nhất kết quả
Console.WriteLine("\nUsing WithMergeOptions()");
var mergeQuery = numbers.AsParallel().WithMergeOptions(ParallelMergeOptions.NotBuffered).Where(num => num % 2 == 0);
foreach (var num in mergeQuery)
{
    Console.WriteLine(num);
}
```

Microsoft Visual Studio Debug Console

```
Using AsSequential():
2
4
6
8
10

Using WithCancellation():
2
4

Using WithMergeOptions():
8
2
4
6
10

C:\Users\hoang\source\repos\Test Parallel\Test Para
```

. Partitioning in PLINQ

29:35

. WithExecutionMode

31:04

. ForAll

Query Splitting

```
List<Department> departments =
    context.Departments
        .Include(d => d.Teams)
        .ThenInclude(t => t.Employees)
        .ThenInclude(e => e.Tasks)
        .ToList();
```

Bulk Updates and Deletes

```

var salesEmployees = context.Employees
    .Where(e => e.Department == "Sales")
    .ToList();

foreach (var employee in salesEmployees)
{
    employee.Salary *= 1.05m;
}

context.SaveChanges();

```

thành thế này

```

context.Employees
    .Where(e => e.Department == "Sales")
    .ExecuteUpdate(s => s.SetProperty(e => e.Salary, e =>
e.Salary * 1.05m));

```

```

context.Carts
    .Where(o => o.CreatedOn < DateTime.Now.AddYears(-1))
    .ExecuteDelete();

```

Raw SQL Queries

```

public class ProductSummary
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public decimal TotalSales { get; set; }
}

var productSummaries = await context.Database
    .SqlQuery<ProductSummary>(
        @$"""
        SELECT p.ProductId, p.ProductName, SUM(oi.Quantity
* oi.UnitPrice) AS TotalSales
        FROM Products p

```

```
        JOIN OrderItems oi ON p.ProductId = oi.ProductId
        WHERE p.CategoryId = {categoryId}
        GROUP BY p.ProductId, p.ProductName
        """)
    .ToListAsync();
```

Query Filters

Eager Loading