در این مقاله قصد داریم مشکلاتی که ممکن است LINQ داشته باشد را باهم بررسی کنیم. که با مثال برای شما توضیح خواهیم داد .

مشکل با LINQ چیست؟

LINQدر زمینه hidden allocations عملکرد ضعیفی داشته است.

از یک تکنولوژی ممکن است به راحتی کد ناکار آمد بنویسید. به LINQ-to-Objects نگاه کنید. به وسیله این کد چه مقدار کد ناکارآمد معرفی شده است.؟

```
int[] Scale(int[] inputs, int lo, int hi, int c) {
  var results = from x in inputs
            where (x >= lo) && (x <= hi)
            select (x * c);
  return results.ToArray();
}
```

برای فهمیدن کامل مشکل ، ما نیاز داریم بدانیم که کامپایلر چه کاری برای ما انجام میدهد.

کدبالا چیزی شبیه کد زیر است.

```
private int[] Scale(int[] inputs, int lo, int hi, int c)
{
   <>c__DisplayClass0_0 CS<>8__locals0;
   CS<>8__locals0 = new <>c__DisplayClass0_0();
   CS<>8__locals0.lo = lo;
   CS<>8__locals0.hi = hi;
   CS<>8__locals0.c = c;
   return inputs
      .Where<int>(new Func<int, bool>(CS<>8__locals0.<Scale>b__0))
      .Select<int, int>(new Func<int, int>(CS<>8__locals0.<Scale>b__1))
      .ToArray<int>();
}

[CompilerGenerated]
private sealed class c__DisplayClass0_0
{
   public int c;
   public int hi;
   public int lo;

   internal bool <Scale>b__0(int x)
   {
      return ((x >= this.lo) && (x <= this.hi));
   }

   internal int <Scale>b__1(int x)
   {
      return (x * this.c);
   }
}
```

همانطور که شما میبینید، یک کلاس اضافی و چند متد برای انجام کارهای منطقی تعریف کرده ایم.

اما این در حقیقت برای نمایش دادن آن ها در Visual Studio **به طور مستقیم** hidden **اختصاص داده شده است.**

اگر شما پلاگین Heap Allocation Viewer **را به خوبی برای** Resharper **نصب کرده باشید** tool-tip **های زیر را در** IDE**خواهید دید:**

```
int[] Scale(int[] inputs, int lo, int hi, int c)
{
    var results = from x in inputs
                  where (x >= lo) && (x <= hi)
                  sel
    return results.To
}
```

Delegate allocation: capture of 'lo', 'hi' parameters

قبل از نگاه کردن به روش های دیگر شما میتوانید تاثیر LINQ را کاهش دهید.

برای مثال الگوی داشتن یک ()where با پیروی از ()select بهینه شده است بنابراین فقط یک تکرار استفاده میشود نه دو تکراری که تصور میکردید. به همین شکل دو عبارت ()select در یک ردیف با هم میکس میشوند بنابراین به یک تکرار نیاز است .

یک نکته روی) Micro-optimizations بهینه سازی ریز: (

شما باید همیشه اول profile و سپس بعد از آن benchmark را بزنید.

اگر شما این کار را انجام دهید در بقیه روش ها یک وسوسه ای برای انجام بهینه سازی کار به وجود میاید.

با همه این تفاسیر کامپایلر #C به مواردی حساس است پس بهتر است کار های زیر را انجام دهیم:

اجتناب از تخصیص hot paths در کامپایلر:

اجتناب ازLINQ

اجتناب از استفاده foreach در طول مقدار دهی وقتی که ساختمان enum نداریم.

استفاده ازobject pool

این عقیده ،حاصل تفکر آدم هایی است که برای اولین بار linq **را طراحی کردند.اما این روش هزینه های چشمگیری دارد.**

RoslynLinqRewriteو بهینه ساز: LINQ

ما میتوانیم به صورت دستی هر عبارت LINQ **را در هر ورژنی بازنویسی کنیم اگر ما نگران عملکرد بودیم اما این جالب نیست که آن**

یک ابزاری بودکه کارهای سخت ما را انجام میداد؟ ما این را داریم

اولین آنها RoslynLinqRewrite **است که در هر صفحه پروژه است.**

این ابزار کد های C# **را توسط اولین بازنویسی درخت گرامر** LINQ **با استفاده از کد** procedural کامپایل میکند.

همچنین در Nessos LinqOptimizer**نیر موجود است که:**

یک query بهینه ساز کامپایلر برای LINQ ترتیبی و موازی است .

LinqOptimizerکوئری های اخباری LINQ را به سرعت به کد اجباری (imperative) کامپایل میکند.

کد های کامپایل شده کم تر به صورت بالقوه فراخوانی میشوند و به صورت heap**تخصیص داده میشوند.**

در سطح بالا ؛ فرق اصلی بین آنها در زیر آمده است:

• RoslynLinqRewrite

-در زمان کامپایل کار میکند(اما از کامپایل افزایشی پروژه شما جلوگیری میکند)

-کد ها تغییر نمیکنند؛ مگر اینکه شما بخواهید آن را با [NoLinqRewrite] **بهینه کنید.**

•LinqOptimiser

-در زمان اجرا (run-time) **کار میکند.**

-شما را به استفاده از AsQueryExpr().Run() برای متد های LINQ مجبور میکند.

LINQ -موازی را بهینه میکند.

در rest یک پست به ابزار در جزییات و آنالیز عملکرد آنها نگاه میکنیم .

مقایسه پشتیبانی: LINQ

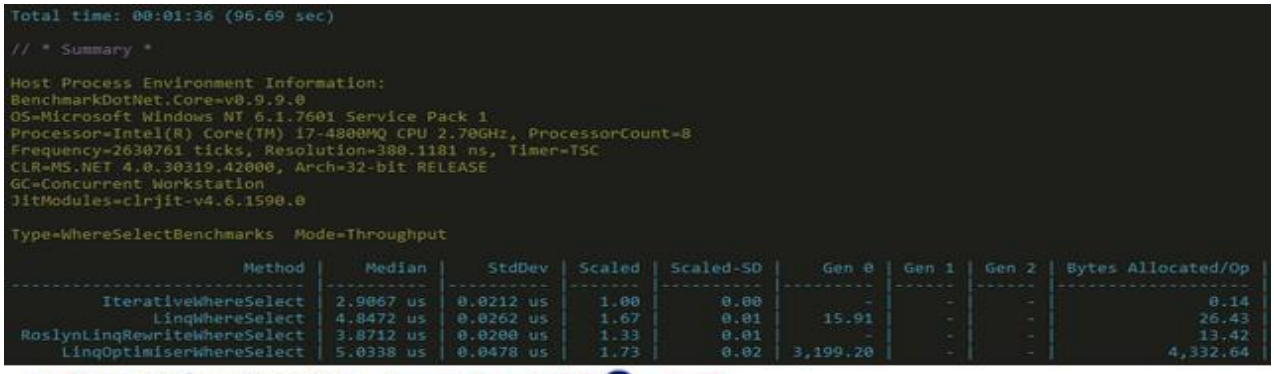
قبل از انتخاب ابزار؛ شما میخواهید مطمئن شوید که به طور واقعی عبارت های LINQ شما در کد بهینه میشود. هرچند ابزار ها کل محدوده موجود LINQ Query Expressions را همانطور که در چارت میبینید پشتیبانی میکنند :

متد	RoslynLinqدوباره نویسی	LINQ بهینه سازی	هر دو ؟
Select	✓	✓	بله
Where	✓	✓	بله
Reverse	✓	X	
Cast	✓	X	
OfType	✓	X	
First/FirstOrDefault	✓	X	
Single/SingleOrDefault	✓	X	
Last/LastOrDefault	✓	X	
ToList	✓	✓	بله
ToArray	✓	✓	بله
ToDictionary	✓	X	
Count	✓	✓	بله
LongCount	✓	X	
Any	✓	X	
All	✓	X	
ElementAt/ElementAtOrDefault	✓	X	
Contains	✓	X	
ForEach	✓	✓	بله
Aggregate	X	✓	
Sum	X	✓	
SelectMany	X	✓	
Take/TakeWhile	X	✓	
Skip/SkipWhile	X	✓	
GroupBy	X	✓	
OrderBy/OrderByDescending	X	✓	
ThenBy/ThenByDescending	X	✓	
Total	22	18	6

با یک سناریو رایج با استفاده از linq برای فیلتر و map کردن اعدا ترتیبی در C# شروع میکنیم:

```
var results = items.Where(i => i % 10 == 0)
                   .Select(i => i + 5);
```

کد LINQ بالا را با دو بهینه ساز مقایسه میکنیم . نتیجه را در زیر میبینیم:



```
Total time: 00:01:36 (96.69 sec)

// * Summary *

Host Process Environment Information:
BenchmarkDotNet.Core=v0.9.9.0
OS=Microsoft Windows NT 6.1.7601 Service Pack 1
Processor=Intel(R) Core(TM) i7-4800MQ CPU 2.70GHz, ProcessorCount=8
Frequency=2630761 ticks, Resolution=380.1181 ns, Timer=TSC
CLR=MS.NET 4.0.30319.42000, Arch=32-bit RELEASE
GC=Concurrent Workstation
JitModules=clrjit-v4.6.1590.0

Type=WhereSelectBenchmarks  Mode=Throughput
```

Method	Median	StdDev	Scaled	Scaled-SD	Gen 0	Gen 1	Gen 2	Bytes Allocated/Op
IterativeWhereSelect	2.9067 us	0.0212 us	1.00	0.00	-	-	-	0.14
LinqWhereSelect	4.8472 us	0.0262 us	1.67	0.01	15.91	-	-	26.43
RoslynLinqRewriteWhereSelect	3.8712 us	0.0200 us	1.33	0.01	-	-	-	13.42
LinqOptimiserWhereSelect	5.0338 us	0.0478 us	1.73	0.02	3,199.20	-	-	4,332.64

در اولین نگاه؛ نسخه LinqOptimiser در مقایسه با دیگری حافظه زیادی را گرفته است.

برای اینکه ببینیم چرا این اتفاق افتاده است نیاز داریم به کد در generator نگاهی بیاندازیم.

که چیزی شبیه زیر است:

```
IEnumerable<int> LinqOptimizer(int [] input)
{
   var collector = new Nessos.LinqOptimizer.Core.ArrayCollector<int>();
   for (int counter = 0; counter < input.Length; counter++)
   {
      var i = input[counter];
      if (i % 10 == 0)
      {
         var result = i + 5;
         collector.Add(result);
      }
   }
```

```
    return collector;
}
```

این موضوع به صورت پیش فرض ؛ ArrayCollectorیک int[1024]] را به عنوان ذخیره سازی پشتوانه آن اختصاص میدهد از این رو تخصیص بیش از حد میشود..

در مقابل RoslynLinqRewrite کد زیر را بهینه میکند:

```
IEnumerable<int> RoslynLinqRewriteWhereSelect_ProceduralLinq1(int[] _linqitems)
{
    if (_linqitems == null)
        throw new System.ArgumentNullException();
    for (int _index = 0; _index < _linqitems.Length; _index++)
    {
        var _linqitem = _linqitems[_index];
        if (_linqitem % 10 == 0)
        {
            var _linqitem1 = _linqitem + 5;
            yield return _linqitem1;
        }
    }
}
```

آنچه که قابل حس است است ، استفاده از کلمه کلیدی yield است، این کامپایلر را برای انجام کارهای سخت به کار میگیرد بنابراین لیست موقتی برای ذخیره نتایج برای تخصیص نداریم. این یعنی آن که مقدار آن stream است .که در کد های اورجینال میباشد.

و در آخر،به یک مثال دیگر میپردازیم، الان با استفاده از اصطلاح:Count()

```
items.Where(i => i % 10 == 0)
    .Count();
```

ما در اینجا به طور واضح مشاهده کردیم که هر دو ابزار به طور قابل ملاحظه ای تخصیص (allocations)نسبت به کد اصلی LINQ را کاهش دادند.

```
// * Summary *

Host Process Environment Information:
BenchmarkDotNet.Core=v0.9.9.0
OS=Microsoft Windows NT 6.1.7601 Service Pack 1
Processor=Intel(R) Core(TM) i7-4800MQ CPU 2.70GHz, ProcessorCount=8
Frequency=2630761 ticks, Resolution=380.1181 ns, Timer=TSC
CLR=MS.NET 4.0.30319.42000, Arch=32-bit RELEASE
GC=Concurrent Workstation
JitModules=clrjit-v4.6.1590.0

Type=CountBenchmarks  Mode=Throughput
```

Method	Median	StdDev	Scaled	Scaled-SD	Gen 0	Gen 1	Gen 2	Bytes Allocated/Op
Iterative	2.9489 us	0.1286 us	1.00	0.00	-	-	-	0.16
Linq	4.6660 us	0.1710 us	1.58	0.08	-	-	-	15.02
RoslynLinqRewrite	2.9307 us	0.0691 us	0.99	0.05	-	-	-	0.15
LinqOptimiser	4.1259 us	0.2175 us	1.41	0.09	-	-	-	0.30

گزینه های آینده:

با اینکه RoslynLinqRewrite **یا** LinqOptimiser **خیلی راحت و ساده هستند ما باید کتابخانه سه قسمتی** a 3rd **را بر روی پروژه مان نصب کنیم.**

این زیبا تر نبود اگر کامپایلر .NET **یا** JITter **یا** runtime **همه ی این ها را برای ما بهینه میکردند ؟**

قطعا این ممکن است. همانطور که در QCon New York talk **و** work has already started **گفته شده زمان زیادی طول نمیکشد.**