

به نظر شما چه تعداد شیء CLR را می‌توان در یک ثانیه ایجاد کرد؟

برنامه کنسول زیر دو نسخه معمولی و نسخه پردازش موازی یک آزمایش ساده را برای اندازه گیری این مطلب ارائه می‌دهد:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks;

namespace ObjectInitSpeedTest
{
    class Program
    {
        //Note: don't forget to build it in Release mode.
        static void Main()
        {
            normalSpeedTest();
            parallelSpeedTest();

            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("Press a key ...");
            Console.ReadKey();
        }

        private static void parallelSpeedTest()
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine("parallelSpeedTest");

            long totalObjectsCreated = 0;
            long totalElapsedTime = 0;

            var tasks = new List<Task>();
            var processorCount = Environment.ProcessorCount;

            Console.WriteLine("Running on {0} cores", processorCount);

            for (var t = 0; t < processorCount; t++)
            {
                tasks.Add(Task.Factory.StartNew(
                    () =>
                    {
                        const int reps = 1000000000;
                        var sp = Stopwatch.StartNew();
                        for (var j = 0; j < reps; ++j)
                        {
                            new object();
                        }
                        sp.Stop();

                        Interlocked.Add(ref totalObjectsCreated, reps);
                        Interlocked.Add(ref totalElapsedTime, sp.ElapsedMilliseconds);
                    }
                ));
            }

            // let's complete all the tasks
            Task.WaitAll(tasks.ToArray());

            Console.WriteLine("Created {0:N} objects in 1 sec\n", (totalObjectsCreated /
            (totalElapsedTime / processorCount)) * 1000);
        }

        private static void normalSpeedTest()
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("normalSpeedTest");
        }
    }
}
```

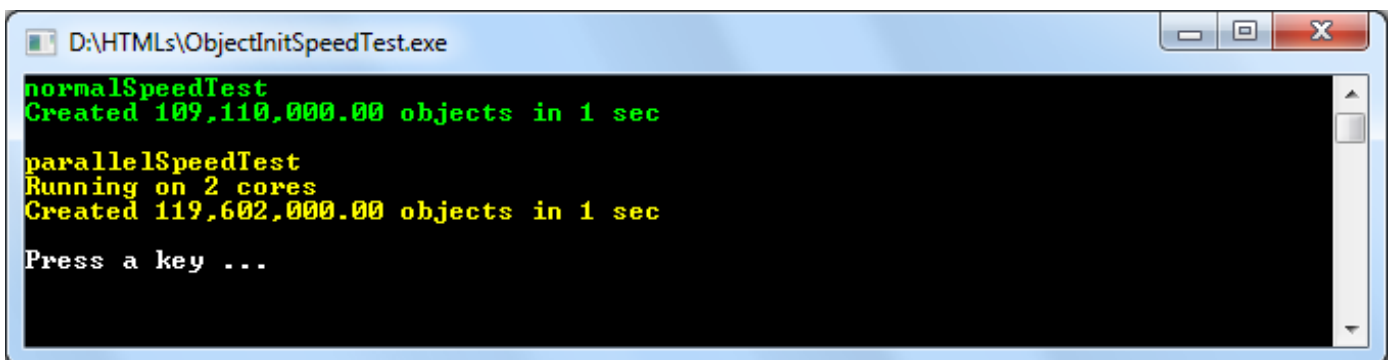
```

        const int reps = 1000000000;
        var sp = Stopwatch.StartNew();
        sp.Start();
        for (var j = 0; j < reps; ++j)
        {
            new object();
        }
        sp.Stop();

        Console.WriteLine("Created {0:N} objects in 1 sec\n", (reps / sp.ElapsedMilliseconds) *
1000);
    }
}

```

هنگام اجرای برنامه فراموش نکنید که حالت Build را بر روی Release قرار دهید.



```

D:\HTMLs\ObjectInitSpeedTest.exe
normalSpeedTest
Created 109,110,000.00 objects in 1 sec

parallelSpeedTest
Running on 2 cores
Created 119,602,000.00 objects in 1 sec

Press a key ...

```

نظرات خوانندگان

نویسنده: Faraz Fallahi
تاریخ: ۱۳:۲۴:۳۹ ۱۳۹۰/۰۳/۲۲

حالا چه نتیجه ای باید گرفت ؟!
<http://img.288.ir/images/objectinitspeedtest.png>

نویسنده: Faraz Fallahi
تاریخ: ۱۳:۳۷:۵۹ ۱۳۹۰/۰۳/۲۲

<http://img.288.ir/images/objectinitspeedtest2.png>

نویسنده: وحید نصیری
تاریخ: ۱۴:۰۹:۰۶ ۱۳۹۰/۰۳/۲۲

wow! این 32 هسته رو از کجا آوردید؟!

نویسنده: Faraz Fallahi
تاریخ: ۱۴:۵۶:۳۱ ۱۳۹۰/۰۳/۲۲

int processorCount = 32

نویسنده: Faraz Fallahi
تاریخ: ۱۵:۲۰:۴۰ ۱۳۹۰/۰۳/۲۲

مسئله اینجاس که با چهار برابر کردن مقدار processorCount واقعی نتیجه هم چهار برابر بهتر شده !
(ولی کردم 64 دیگه نتیجه زیاد با 32 تغییری نکرد.)

نویسنده: Hossein Moradinia
تاریخ: ۱۸:۳۰:۱۹ ۱۳۹۰/۰۳/۲۳

من که نفهمیدم چرا با تغییر processorcount به 32 زمان کاهش پیدا کرد!!!
اگه کسی میدونه بگه!!!

نویسنده: وحید نصیری
تاریخ: ۲۱:۴۹:۱۱ ۱۳۹۰/۰۳/۲۳

نسخه پردازش موازی فوق به صورت پیش فرض بر اساس تعداد هسته های CPU تنظیم شده یعنی تصور کنید که روی هر هسته واقعا نسخه های normalSpeedTest به موازات هم در حال اجرا هستند. اگر 8 هسته باشد، 8 tasks خواهیم داشت. زمانیکه دستی تعداد هسته رو تنظیم کنید (بیشتر از تعداد واقعی هسته ها البته)، به همان نسخه ی threading سابق بر می گردیم یعنی اینبار task های اضافه شده بر اساس زمان آزاد پردازشی هسته ها، بین آن ها مرتبا سوئیچ خواهند شد و هر کدام که زودتر پایان یابد زمان پردازشی بیشتری را در اختیار مابقی تردهای ایجاد شده قرار می دهد. بدیهی است افزایش این عدد زمانیکه CPU به حد اشباع رسیده است (احتمالا شاید بوی سوختن آن به مشام رسیده باشد !...)، تاثیر آنچنانی نخواهد داشت. علت اینکه عموما از تعداد هسته برای ایجاد parallel tasks استفاده می کنند هم همین مورد است. چون فقط برنامه شما نیست که قرار است از تمام توانایی های CPU استفاده کند.

نویسنده: وحید نصیری
تاریخ: ۲۲:۰۴:۱۷ ۱۳۹۰/۰۳/۲۳

یک مطلب دیگر رو هم اضافه کنم. ThreadPool در دات نت 4 به 64 logical processors محدود شده. به عبارتی مثلاً حین استفاده از Parallel.For/ForEach این محدودیت وجود دارد و پس از آزاد شدن یک task ، task بعدی (پس از 64 البته) وارد عمل خواهد شد و همینطور الی آخر

+

ویندوز سرور 2008 نگارش R2 فقط تا 256 logical processors رو پشتیبانی می‌کنه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۳۰ ۰۱:۱۲:۵۵

[Head-to-head benchmark: C++ vs .NET](#)

نویسنده: A. Karimi
تاریخ: ۱۳۹۰/۰۴/۰۳ ۱۴:۱۳:۵۶

ظاهراً پردازنده‌های نسل i مثل i5، i3 و i7 دارای مکانیزم موازی سازی متفاوتی هستند که در این موارد تاثیر زیادی دارند برای مثال از یک پردازنده یک هسته‌ای نسل i می‌توان توان دو هسته‌ای گرفت به شرط دادن توان الکتریکی کافی به پردازنده. به علاوه بسیاری از سخت افزارهای جدید از Overclocking به صورت پویا پشتیبانی می‌کنند و ریختن بار زیادتر بر روی CPU جواب بهتری می‌دهد. غیر از Overclocking وقتی از CPU های مدرن کمتر از حد توانشان استفاده شود بخش‌هایی از خودشان را موقتاً غیر فعال می‌کنند تا از مصرف برق بکاهند وقتی بار CPU زیاد شود این بخش‌های خاموش، روشن شده و قدرت CPU بیشتر خواهد شد.

شبیه همین مکانیزم در صورتی که سخت‌افزار مدرن باشد، درباره RAM نیز اتفاق می‌افتد. فکر می‌کنم به همین دلیل ریختن بار بیشتر بر CPU نتایجی دارد که گاهی عجیب به نظر می‌رسد.