RyuJIT: The next-generation JIT compiler for .NET

.NET Team

September 30th, 2013

> *This post introduces the .NET team's new 64-bit Just-In-Time (JIT) compiler. It was written by Andrew Pardoe, PM Manager for the CLR Runtime PM team.*

The world is moving to 64-bit computing even though it isn't always faster or more efficient than 32-bit. A lot of programs run faster on 32-bit than on 64-bit, for a variety of reasons. One example of this is the 64-bit JIT compiler in .NET. It does a great job of making your program run fast, but it's not a fast program itself. All that's about to change: a new, next-generation x64 JIT compiler that compiles code twice as fast is ready to change your impressions of 64-bit .NET code.

Remind me again about 64-bit

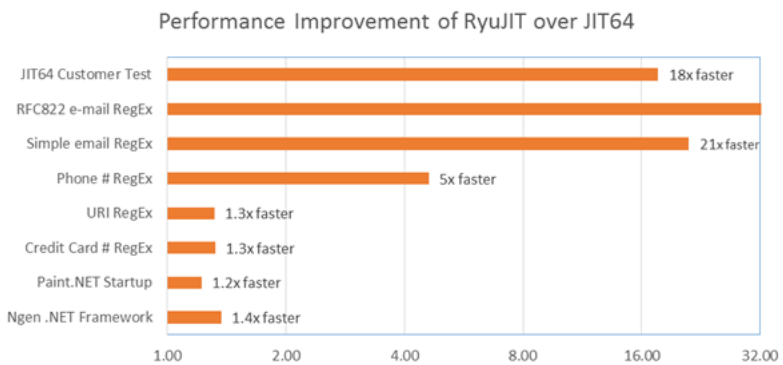
It seems like the 32-bit x86 computer has been around since the dawn of time. It's a great computer architecture, but it has one big problem: a 32-bit pointer can only address 4 GB of RAM. 64-bit computers, with their wider pointers, can address practically unlimited amounts of RAM. RAM was relatively expensive when 64-bit Windows was new so 64-bit machines were originally just used as servers. Nowadays 64-bit computing has gone mainstream and computers routinely ship with more than 4 GB of RAM. Even some smartphones have moved to 64 bit, despite the fact that they only have 1 GB of RAM installed.

The .NET 64-bit JIT was originally designed to produce very efficient code throughout the long run of a server process. This differs from the .NET x86 JIT, which was optimized to produce code quickly so that the program starts up fast. Taking time to compile efficient code made sense when 64-bit was primarily for server code. But "server code" today includes web apps that have to start fast. The 64-bit JIT currently in .NET isn't always fast to compile your code, meaning you have to rely on other technologies such as NGen or background JIT to achieve fast program startup.

RyuJIT to the rescue!

RyuWHAT? The .NET Code Generation team has been working on a new, next-generation x64 compiler, codenamed RyuJIT. This new JIT is twice as fast, meaning apps compiled with RyuJIT start up to 30% faster (Time spent in the JIT compiler is only one component of startup time, so the app doesn't start twice as fast just because the JIT is twice as fast.) Moreover, the new JIT still produces great code that runs efficiently throughout the long run of a server process.

This graph compares the compile time ("throughput") ratio of JIT64 to RyuJIT on a variety of code samples. Each line shows the multiple of how much faster RyuJIT is than JIT64, so higher numbers are better.



While all the code samples compile faster with RyuJIT, you should note the second line—RFC822 e-mail RegEx. It's literally off the chart! That's because regular expressions (a.k.a. RegEx) tend to perform especially badly on JIT64. Compiling this benchmark on JIT64 takes 60 seconds and consumes 1.4 GB of peak working set. With RyuJIT, it only takes 1.8 seconds and uses 199 MB of peak working set.

Paint.NET is a more normal example of RyuJIT wins. Its startup time went from 2.3 seconds on our test machine to 1.8 seconds. That indicates a compilation time decrease from 1510 ms on JIT64 to 970 ms on RyuJIT. Faster compilation with less memory usage makes everyone's code run better.

The best is yet to come

The performance gains and working set reductions of RyuJIT are fantastic, but they're not the best part of RyuJIT. When JIT64 was developed, we decided to base it off of the C++ team's optimizing compiler instead of basing it off of the existing 32-bit x86 JIT which is more optimized for dynamic compilation scenarios. Most 64-bit computers were being used as servers so it made sense to emphasize compiled code quality over compiler throughput. But this left the .NET CodeGen team with two compiler codebases to maintain. Adding features (and fixing bugs!) in two places slowed down the pace of innovation. Adding the ARM architecture and MDIL for Windows Phone 8.0 in recent years made it even harder to keep up.

RyuJIT is based off of the same codebase as the x86 JIT. While it's only for x64 right now, it's a modern compiler that will be the basis of all our JIT compilers in the future: x86, ARM, MDIL and whatever else comes along. Having a single codebase means that .NET programs are more consistent between architectures—put another way, you generally get bug-for-bug compatibility. But having a single codebase also means we can innovate faster and bring you more code generation features more quickly.

What do I need to do to get this?

RyuJIT is available right now as a CTP that you can try out in your non-production environment. It's not supported for production code right now but we definitely want to hear about any issues, bugs or behavioral differences that you encounter. Send feedback and questions to ryujit@microsoft.com. Even if you've figured out the issue or a workaround yourself we want to hear from you.

You can download the RyuJIT installer now. RyuJIT only works on 64-bit editions of Windows 8.1 or Windows Server 2012 R2. Also, RyuJIT doesn't change NGen on your system—we wanted to keep the CTP install clean. Lastly, if you enable RyuJIT while writing code, you'll find that Edit & Continue doesn't work on 64-bit (that's so 2012!)

After installation, there are two ways to turn on RyuJIT. If you just want to enable RyuJIT for one application, set an environment variable: COMPLUS_AltJit=*. If you want to enable RyuJIT for your entire machine, set the registry key HKLMSOFTWAREMicrosoft.NETFrameworkAltJit to the string "*". Both methods cause the 64-bit CLR to use RyuJIT instead of JIT64. And both are temporary settings—installing RyuJIT doesn't make any permanent changes to your machine (aside from installing the RyuJIT files in a directory, that is.)

Stay tuned to this blog for more on RyuJIT as it progresses from a CTP to become the One True .NET JIT Compiler™. If you want to dive more deeply into the geeky details behind RyuJIT's development, you can do that at the .NET CodeGen blog. We'll answer questions and explain design decisions there. (If you look right now, you'll see a post that explains the funny codename!) And remember to send us mail at ryujit@microsoft.com after you download and install the CTP. We want to hear from you!

.NET Team

Follow