Foreword

"Managed code is too slow. We've got to do this in C++."

—Anonymous

Back in 1999, the ACM published a study[1] that presented a comparison of 40 independent implementations of a computationally intensive problem, created by different programmers in either Java—the then-current managed runtime environment—or C/C++. It concluded with the finding that interpersonal differences between the developers "are much larger than the average difference between Java and C/C++" and that "performance ratios of a factor of 30 or more are not uncommon between the median programs from the upper half versus the lower half."

This should teach you something: If you are not a guru-level C++ programmer, then the chance is quite high that a managed code implementation performs as well as the average C++ solution—especially given the fact that most .NET languages simply allow you fewer possibilities to introduce subtle memory-related or performance-related issues. And keep in mind that this study was conducted several years ago, and that Just-In-Time Compilation (JIT) as well as memory management and garbage collection (GC) technologies have been improved in the meantime!

This however doesn't mean that you can't create horribly slow, memory eating applications with .NET. That's why you should be really concerned about the other part of the study's conclusion, namely that "interpersonal differences . . . are much larger." In essence, this means that you have to know about how to optimize your applications so that they run with the expected performance in a managed environment. Even though .NET frees you from a lot of tasks that in C++ would have been your responsibility as a developer, these tasks still exist; these "little puppets" have only cleared the main stage and now live in some little corner behind the scenes. If you want your application to run in the top performance range, you will still need to find the right strings to pull to move these hidden figures and to basically keep them out of the way of negatively affecting your application's performance.

1. Lutz Prechtelt, "Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences," *Communications of the ACM* 42, no. 10 (October 1999): 109–112

But knowing about the common language runtime's internals is still not enough, as lots of performance issues actually turn up during application design and not just during the coding stage. Collections, remoting, interoperability with unmanaged code, and COM components are not the only things that come to my mind in this regard.

It is the aim of Nick's book to enable you to understand the design issues as well as the underlying CLR mechanisms in order to create the programs that run on the better side of the 30-times performance difference quoted in the ACM study. Nick really managed to create a book that addresses these issues, which will otherwise turn up when carelessly coding to a managed environment. This book will allow you to get into the details without being overwhelmed by the underlying complexity of the common language runtime.

The only thing you have to resist after reading the book is the urge to overoptimize your code. I was delighted to see that Nick begins with a discussion of identifying an application's performance-critical sections and only later turns towards isolating and resolving these *real* performance bottlenecks. This is, in my opinion, one of the most important tasks—and also one of the most complex ones—when working with large-scale applications.

And now read on and enjoy the ride to the better side of a 30-fold performance difference.

Ingo Rammer, author of *Advanced .NET Remoting*
Vienna, Austria
`http://www.ingorammer.com`