PARTICLE TRACKING AND SIMULATION ON THE .NET FRAMEWORK*

H. Nishimura and T. Scarvie
ALS, LBNL, Berkeley, CA 94720, USA

Abstract

Particle tracking and simulation studies are becoming increasingly complex. In addition to the use of more sophisticated graphics, interactive scripting is becoming popular. Compatibility with different control systems requires network and database capabilities. It is not a trivial task to fulfill all the various requirements without sacrificing runtime performance. We evaluated the effectiveness of the .NET framework by converting a C++ simulation code to C#. The portability to other platforms is mentioned in terms of Mono.

INTRODUCTION

Scientific Computing and .NET

Software technology has been rapidly evolving by layering new concepts on top of traditional Object-Oriented Programming (OOP). One of these practices is the .NET Framework, [1] in which the C# programming language plays a major role.

On the other hand, scientific computing in general is far behind such trends and our field of accelerator modelling and simulation is no exception. If C# can provide the runtime performance sufficient for our computations, it makes sense to move our C++ codes to C# to take advantage of modern technologies that have been already used in other industries. This migration should not be difficult as C# is designed to be close to C++.

Early Experience with Java

We did a similar study[2] when Java came out. We ported a portion of our C++ library Goemon[3] to Java and evaluated its runtime performance. Contrary to our expectations, Java 1.2 showed even better performance than C++ for simple numerical calculations. However, the availability of critical math libraries, the compatibility with other programming languages and the graphical capabilities were not sufficient to migrate to Java at that time. The lack of operator overloading was also a significant problem. It should be noted that many of these issues have still not been resolved for Java.

C# and the .NET Platform

This time, we evaluate C#, since the situation looks much better. It is true that we spend most of our time in developing and debugging scientific programs, and since CPU speed has been significantly improved, the runtime performance should be confirmed with our real routines. If it is sufficient, we can port our programs to C# to take advantage of its various modern features (graphics, networking, database, threading, testing, and documentation tools) to create highly reliable software.

*This work was supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Material Sciences Division, U.S. Department of Energy, under Contract No. DE-AC03-76SF00098.

MIGRATING TO C#

Graphics Programming

Graphics programming is the primary reason that we keep developing programs in C++ on Win32. In case of Goemon, the physics modules have been separated from the graphics so they are portable to multiple platforms. However, if graphics are required, we primarily use Borland C++ Builder for efficient GUI development, and are therefore tied to Win32.

In case of .NET, however, graphics programming that is sufficient for our use is standardized as a part of the framework. If we use Mono[4], described later, the GUI programs can be made portable to other platforms.

Math Libraries

Various kinds of math libraries are available for C#. Externally calling routines in other languages is also simpler than it is in Java. Also, we can use existing ANSI C/C++ routines almost as they are by using C++/CLI[5]. Hence, the compatibility of C# with existing math routines is sufficient. At this stage, we only need singular value decomposition (SVD) routines from one of these libraries, since all the other math routines we need are available in C# already.

Differential Algebra

One of the many merits of using C++ is its flexible support of operator overloading for user-defined types. This capability was applied to handle the numerically exact differentiation called differential algebra (DA)[7], and used for lattice definitions. As operator overloading is well supported in C#, the port was straightforward. Our DA library in C++ allocates DA objects on stack to avoid dynamic allocations for faster execution speed. However, that technique is not possible in C#, so modifications were required on the client side to use the references effectively.

Lattice Definition

The lattice is defined in Goemon (C++) by using operator overloading and macro definitions effectively, as show below:

```
DRIFT(L, 1.23);
QUAD(QF, 0.15, 2.35);
Eline SEC1=L1+2*QF+L1;
```

Here the DRIFT macro creates a Drift object and registers it to the table for memory management. However, C# does not support this kind of macro definition, so we must use "new" explicitly:

```
DRIFT L=new DRIFT("L",1.23);
QUAD QF=new QUAD("QF".0.15,.235);
Eline SEC1=L1+2*QF+L1;
```

In this case, there is no need for memory management due to automatic and managed garbage collection mechanisms.

Runtime Performance

The length of Goemon is about 40K lines (excluding comments), and 30K lines have been rewritten in C# with redesign, clean up, refactoring using automated tools, and plentiful hand optimization that resulted in 15K lines of C# code. After confirming its correctness, the runtime profile was analyzed by using a profiler to improve execution speed. The routines modified during this process were mostly in the math units, including matrix, vector and DA routines.

Finally, the execution speed was measured using three tracking programs on a simplified ALS lattice with about 360 elements:

(1) 10,000 turns of a particle in 5-dim phase space.
(2) 1,000 turns of a particle in 6-dim phase space.
(3) 100 turns of a linear DA map.

Programs were written in both C++ and C# to produce identical results. Case (1) is basically multiplications of 4x5 matrices to a 5-dim vector which is a base of most of the linear calculations. Case (2) is in 6-dim including time. It uses the 2nd order symplectic integrator that is a series of drift-kick-drift segments. Each quadrupole or bending magnet uses 20 segments. Case (3) is for a DA map that uses DA objects intensively.

Tables 1 and 2 show the results from Pentium 4 PCs running Windows XP at 3.4 GHz and 2.8 GHz. The C++ compiler is Visual C++ 2005, and C# is Visual C# 2005 on .NET Framework 2.0.

Table 1. Performance on Intel Pentium 4 at 3.4 GHz.

Compiler	Test1	Test2	Test3
C++	0.283 sec	0.388 sec	0.247 sec
C#	0.233 sec	0.786 sec	0.244 sec

Table 2. Performance on Intel Pentium 4 at 2.8 GHz.

Compiler	Test1	Test2	Test3
C++	0.282 sec	0.469 sec	0.237 sec
C#	0.394 sec	0.995 sec	0.297 sec

Table 3 is the case of an AMD Athlon 4800+ PC running Windows XP x64 at 4.8 GHz. There are 32-bit and 64-bit versions in the case of C++, while the C# assembly stays unchanged.

Table 3. Performance on AMD Athlon 4800+ at 4.8 GHz

Compiler	Test1	Test2	Test3
C++ /32	0.263 sec	0.294 sec	0.197 sec
C++/64	0.247 sec	0.223 sec	0.145 sec
C#	0.220 sec	0.386 sec	0.153 sec

Although these tests are not regulated, they indicate that the runtime performance of C# is not only sufficient, but sometimes exceeds that of the native codes, due to careful optimizations.

The execution speeds of application programs are subject to the efficiency of their fitting routines, including closed-orbit finders. Therefore, it is not straight forward to compare two versions of programs in C++ and in C#. Generally speaking, C# runs a little bit slower than C++, but is always sufficiently fast in our experience.

Compared to our previous Java-based study[2], the execution speed has become about 20 times faster since 1999, when we were using Pentium 2 computers at 500 MHz. However, we still spend most of our time developing and debugging programs, and so we can safely conclude that the runtime performance of C# is quite adequate for our scientific computations.

Generic and Serialization

Goemon in C# is in a process of adopting two of the many new features of C#: generic and serialization.

Generic is a powerful mechanism of using the same routine with different variable types, which efficiently simplifies class design. In the case of C++, it is supported as templates. However, it often slows compiling speed significantly, and so is not suitable for the software development phase. On the other hand, C# supports generics with no noticeable penalty, so it can be used for our purposes very effectively.

Serialization is an automated load/restore mechanism of data. It simplifies and speeds up file I/O considerably.

Together, these two features will improve the design and implementation of Goemon in C# in the near future.

Managed/Unmanaged Routines

C# programs are managed by default. This means that resources such as memory are automatically managed by the system to run in a safe mode. However, when legacy external routines must be called, we can turn off this safety feature to allow unsafe behaviors. This mode is called unmanaged.

Up to this point, Goemon in C# is a managed code, and therefore is portable as described in the next section. Whenever faster speed is needed, or external calls are required, we have the option of making critical routines unmanaged.

C++/CLI can be used to access ANSI C/C++ routines with minimum modification by skipping a dynamic link library (DLL) layer.

PORTABILIY BROUGHT BY MONO

Microsoft supports .NET only on its Windows platforms. However, there have been several independent implementations that cover other operating systems. Mono[5] is one of these implementations, covering Windows, Linux, Mac, Solaris and BSD. While it does not completely support .NET 2.0, it has been adequate to run console mode C# programs developed on Windows. By booting the PC used in Table 2 from a Knoppix 4.02 DVD, we recompiled the C++ programs with GCC 3.4.2, and C# with MCS in Mono 1.1.8. The results are shown in Table 4.

Table 4. Performance on Intel Pentium 4 at 2.8 GHz.

Compiler	Test1	Test2	Test3
GC++	0.500 sec	0.680 sec	0.335 sec
MONO C#	0.471 sec	0.938 sec	0.342 sec

A .NET GUI program developed on Windows in C# using WinForm is also becoming portable on Mono. Basic WinForm components have been ported to other platforms already.

INTERACTIVE SCRIPTING

Interactive Scripting and C++

Interactive scripting is a common and useful tool for accelerator studies, but it typically has been outside the scope of OOP. However, applications such as the MATLAB Middle Layer[9] at the Advanced Light Source have demonstrated that it is quite powerful, especially for machine studies that may require continuous changes in programming logic.

We evaluated the use of Python to provide this capability with Goemon[10]. However, object methods have to be exported as C APIs (application programming interfaces) in a DLL, which is not a simple process since OOP features must be concealed.

Interactive Scripting and .NET

The reflection mechanism allows for figuring out the attributes of an object that has been dynamically loaded at runtime. This method makes multiple interactive scripting languages available today. For example, we can use IronPython[11] to access .NET assemblies directly at runtime. Although Python does not support matrices and graphics as Matlab does, this capability is quite powerful.

We take an example we used previously[9] and run it on IronPython with minor modifications:

```
>>> from Goemon import *
>>> SR=ALSSRW()
>>> SR.fitNuEta(14.20,8.20,0.06)
True
>>> SR.setKQD(7,2,SR.getKQD(7,2)*1.02)
>>> SR.setKHCM(27,0.001)
>>> SR.getCOD(0.0)
>>> X=SR.getBPMX(40)
>>> X
0.000979172815796
```

Here is the content:
(1) Loads the Goemon assembly built by C#.
(2) Create an object of the ALS storage ring (full lattice).
(3) Fit tunes and dispersion.
(4) Increase the strength of SR07C QD2 by 2%.
(5) Set the 27th horizontal steering to kick 1 mrad.
(6) Calculate the COD for an on-energy particle.
(7) Save the horizontal orbit at the 40th BPM to X.
(8) Print X.

We take this as one of the most significant benefits of moving to C#.

CONCLUSION

We have ported Goemon to C#. Its performance is sufficient for our accelerator modelling and simulation studies. IronPython works quite effectively for interactive scripting. Migration to .NET seems to be a reasonable choice at this point in time.

REFERENCES

[1] http://msdn.microsoft.com/netframework/.

[2] H. Nishimura, PCaPAC '99, Tsukuba, Japan, 1999. http://conference.kek.jp/pcapac99/cdrom/paper/tu/tu1.pdf.

[3] H. Nishimura, PAC'01, Chicago, p3066.

[4] H. Nishimura and C. Timossi, PCaPAC'05, Hayama, Japan,2005.

http://conference.kek.jp/pcapac2005/paper/TUB2.pdf

[5] http://www.mono-project.com.

[6] S.R.G. Fraser, " Pro Visual C++/CLI and the .NET2.0 Platform", Apress, ISBN 1590596404. 1005.

[7] M.Berz, SSC-152, 1988. Leo Michelotti, PAC'89, p839. N. Malitskey, et.al., SSCL-659,1994.

[9] G. Portmann, J. Corbett, A. Terebilo, PCaPAC'05, Hayama, Japan, 2005.

http://repositories.cdlib.org/cgi/viewcontent.cgi?article=3163&context=lbnl.

[10] H. Nishimura, C. A.Timossi, M. E. Urashka, T. Kosuge, K. Nigorikawa, PCaPAC'05, Hayama, Japan, 2005.

http://conference.kek.jp/pcapac2005/paper/WEP34.pdf

[11] http://www.codeplex.com/Wiki/View.aspx?ProjectName=IronPython.