bluebit

.NET Matrix Library (NML) 4.1

Software brochure and performance benchmarks

Linear Algebra for the .NET Framework

Bluebit .NET Matrix Library provides classes for object-oriented linear algebra for the .NET platform.

It can be used to solve systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalues and eigenvectors problems, and singular value problems. Also provided are the associated matrix factorizations such as Eigen, LQ, LU, Cholesky, QR and SVD.

The above functionality is present for both real and complex matrices. Two analogous sets of classes are provided for real and complex matrices, vectors and factorizations.

Get the best of two worlds

While exposing an easy to use and powerful interface, NML does not sacrifice any performance. Highly optimized BLAS and the standard LAPACK routines are used within the library and provide fast execution and accurate calculations.

NML has been developed as a mixed mode C++ project, combining together managed and unmanaged code and delivering the best of both worlds; the speed of native C++ code and the feature-rich and easy to use environment of the .NET Framework.

Internally it uses highly optimized code at processor level. This means that the processor type is detected at runtime and different brunches of code are executed in order to achieve optimal performance.

Native 64-bit support

Starting from version 4.1, NML is available in separate builds each one targeting either the 32-bit or the 64-bit platform.

- The 32-bit version of NML 4.1 will run on all the 32-bit versions of Windows and also will run on 64-bit Windows as a 32-bit process.
- The 64-bit version of NML 4.1 will run as a native 64-bit process on 64-bit versions of Windows offering an additional 10 to 30% performance boost.

Parallelism

NML 4.1 makes full use of the capabilities of modern processors. It achieves impressive performance gains through parallelism provided by the newest multi-core architectures. Applications using NML 4.1 can benefit from its multithreading capabilities in either of the following ways:

- On single threaded applications, NML 4.1 is by default configured to distribute the computational workload on all physical cores of the machine, dramatically reducing the execution time especially when it comes to operations on large matrices.
- Being thread safe, NML 4.1 allows its use in threaded applications. Threads defined in client applications can use objects of the library and execute methods independently of other threads.

Starting from version 4.1, new static methods have been introduced allowing a more precise control of the threading behavior of the library.

Performance Benchmarks

The following benchmarks were run on a machine with the following configuration:

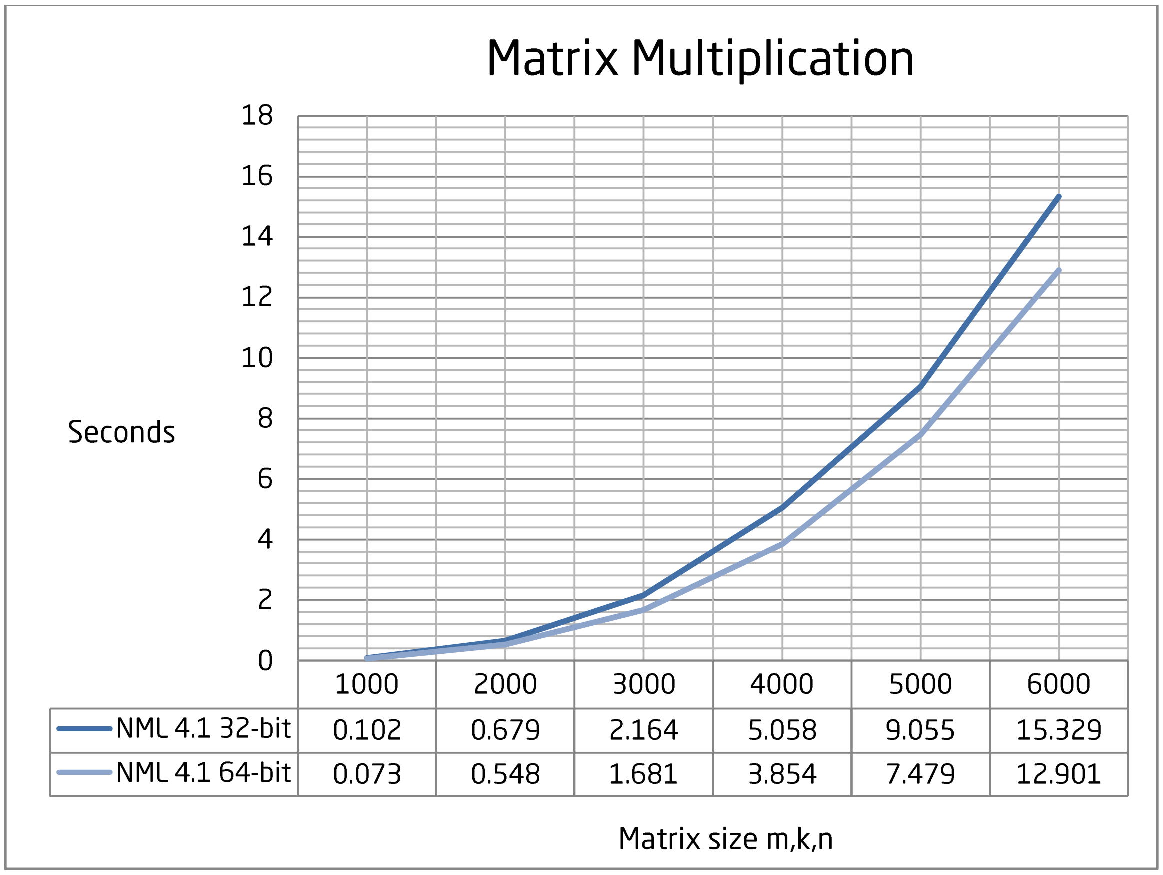
Processor:	Intel® Core™2 Quad CPU Q6600 @2.4GHz
Motherboard:	ASUS P5K, Intel P35
RAM:	2.00 GB DDR2 1066MHz
OS:	Windows Vista 32-bit / Windows Vista 64-bit

The benchmark program we have used was the one found in the samples folder of NML installation. Time measurement is based on the functions of the windows API QueryPerformanceCounter and QueryPerformanceFrequency. In case of the matrix multiplication benchmark the command line was:

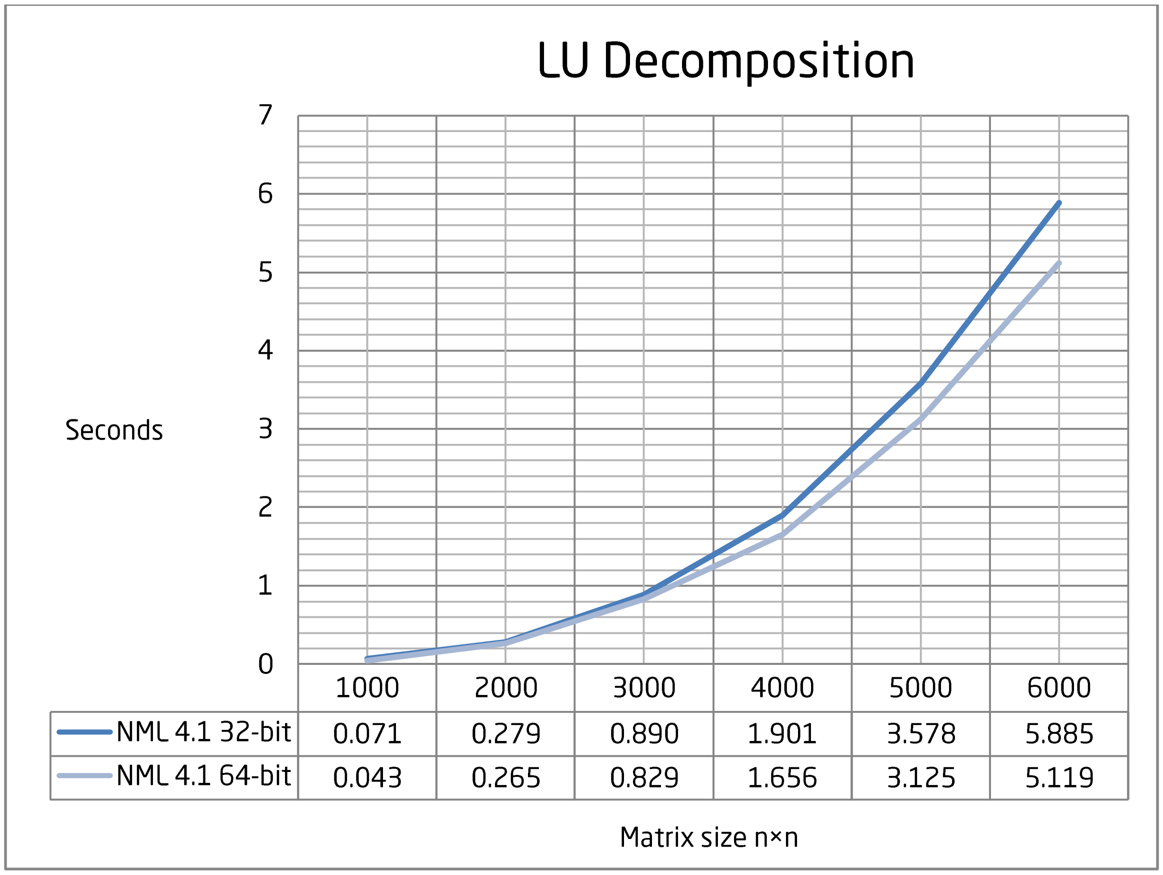
```
bench /MUL 50 100 250 500 1000 2000 3000 4000 5000 6000 >logMult.txt
```

We measured timings for NML 4.1 32-bit on Windows Vista 32-bit and for NML 4.1 64-bit on Windows Vista 64-bit.

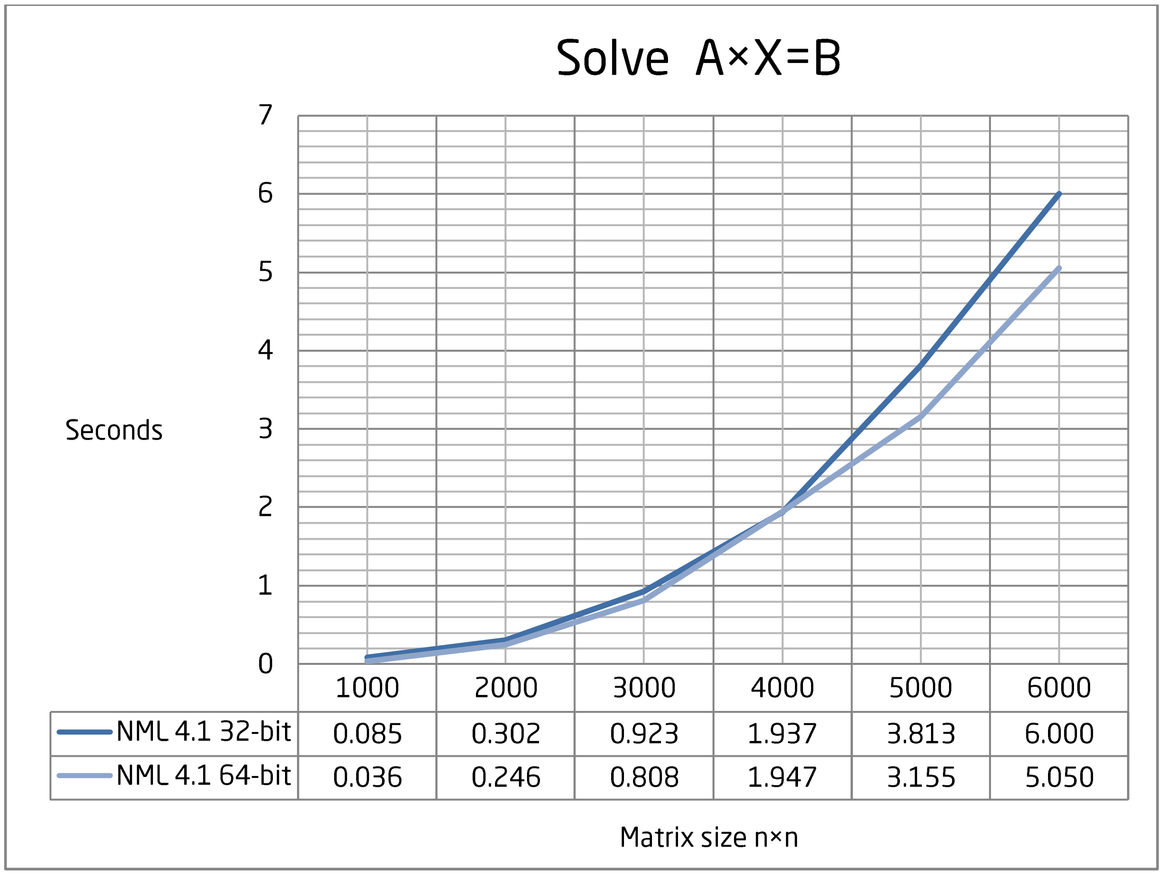
The results are shown in the following charts:



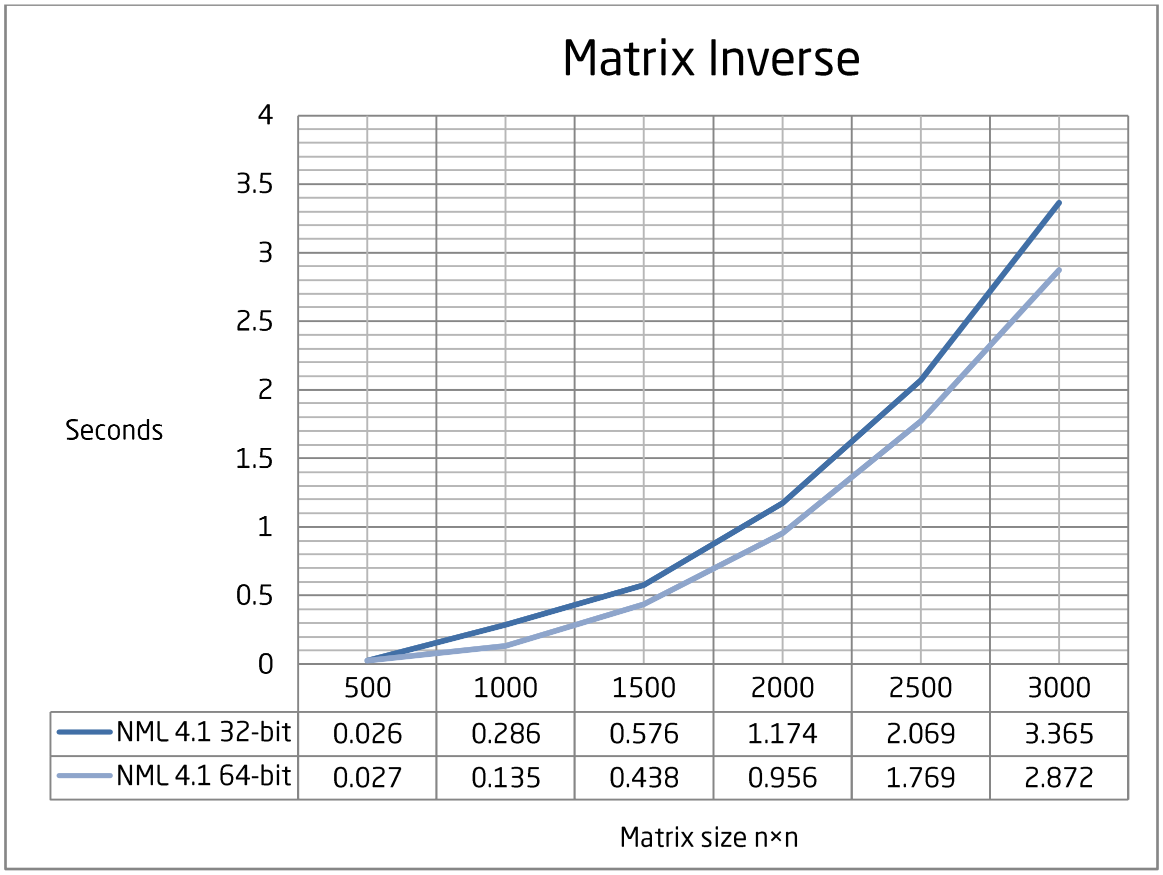
	1000	2000	3000	4000	5000	6000
NML 4.1 32-bit	0.102	0.679	2.164	5.058	9.055	15.329
NML 4.1 64-bit	0.073	0.548	1.681	3.854	7.479	12.901



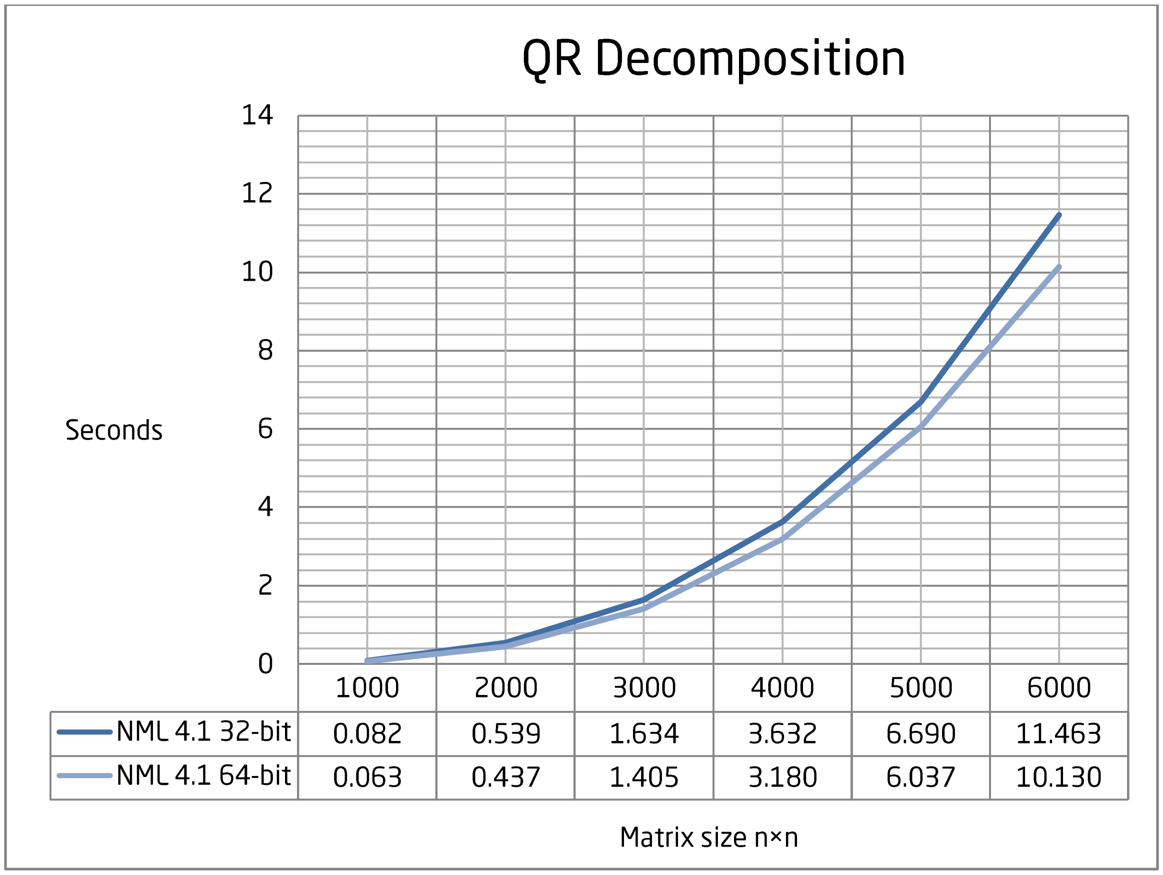
	1000	2000	3000	4000	5000	6000
NML 4.1 32-bit	0.071	0.279	0.890	1.901	3.578	5.885
NML 4.1 64-bit	0.043	0.265	0.829	1.656	3.125	5.119



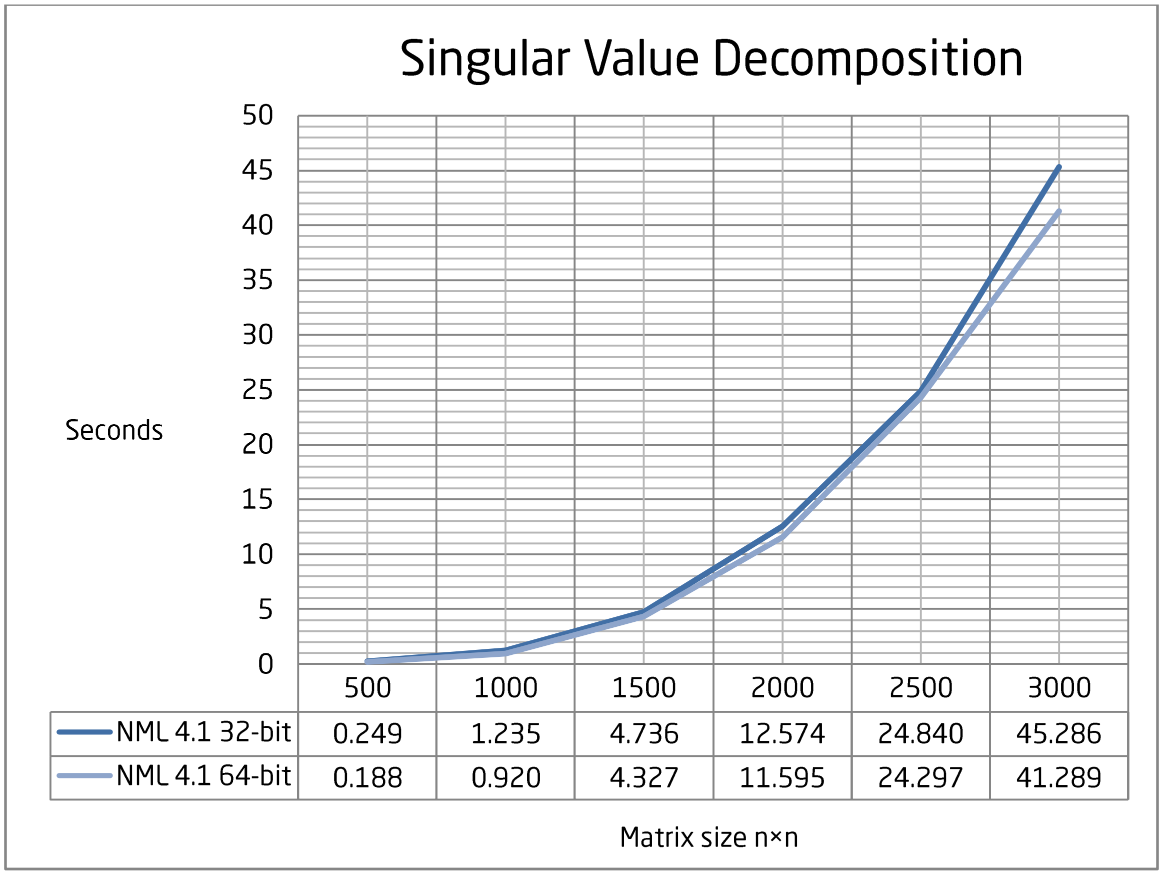
	1000	2000	3000	4000	5000	6000
NML 4.1 32-bit	0.085	0.302	0.923	1.937	3.813	6.000
NML 4.1 64-bit	0.036	0.246	0.808	1.947	3.155	5.050



	500	1000	1500	2000	2500	3000
NML 4.1 32-bit	0.026	0.286	0.576	1.174	2.069	3.365
NML 4.1 64-bit	0.027	0.135	0.438	0.956	1.769	2.872

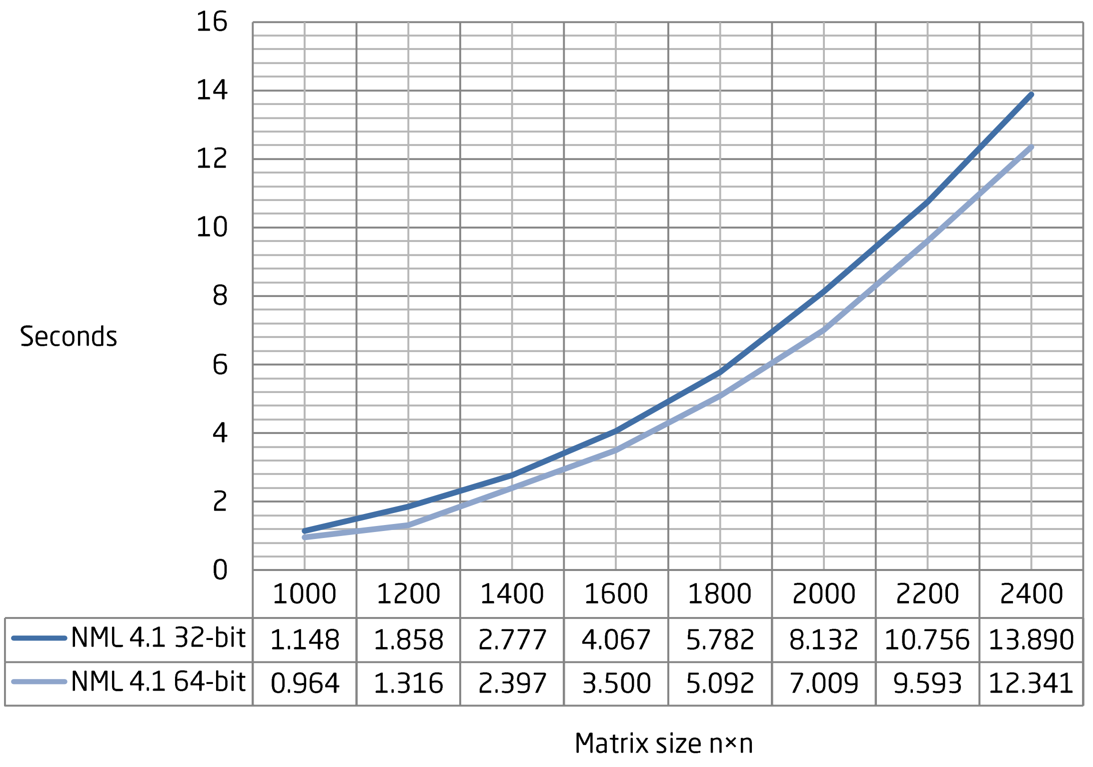


	1000	2000	3000	4000	5000	6000
NML 4.1 32-bit	0.082	0.539	1.634	3.632	6.690	11.463
NML 4.1 64-bit	0.063	0.437	1.405	3.180	6.037	10.130



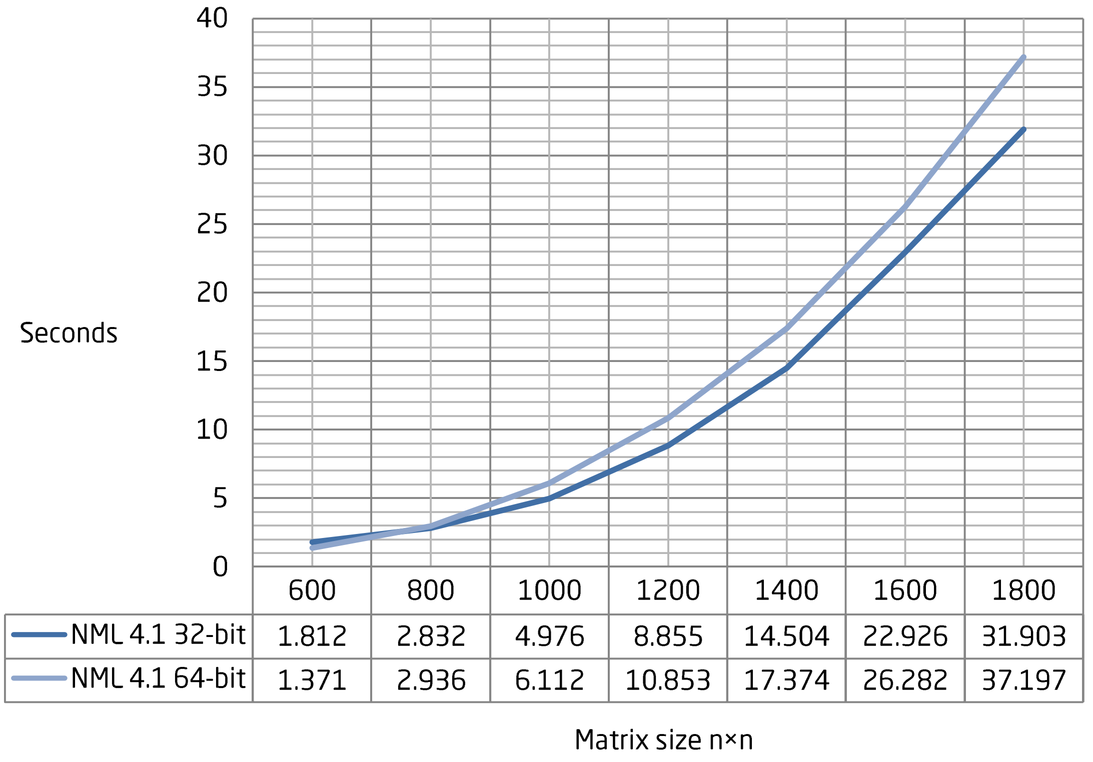
	500	1000	1500	2000	2500	3000
NML 4.1 32-bit	0.249	1.235	4.736	12.574	24.840	45.286
NML 4.1 64-bit	0.188	0.920	4.327	11.595	24.297	41.289

Symmetric Eigenvalues-Eigenvectors



	1000	1200	1400	1600	1800	2000	2200	2400
NML 4.1 32-bit	1.148	1.858	2.777	4.067	5.782	8.132	10.756	13.890
NML 4.1 64-bit	0.964	1.316	2.397	3.500	5.092	7.009	9.593	12.341

General Eigenvalues-Eigenvectors



	600	800	1000	1200	1400	1600	1800
NML 4.1 32-bit	1.812	2.832	4.976	8.855	14.504	22.926	31.903
NML 4.1 64-bit	1.371	2.936	6.112	10.853	17.374	26.282	37.197

Note:

The above results are indicative only. Users should test and verify performance in their own environment.