

# A Very Simple Benchmark for Brutal-force Loops in Several Languages

Revised

The purpose of this simple benchmark is to test speed of executing brutal-force loops with floating-point calculation for various high-level programming languages on Windows. (For work reason I have to stick to Windows, unfortunately.) I am particularly curious if Julia can be shown to approach C speed in this test.

It is well known that we should maximize the use of vectorized calculation in both Python and Julia as they employ blazing-fast C-based array libraries under the hood. However, in my work I often found myself not able to vectorize things – the next step result usually depends on the previous step in a way that cannot be determined beforehand. I love Python but hate its slow speed when loops are unavoidable. That's why I am particularly interested to see if Julia can deliver on the promise that, in addition to efficient vectorized calculations, it approaches C speed in traditional algorithms that involves a lot of loops.

The test is very simple: loop through 1,000,000 elements in a double array and do some floating-point calculations involving cosine and random number generation. Then repeat this for 100 times. This is done for Python, Julia, Visual C++, C#.NET, Java, and Matlab on a Lenovo Thinkpad W530 laptop computer with Intel Core i7-3740QM CPU @ 2.70GHz (4 physical cores, 2 logical cores per physical), 24 GB RAM, 64-bit Windows 7 Professional SP1.

In Julia and Matlab, the functions used in the loops are deliberately called before the stop watch starts, thus (hopefully) ensuring that the one-time JIT overhead is not included in the measured execution time. Note I didn't do this for C#.NET or Java because they are compiled in my eyes, though I am aware that both .NET engine and Java VM employ JIT compilation to convert bytecode to native code.

Also random number generations are used deliberately in both the inner and outer loops. I am worried a smart compiler/interpreter might realize I am repeating identical calculations and somehow optimize part of the calculations away. By adding randomness in every loop I hope this would not happen.

(Note: Matlab complained about “Out of memory” when I set the number of elements *nsamples* to 1,000,000 or even 100,000. So I set *nsamples* to 10,000, and increased the outer loop count from 100 to 10,000.)

## Result summary: measured execution time (smaller is faster)

Time	Python	Julia	Visual C++	C#.NET	Java	Matlab
Seconds	262.26	3.14	4.08	4.46	12.10	13.60
VC++ = 1.0	64.3	0.8	1.0	1.1	3.0	3.3

## Comments:

1. Julia's speed is amazing in this test! It's 84 times faster than Python, and even slightly faster than VC++. My first attempt actually made the mistake of using global variables, and the measure time was 50.32 sec. Simply wrapping the whole benchmark in a function did the magic.
2. It surprised me how fast C#.NET is! Maybe many .NET applications are burdened by the complicated object-oriented designs, but the underlying language itself is surely fast!
3. As expected, Java and Matlab are several times slower than VC++, but the speeds are decent probably thanks to the JIT technology they employ.

Details such as the compiler/interpreter versions, source code, and console output can be found in the tables below.

<h2 style="text-align: center;">Python</h2> <p style="text-align: center;">2.7.5  Anaconda 2.1.0 (64-bit)  (default, Jul 1 2013, 12:37:52) [MSC v.1500 64 bit (AMD64)] IDE: JetBrains PyCharm Community Edition 3.4.1</p>	<h2 style="text-align: center;">Julia</h2> <p style="text-align: center;">Version 0.4.6 (2016-06-19 17:16 UTC), x86_64-w64-mingw32 IDE: Juno (Atom 1.8.0 with julia-client 0.4.3)</p>
<pre> from __future__ import division, print_function from numpy import * from time import *  nsamples = 1000000  x = linspace(0, 5, nsamples) y = zeros(nsamples) noise = zeros(nsamples)  print("\nBrutal-force loops, 100 times:") t0 = time() for m in range(100):     morenoise = random.randf()     for n in range(nsamples):         noise[n] = random.randf()         y[n] = cos(2*x[n] + 5) + noise[n] + morenoise te = (time() - t0) print("\tTime elapsed: {:f} sec".format(te)) </pre>	<pre> function benchmark()     nsamples = 1000000;      x = linspace(0, 5, nsamples);     y = zeros(nsamples);     noise = zeros(nsamples);      # attempt to trigger JIT to compile all functions needed in the loops     before     # profiling     a = rand(); a = cos(0.0); a = 1.5*2.5; a = 1.5+2.5;      println("\nBrutal-force loops, 100 times:")     @time(         for m = 1:100             morenoise = rand();             for n = 1:nsamples                 noise[n] = rand();                 y[n] = cos(2*x[n]+5) + noise[n] + morenoise;             end         end     ) end  benchmark(); </pre>
<p><b>Brutal-force loops, 100 times:</b> Time elapsed: 262.259000 sec</p>	<p><b>Brutal-force loops, 100 times:</b> 3.136876 seconds</p>

<h2 style="text-align: center;">Visual C++</h2> <p style="text-align: center;">Visual C++ 2015 00322-20000-00000-AA082 Build config: Release x64 IDE: Microsoft Visual Studio Community 2015 Version 14.0.25123.00 Update 2</p>	<h2 style="text-align: center;">C#.NET</h2> <p style="text-align: center;">Microsoft .NET Framework Version 4.6.01055 Visual C# 2015 00322-20000-00000-AA082 Build config: Release x64 IDE: Microsoft Visual Studio Community 2015 Version 14.0.25123.00 Update 2</p>
<pre> // CppVsJulia.cpp : Defines the entry point for the console application. //  #include "stdafx.h" #include &lt;stdlib.h&gt; #include &lt;time.h&gt; #include &lt;math.h&gt;  int main() {     const int nsamples = 1000000;      double *x = new double[nsamples];     double *y = new double[nsamples];     double *noise = new double[nsamples];     double morenoise;      for (int n = 0; n &lt; nsamples; n++) {         x[n] = 5.0 * n / nsamples;     }      srand((unsigned)time(NULL));      printf("\nBrutal-force loops, 100 times:\n");     clock_t t0 = clock();     for (int m = 0; m &lt; 100; m++) {         morenoise = (double)rand() / RAND_MAX;         for (int n = 0; n &lt; nsamples; n++) {             noise[n] = (double)rand() / RAND_MAX;             y[n] = cos(2 * x[n] + 5) + noise[n] + morenoise;         }         clock_t t1 = clock();         printf("\tTime elapsed: %f sec", double(t1 - t0) / CLOCKS_PER_SEC);          getchar();     }     return 0; } </pre>	<pre> using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;  namespace CSharpVsJulia {     class Program     {         static void Main(string[] args)         {             int nsamples = 1000000;             double[] x, y, noise;             Random rand;             double morenoise;              x = new double[nsamples];             y = new double[nsamples];             noise = new double[nsamples];             rand = new Random();              for (int n = 0; n &lt; nsamples; n++)                 x[n] = 5.0 * n / nsamples;              Console.WriteLine("\nBrutal-force loops, 100 times:");             DateTime t0 = DateTime.Now;             for(int m=0; m&lt;100; m++)             {                 morenoise = rand.NextDouble();                 for(int n=0; n&lt;nsamples; n++)                 {                     noise[n] = rand.NextDouble();                     y[n] = Math.Cos(2 * x[n] + 5) + noise[n] + morenoise;                 }                 DateTime t1 = DateTime.Now;                 Console.WriteLine(string.Format("\tTime elapsed: {0} sec", (t1 - t0).TotalSeconds));                 String input = Console.In.ReadLine();             }         }     } } </pre>
<p><b>Brutal-force loops, 100 times:</b> Time elapsed: 4.082000 sec</p>	<p><b>Brutal-force loops, 100 times:</b> Time elapsed: 4.4562548 sec</p>

<p style="text-align: center;"><b>Java</b></p> <p style="text-align: center;">java version "1.8.0_91"  Java(TM) SE Runtime Environment (build 1.8.0_91-b15)  Java HotSpot(TM) 64-Bit Server VM (build 25.91-b15, mixed mode)  IDE: IntelliJ IDEA Community Edition 14.1</p>	<p style="text-align: center;"><b>Matlab</b></p> <p style="text-align: center;">R2013b (8.2.0.701), 64-bit (win64), August 13, 2013</p>
<pre> package com.company; import java.util.Random;  public class Main {      public static void main(String[] args) {         // write your code here         int nsamples = 1000000;         double[] x, y, noise;         double morenoise;         Random rand = new Random();          x = new double[nsamples];         y = new double[nsamples];         noise = new double[nsamples];          for(int n=0; n&lt; nsamples; n++){             x[n] = 5.0 * n / nsamples;         }          System.out.println("\nBrutal-force loops, 100 times:\n");         long t0 = System.currentTimeMillis();         for(int m=0; m&lt;100; m++){             morenoise = rand.nextDouble();             for(int n=0; n&lt;nsamples; n++){                 noise[n] = rand.nextDouble();                 y[n] = Math.cos(2*x[n] + 5) + noise[n] + morenoise;             }         }         long t1 = System.currentTimeMillis();         System.out.format("\tTime elapsed: %f sec", (t1-t0)/1000.0);     } } </pre>	<pre> clc; clear; close all; nsamples = 10000;  x = linspace(0, 5, nsamples); y = zeros(nsamples); noise = zeros(nsamples);  % attempt to trigger JIT to compile all functions needed in the loops % before profiling a = rand(); a = cos(0.0); a = 1.5*2.5; a = 1.5+2.5;  fprintf('\nBrutal-force loops, 10000 times:\n\t') tic     for m = 1:10000         morenoise = rand;         for n = 1:nsamples             noise(n) = rand;             y(n) = cos(2*x(n)+5) + noise(n) + morenoise;         end     end toc </pre>
<p><b>Brutal-force loops, 100 times:</b></p> <p style="text-align: center;">Time elapsed: 12.099000 sec</p> <p>Process finished with exit code 0</p>	<p><b>Brutal-force loops, 10000 times:</b></p> <p style="text-align: center;">Elapsed time is 13.603702 seconds.</p>

Note: Matlab complained about “Out of memory” when I set *nsamples* to 1,000,000 or even 100,000. So I set *nsamples* to 10,000, and increased the outer loop count from 100 to 10,000.