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

Rev 3

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, unfortunatley.) 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. 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 delibretely 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.

Result summary: measured execution time (smaller is faster)

Time	Python	Julia	Visual C++	C#.NET	Java	Matlab
Seconds	156.60	2.55	1.60	3.47	10.09	7.77
VC++ = 1.0	97.9	1.6	1.0	2.2	6.3	4.9

Comments:

1. Julia's speed is amazing in this test! It's 61 times faster than Python, and only slower than VC++. My first attempt actually made the mistake of using global variables which made Julia slow. Simply wrapping the whole benchmark in a function achieved 18x speedup.
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.

Python

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

```
from __future__ import division, print_function
from numpy import *
from time import *

def benchmark():
    nsamples = 1000000

    x = linspace(0, 5, nsamples)
    y = zeros(nsamples)

    print("\nBrutal-force loops, 100 times:")
    t0 = time()
    for m in range(100):
        for n in range(nsamples):
            y[n] = cos(2*x[n] + 5)
    te = (time() - t0)
    print("\tTime elapsed: {:f} sec".format(te))

benchmark()
```

```
Brutal-force loops, 100 times:
         Time elapsed: 156.601000 sec
```

Julia

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)

```
function benchmark()
  nsamples = 1000000;

  x = linspace(0, 5, nsamples);
  y = zeros(nsamples);

  # attempt to trigger JIT to compile all functions needed in the loops
before profiling
  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
      for n = 1:nsamples
        y[n] = cos(2*x[n]+5);
      end
    end
  )
end

benchmark();
```

```
Brutal-force loops, 100 times:
  2.548241 seconds
```

Visual C++

Visual C++ 2015 00322-20000-00000-AA082
Build config: Release x64
IDE: Microsoft Visual Studio Community 2015 Version 14.0.25123.00 Update 2

```
// CppVsJulia.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <stdlib.h>
#include <time.h>
#include <math.h>

int main()
{
          const int nsamples = 1000000;

          double *x = new double[nsamples];
          double *y = new double[nsamples];

          for (int n = 0; n < nsamples; n++) {
                    x[n] = 5.0 * n / nsamples;
          }

          printf("\nBrutal-force loops, 100 times:\n");
          clock_t t0 = clock();
          for (int m = 0; m < 100; m++) {
                    for (int n = 0; n < nsamples; n++) {
                              y[n] = cos(2 * x[n] + 5);
                    }
          }
          clock_t t1 = clock();
          printf("\tTime elapsed: %f sec", double(t1 - t0) /
CLOCKS_PER_SEC);

          getchar();
    return 0;
}
```

```
Brutal-force loops, 100 times:
         Time elapsed: 1.601000 sec
```

C#.NET

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

```
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;

            x = new double[nsamples];
            y = new double[nsamples];

            for (int n = 0; n < nsamples; n++)
                x[n] = 5.0 * n / nsamples;

            Console.WriteLine("\nBrutal-force loops, 100 times:");
            DateTime t0 = DateTime.Now;
            for(int m=0; m<100; m++)
            {
                for(int n=0; n<nsamples; n++)
                    y[n] = Math.Cos(2 * x[n] + 5);
            }
            DateTime t1 = DateTime.Now;
            Console.WriteLine(string.Format("\tTime elapsed: {0} sec",
(t1 - t0).TotalSeconds));
            String input = Console.In.ReadLine();
        }
    }
}
```

```
Brutal-force loops, 100 times:
        Time elapsed: 3.4721986 sec
```

Java	Matlab
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	R2013b (8.2.0.701), 64-bit (win64), August 13, 2013

`package com.company;`
 	
`public class Main {`
 	
`    public static void main(String[] args) {`
`        // write your code here`
`        int nsamples = 1000000;`
`        double[] x, y;`
 	
`        x = new double[nsamples];`
`        y = new double[nsamples];`
 	
`        for(int n=0; n< 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<100; m++){`
`            for(int n=0; n<nsamples; n++)`
`                y[n] = Math.cos(2*x[n] + 5);`
`        }`
`        long t1 = System.currentTimeMillis();`
`        System.out.format("\tTime elapsed: %f sec", (t1-t0)/1000.0);`
`    }`
`}`
	`clc; clear; close all;`
	`nsamples = 1000000;`
	
	`x = linspace(0, 5, nsamples);`
	`y = zeros(1, nsamples);`
	
	`% attempt to trigger JIT to compile all functions needed in the loops`
	`% before profiling`
	`a = cos(0.0); a = 1.5*2.5; a = 1.5+2.5;`
	
	`fprintf('\nBrutal-force loops, 100 times:\n\t')`
	`tic`
	`  for m = 1:100`
	`    for n = 1:nsamples`
	`      y(n) = cos(2*x(n)+5);`
	`    end`
	`  end`
	`toc`

`Brutal-force loops, 100 times:`
 	
`        Time elapsed: 10.089000 sec`
`Process finished with exit code 0`
	`Brutal-force loops, 100 times:`
	`        Elapsed time is 7.774488 seconds.`