Boo double precision benchmark

This is a comparative double precision benchmark in C#, C++, Java, Python, and Boo. It computes a piece of the Mandelbrot set. The times to complete the benchmark on a 1.7GHz Pentium M IBM T41p running Windows XP Professional SP1:

Language / runtime	Normalized time	Actual time (secs)
Visual C# 2005 Express beta / .Net 2.0 beta	.64	6.44
boo 0.4.5 / .Net 2.0 beta	.64	6.44 ⭐
Visual C++ 2005 Express beta / Win32	.66	6.68 ⭐
Visual Studio .Net 2003 C++ / Win32	1	10.12 (approx)
boo 0.4.3 / .Net 1.1	1	10.12
SharpDevelop C# / .Net 1.1	1.03	10.43
Sun Java 1.4.1_01	1.22	12.3 (approx)
ActivePython 2.3.2	89.31	903.79

⭐ - Benchmark values extrapolated from normalized times.

Boo's static typing speeds it up dramatically over Python. Avery Andrews at Australian National University reports a 5x Python speedup using psyco:

```
Using psyco:

import psyco
psyco.fill()

brings the 512 1000 case from 143.5 down to
27, pretty close to the
average 5x improvement that psyco tends to
bring (Python 2.3)
```

The C++ version, compiled under Visual Studio .Net 2003 for the Win32 runtime, is no faster than boo and much slower than C# 2.0, raising the question of what optimizations Microsoft has applied in .Net 2.0?

The benchmark code is reproduced below for boo, C#, C++, Java, and Python. Run the benchmark with the command line "mbtest 1500 1000". 1500 is the height and width of a square of Mandelbrot pixels to calculate, 1000 is the number of iterations to run on each pixel before giving up and assigning a color of 0.

boo version

```
/*
 * boo version - Mandelbrot Benchmark by
Bill Wood
 */

def mb_d(xl as double, xs as double, yl as
double, ys as double, ix as int, iyl as int,
iyh as int, r as double, iterations as int,
buf as (short)):

    bp = 0
    for iy in range(iyl, iyh):
        cx = xl + ix * xs
```

```
        ci = yl + iy * ys
#        rx2 = ri2 = rx = ri = 0.0
        rx2 = 0.0; ri2 = 0.0; rx = 0.0; ri =
0.0
        count = 0
        while ((rx2 + ri2) <= r) and (count
< iterations):
            ri = (rx + rx) * ri + ci
            rx = rx2 - ri2 + cx
            rx2 = rx * rx
            ri2 = ri * ri
            count += 1
        if (rx2 + ri2 > r) and (count <=
iterations):
            buf[bp] = count
            bp += 1
        else:
            buf[bp] = 0
            bp += 1

def main(argv as (string)):

    xl = -0.74526593488600
    yl = 0.11303858131900
    xh = -0.74525997120900
    yh = 0.11304454499600
    r = 4.0

    size = int.Parse(argv[0])
    iterations = int.Parse(argv[1])
    print ("size = $size, iterations =
```

```
$iterations")

    buf = array(short, size)
    xs = (xh - xl) / (size - 1)
    ys = (yh - yl) / (size - 1)

    start = date.Now
    for ix in range(0, size):
        mb_d(xl, xs, yl, ys, ix, 0, size, r,
iterations, buf)
    elapsed = date.Now.Subtract(start)
    print ("Boo elapsed time = $elapsed")
#    for i in buf:
#        print(i)
```

```
main(argv)
```

C# version

```
/*
 * C# version - Mandelbrot Benchmark by Bill
Wood
 */
using System;

namespace MBTest
{
    class MBTest {

        static void mb_d(double xl, double
xs, double yl, double ys, int ix, int iyl,
int iyh, double r, int iterations, short[]
buf) {
            double cx, ci, rx, rx2, ri, ri2;
            int iy, count, bp = 0;

            for (iy = iyl; iy <= iyh; iy++)
{
                cx = xl + ix * xs;
                ci = yl + iy * ys;
                rx2 = ri2 = rx = ri = 0;
                for (count = 0; rx2 + ri2 <=
r && count < iterations; count++) {
                    ri = (rx + rx) * ri +
```

```
ci;
                    rx = rx2 - ri2 + cx;
                    rx2 = rx * rx;
                    ri2 = ri * ri;
                    }
                if (rx2 + ri2 > r && count
<= iterations)
                    buf[bp++] =
(short)count;
                else
                    buf[bp++] = 0;
                }
            }

        static void Main(string[] args) {
            double xl, xh, xs, yl, yh, ys,
r;
            int ix, iterations, size;
            short[] buf;

            xl = -0.74526593488600;
            yl = 0.11303858131900;
            xh = -0.74525997120900;
            yh = 0.11304454499600;
            r = 4.0;

            size = Int32.Parse(args[0]);
            iterations =
Int32.Parse(args[1]);
            Console.WriteLine("size = " +
size + ", iterations = " + iterations);
```

```
            buf = new short[size];
            xs = (xh - xl) / (size - 1);
            ys = (yh - yl) / (size - 1);

            DateTime start = DateTime.Now;
            for (ix = 0; ix < size; ix++)
                mb_d(xl, xs, yl, ys, ix, 0,
size - 1, r, iterations, buf);
            Console.WriteLine("C# elapsed
time = " + (DateTime.Now - start));
//            for (ix = 0; ix < size; ix++)
//                Console.Write(buf[ix] + "
");
            }
```

```
        }
}
```

Java version

```
/*
 * Java version - Mandelbrot Benchmark by
Bill Wood
 */

public class mbtest {

    public static void mb_d(double xl,
double xs, double yl, double ys, int ix, int
iyl, int iyh, double r, int iterations,
short buf[]) {
        double cx,ci, rx,rx2,ri,ri2;
        int iy, count, bp = 0;

        for (iy = iyl; iy <= iyh; iy++) {
            cx = xl + ix*xs;
            ci = yl + iy*ys;
            rx2 = ri2 = rx = ri = 0;
            for (count = 0; rx2 + ri2 <= r
&& count < iterations; count++) {
                ri = (rx+rx)*ri + ci;
                rx = rx2 - ri2 + cx;
                rx2 = rx*rx;
                ri2 = ri*ri;
            }
```

```
            if (rx2 + ri2 > r && count <=
iterations)
                buf[bp++] = (short)count;
            else
                buf[bp++] = 0;
        }
    }

    public static void main(String[] args) {
        double xl,xh,xs, yl,yh,ys, r;
        int ix, iterations, size;
        short buf[];

        xl = -0.74526593488600;
        yl = 0.11303858131900;
        xh = -0.74525997120900;
        yh = 0.11304454499600;
        r = 4.0;

        size = Integer.parseInt(args[0]);
        iterations =
Integer.parseInt(args[1]);
        System.out.println("size = " + size
+ ", iterations = " + iterations);
        buf = new short[size];
        xs = (xh - xl)/(size - 1);
        ys  = (yh - yl)/(size - 1);

        for (ix = 0; ix < size; ix++)
            mb_d(xl,xs, yl,ys, ix, 0,size-1,
r,iterations, buf);
```

```
//          for (ix = 0; ix < size; ix++)
//              System.out.println(buf[ix] + "
");
    }
```

```
}
```

Python version

```
"""
  Python version - Mandelbrot benchmark by
Bill Wood

"""



def mb_d(xl, xs, yl, ys, ix, iyl, iyh, r,
iterations, buf):

    bp = 0
    for iy in xrange(iyl, iyh):
        cx = xl + ix * xs
        ci = yl + iy * ys
        rx2 = ri2 = rx = ri = 0
        count = 0
        while ((rx2 + ri2) <= r) and (count
< iterations):
            ri = (rx + rx) * ri + ci
            rx = rx2 - ri2 + cx
            rx2 = rx * rx
            ri2 = ri * ri
            count += 1
        if (rx2 + ri2 > r) and (count <=
iterations):
```

```
            buf[bp] = count
            bp += 1
        else:
            buf[bp] = 0
            bp += 1

import sys, time

def Main():

    xl = -0.74526593488600
    yl = 0.11303858131900
    xh = -0.74525997120900
    yh = 0.11304454499600
    r = 4.0

    size = int(sys.argv[1])
    iterations = int(sys.argv[2])
    print "size = ", size, ", iterations =
", iterations

    buf = range(0, size)
    xs = (xh - xl) / (size - 1)
    ys = (yh - yl) / (size - 1)

    starttime = time.clock()
    for ix in xrange(0, size):
        mb_d(xl, xs, yl, ys, ix, 0, size, r,
iterations, buf)
    print "Total time:    ", time.clock() -
starttime
```

```
#     print buf
```

```
Main()
```

C++ version

```
#include "stdafx.h"

/*
 *  C++ version - Mandelbrot benchmark by
Bill Wood
 */

void mb_d(double xl, double xs, double yl,
double ys, int ix, int iyl, int iyh, double
r, int iterations,

short int *buf)
{
  double cx,ci, rx,rx2,ri,ri2;
  int iy, count;

  for (iy = iyl; iy <= iyh; iy++) {
    cx = xl + ix*xs;
    ci = yl + iy*ys;
    rx2 = ri2 = rx = ri = 0;
    for (count = 0; rx2 + ri2 <= r && count
< iterations; count++) {
      ri = (rx+rx)*ri + ci;
      rx = rx2 - ri2 + cx;
      rx2 = rx*rx;
```

```
      ri2 = ri*ri;
      }
    if (rx2 + ri2 > r && count <=
iterations)
      *buf++ = count;
    else
      *buf++ = 0;
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
  double xl,xh,xs, yl,yh,ys, r;
  int ix, iterations, size;
  short int *buf;

  xl = -0.74526593488600;
  yl = 0.11303858131900;
  xh = -0.74525997120900;
  yh = 0.11304454499600;
  r = 4.0;

  size = atoi(argv[1]);
  iterations = atoi(argv[2]);
  printf("size = %d, iterations = %d\n",
size, iterations);
  buf = (short int
*)malloc(size*sizeof(short int));
  xs = (xh - xl)/(size - 1);
  ys  = (yh - yl)/(size - 1);
```

```
  for (ix = 0; ix < size; ix++)
    mb_d(xl,xs, yl,ys, ix, 0,size-1,
r,iterations, buf);
//  for (ix = 0; ix < size; ix++)
```

```
//    printf("%d ", buf[ix]);
}
```