Subject: speed comparison of IDL, numPy, Matlab
Posted by Benyang Tang on Mon, 05 Feb 2001 18:18:04 GMT
View Forum Message <> Reply to Message

Out of curiosity, I did a quick benchmark test of IDL, NumPy and Matlab on my desktop machine. I know benchmarking is a complicated issue; don't take my naive test too serious.

==========
Conditions of the test:
==========

* Machine: a dual Intel Xeon 550 MHz box with 1GB ram, running RedHat Linux 6.2. The machine was not doing any serious service, so the test code should have had close to 100% of the resources.

* IDL, version 5.3

* Python version 1.5.2; NumPy release 15.3-1

* Matlab version 5.3

* For each code, I ran it several times so the timing became somewhat stable. I just took the last reading; for the size of 600X600, timing fluctuation of less than 2% was observed.

* All tests were done in a period of 20 minutes; within this period the following data were collected. About 2 hours later, I redid the tests and obtained similar results.

========== =====
Timing (in seconds) of matrix multiplication of 2 m-by-m matrixes
========== =====

m 100 200 300 400 500 600
---
single precision
IDL 0.02 0.12 0.54 1.83 4.06 6.90
NumPy 0.01 0.12 0.46 1.37 3.05 5.49
Matlab (1)

double precision
IDL 0.04 0.41 1.89 5.14 10.72 18.61
NumPy 0.02 0.16 0.97 2.50 5.05 8.75
Matlab 0.01 0.15 0.94 2.50 4.90 8.69
---
m 100 200 300 400 500 600

(1) Matlab does not do single-precision calculation.

==========
What the tests tell:
==========
1) NumPy is about as fast as Matlab;
2) NumPy is about 25% faster (in single precision), or more than 100% faster
(in double precision) than IDL.

==========
Here are the codes (for the double precision test):
==========

IDL:
===
```
for m = 100,600,100 do begin
  a = double(randomn(0,m,m))
  b = double(randomn(0,m,m))

  time0 = systime(1)
  c = a##b
  dtime = systime(1)-time0

  print, string(m,m,dtime, format='("multiplication of " ,i3, "X", i3, " matrixes takes ", f5.2)')
endfor
```

NumPy:
=====
```
from Numeric import *
from RandomArray import *
import time

for m in [100,200,300,400,500,600]:
  a= uniform(-1,1,(m,m,))
  b= uniform(-1,1,(m,m,))
  a=a.astype(Float64)
  b=b.astype(Float64)

  time0 = time.time()
  c = matrixmultiply(a,b)
  dtime = time.time() - time0

  print 'multiplication of %dX%d matrixes takes %5.2f' %(m,m,dtime)
```

Matlab:
======
```
for m = [100 200 300 400 500 600];
```

```
  a = randn(m);
  b = randn(m);

  time0 = clock;
  c = a*b;
  dtime = etime(clock,time0);

  fprintf('multiplication of %dX%d matrixes takes %5.2f\n', m,m,dtime)
end
```

--
<> Benyang Tang
<> 300-323, JPL
<> 4800 Oak Grove Drive
<> Pasadena, CA 91109, USA

Subject: Re: speed comparison of IDL, numPy, Matlab
Posted by Gavrie on Mon, 12 Feb 2001 12:19:48 GMT
View Forum Message <> Reply to Message

In article <3A7EEE5C.6419B707@pacific.jpl.nasa.gov>,
  Benyang Tang <btang@pacific.jpl.nasa.gov> wrote:

[...]

> * Machine: a dual Intel Xeon 550 MHz box with 1GB ram, running RedHat Linux
> 6.2. The machine was not doing any serious service, so the test code should
> have had close to 100% of the resources.

Well, I don't know how you got these results on a
dual Xeon 550 with 1G of RAM.
I ran the same benchmarks on a dual PIII-550 with
256MB of RAM, and got:

> python:
multiplication of 100X100 matrixes takes 0.02
multiplication of 200X200 matrixes takes 0.20
multiplication of 300X300 matrixes takes 1.07
multiplication of 400X400 matrixes takes 2.81
multiplication of 500X500 matrixes takes 5.49
multiplication of 600X600 matrixes takes 9.58

> matlab:

multiplication of 100X100 matrixes takes 0.01

multiplication of 200X200 matrixes takes 0.06
multiplication of 300X300 matrixes takes 0.15
multiplication of 400X400 matrixes takes 0.35
multiplication of 500X500 matrixes takes 0.68
multiplication of 600X600 matrixes takes 1.19

So, this difference seems a bit strange, doesn't
it?
I'm using MATLAB 6 (whichis supposed to be
slower than 5.3), and Python 1.5.2.

-- Gavrie Philipson.

Sent via Deja.com
http://www.deja.com/

Subject: Re: speed comparison of IDL, numPy, Matlab
Posted by Christopher Lee on Mon, 12 Feb 2001 15:14:04 GMT
View Forum Message <> Reply to Message

>>>> > "Gavrie" == Gavrie <gavrie@my-deja.com> writes:

    Gavrie> In article <3A7EEE5C.6419B707@pacific.jpl.nasa.gov>, Benyang
    Gavrie> Tang <btang@pacific.jpl.nasa.gov> wrote:

    Gavrie> So, this difference seems a bit strange, doesn't it? I'm using
    Gavrie> MATLAB 6 (whichis supposed to be *slower* than 5.3), and Python
    Gavrie> 1.5.2.

I believe there's an explaination for this:

While I've heard that Matlab 6's footprint is much larger, especially the interface, I have heard that Matlab 6 does (finally) use LAPACK and BLAS which are optimized for modern machines w/ hierarchical memory. I would therefore expect that it would do substantially better than the out-of-the-box numpy build for matrix multiplication. By default, numpy uses a naive algorithm for matrix multiplication which is quite cache unfriendly. When numpy is linked to ATLAS's BLAS routines and LAPACK, it's more cache-friendly---and much faster.

I did some benchmarks myself: For matrix inversion of a 1000x1000 matrix, numpy-atlas is 7 times faster than matlab 5.3 (no lapack). The difference is greater if you have a dual processor machine because ATLAS now has options for multi-threaded operation. And for matrix multiplication you can compare the naive numpy multiplication with the optmized-dgemm based

version on the same matrix:

1000 x 1000 matrix
time elapsed for matrixmultiply (sec) 24.142714
time elapsed for dgemm-matrixmultiply (sec) 5.997188

-chris

[BTW: I'm working on putting together an optimized-numpy distribution that uses ATLAS by default.]
