Subject: Re: Python for your applications
Posted by Sender Ghost on Sun, 04 Jan 2009 22:05:09 GMT
View Forum Message <> Reply to Message

Hello, Andrei.

I tested your test case on Windows XP 32 bit. My results are:
1. For 10000000
MSVC++ v9.0 SP1: 16 msec
Python v2.6.1: 3 sec, 156 msec
Python v3.0: 3 sec, 344 msec
2. For 100000000
MSVC++ v9.0 SP1: 47 msec
Python v2.6.1: 31 sec, 828 msec
Python v3.0: 33 sec, 578 msec
3. For 1000000000
MSVC++ v9.0 SP1: 422 msec
Python v2.6.1: MemoryError
Python v3.0: 5 min, 25 sec, 781 msec.

Memory consumption of Python v2.6.1 are large (about 1.5 Gb of 3.25 Gb accessible memory for this test case). Python v3.0 - about 268 Kb.

Because in Windows I haven't the time program for benchmarking I created following gettime program

```
// gettime.cpp
#include <Core/Core.h>

using namespace Upp;

String FormatElapsedTime(double run)
{
 String rtime;
 int msec = int(run) % 1000;
 run /= 1000;
 double hrs = floor(run / 3600);

 if (hrs > 0) rtime << NFormat("%0n hours, ", hrs);

 int minsec = fround(run - 3600 * hrs);
 int min = minsec / 60, sec = minsec % 60;

 if (min || hrs) rtime << min << " min, ";
 if (sec || min) rtime << sec << " sec, ";

 rtime << msec << " msec";
 return rtime;
```

```
}

CONSOLE_APP_MAIN
{
 const Vector<String>& cmdline = CommandLine();

 if (cmdline.GetCount() == 0)
 {
  Cout() << "Not enough command line arguments\n";
  exit(0);
 }

 StringBuffer sb;

 for (int i = 0, n = cmdline.GetCount(); i < n; ++i)
 {
  sb << cmdline[i];
  if (i < n - 1) sb << " ";
 }

 String output;
 TimeStop ts;
 int exitCode = Sys(String(sb), output);
 double elapsed = ts.Elapsed();

 Cout() << "Elapsed: " << FormatElapsedTime(elapsed) << EOL
  << "Exit code: " << AsString(exitCode) << EOL;

 if (!output.IsEmpty()) Cout() << "Output:\n" << output;
}
```

Then I compiled test case python script to optimized bytecode using following batch file (also with benchmarking):

```
rem benchmark.bat
SETLOCAL
SET Python26=C:\Python26
SET Python30=C:\Python30
SET DirectoryToCompile=%~p0

gettime cpptest

%Python26%\python -O %Python26%\Lib\compileall.py %DirectoryToCompile%
gettime %Python26%\python pytest.pyo
del *.pyo

%Python30%\python -O %Python30%\Lib\compileall.py %DirectoryToCompile%
```

```
gettime %Python30%\python pytest.pyo
del *.pyo
```

You can try Lua programming language. I created similar test case:

```
-- luatest.lua
function fun(x)
 return x + 1
end

for x = 0, 1000000000 do
 y = fun(x)
end
```

My results are:
1. For 10000000
Lua v5.1.4: 1 sec, 171 msec
2. For 100000000
Lua v5.1.4: 11 sec, 469 msec
3. For 1000000000
Lua v5.1.4: 1 min, 54 sec, 391 msec.

And memory consumption about 112 Kb.

Also for comparison purposes I created test cases for PHP

```
<?php
// phptest.php
function fun($x)
{
 return $x + 1;
}

for ($x = 0; $x < 1000000000; ++$x)
{
 $y = fun($x);
}
?>
```

and Perl:

```
# perltest.pl
sub fun
{
 my $x = shift;
 return $x + 1;
}
```

```
for my $x (0 .. 1000000000)
{
 my $y = fun($x);
}
```

with:

```
gettime php -n -f phptest.php
gettime perl perltest.pl
```

My results are:
1. For 10000000
PHP v5.2.8: 8 sec, 656 msec
Perl v5.10.0: 6 sec, 187 msec
2. For 100000000
PHP v5.2.8: 1 min, 15 sec, 78 msec
Perl v5.10.0: 1 min, 1 sec, 422 msec
3. For 1000000000
PHP v5.2.8: 12 min, 37 sec, 172 msec
Perl v5.10.0: 10 min, 14 sec, 157 msec.

Memory consumption:
PHP v5.2.8: 388 Kb
Perl v5.10.0: 268 Kb.

Added:
The correct cpptest.cpp file are follows:

```
int fun(int x)
{
 return x + 1;
}

int main()
{
 int y;

 for(int x = 0; x < 1000000000; ++x)
 {
  y = fun(x);
 }

 return y;
}
```

And what about C++ (or JavaScript) interpreter embedded into your program? Examples: Ch, CINT C/C++ Interpreter.

Added:
1. For 10000000
Cint v5.16.19: 5 sec, 703 msec
2. For 100000000
Cint v5.16.19: 47 sec, 562 msec
3. For 1000000000
Cint v5.16.19: 7 min, 44 sec, 719 msec.

Memory consumption of Cint v5.16.19: 260 Kb.

Updated and added:
Result table:

#	Name	10000000	100000000	1000000000
1.	MSVC++ v9.0 SP1	16 msec	47 msec	422 msec
2.	GNU GCC v4.2.4 TDM-1	31 msec	31 msec	31 msec
3.	Python v2.6.1	3 sec, 156 msec	31 sec, 828 msec	MemoryError
4.	Python v2.6.1 with xrange	3 sec, 860 msec	28 sec, 938 msec	4 min, 38 sec, 157 msec
5.	Python v3.0	3 sec, 344 msec	33 sec, 578 msec	5 min, 25 sec, 781 msec
6.	Lua v5.1.4	1 sec, 171 msec	11 sec, 469 msec	1 min, 54 sec, 391 msec
7.	PHP v5.2.8	8 sec, 656 msec	1 min, 15 sec, 78 msec	12 min, 37 sec, 172 msec
8.	Perl v5.10.0	6 sec, 187 msec	1 min, 1 sec, 422 msec	10 min, 14 sec, 157 msec
9.	Cint v5.16.19	5 sec, 703 msec	47 sec, 562 msec	7 min, 44 sec, 719 msec