Runtime Performance evaluation of programming language

[leili@cs.berkeley.edu](mailto:leili@cs.berkeley.edu)

3/28/2012

Contents

[1. exact computing for Urn-Ball model 1](#_Toc320707737)

[2. Comparison of matrix multiplication 1](#_Toc320707738)

[3. Result from http://shootout.alioth.debian.org/ 5](#_Toc320707739)

[4. A result from a recent paper at scala day 2011: 6](#_Toc320707740)

# exact computing for Urn-Ball model

java with BigFraction: openjdk 6.20, with Apache commons math2.2

java7 with BigFraction: oracle jdk 7\_03, with apache commons math 2.2

java7 with math3: oracle jdk 7\_03, with apache commons math 3.0

java 7 with double: oracle jdk 7\_03, with native double

cpp with GMP: g++ 4.4.3, plain flag, with GMP (GNU multiple precision arithmetic library 5.0.4)

cpp with double: g++ 4.4.3, plain flag, with double



# Comparison of matrix multiplication

In this experiment, I will test the wall clock time of computing a multiplication of a 2000 by 2000 matrix with a 2000 vector.

Java is much faster than plain g++ compiler, comparable with O2 optimization, and only tiny slower than O3.

=======Result=========

+ javac jmatrix.java

+ java jmatrix

java allsum=1.8658666E16

real 15.03

user 17.30

sys 8.69

+ g++ cmatrix.cpp -o cmatrix

+ ./cmatrix

c++ allsum=1.86587e+16

real 57.28

user 57.25

sys 0.02

+ g++ -O2 cmatrix.cpp -o cmatrix2

+ ./cmatrix2

c++ allsum=1.86587e+16

real 15.34

user 15.33

sys 0.02

+ g++ -O3 cmatrix.cpp -o cmatrix3

+ ./cmatrix3

c++ allsum=1.86587e+16

real 14.86

user 14.85

sys 0.02

======Specification of the machine===============

ubuntu 10.04 64 bit Linux 2.6.32-37-generic

g++ (Ubuntu 4.4.3-4ubuntu5) 4.4.3

OpenJDK Runtime Environment (IcedTea6 1.9.10) (6b20-1.9.10-0ubuntu1~10.04.2)

OpenJDK 64-Bit Server VM (build 19.0-b09, mixed mode)

6 CPU

8G memory

========jmatrix.java=========

public class jmatrix {

final static int size=2000;

public static void main(String argv[]){

double x[];

double y[];

double m[][];

int i,j,k;

double sum;

double allsum;

try{

m=new double[size][size];

y=new double[size];

x=new double[size];

for(i=0;i<size;i++){

for(j=0;j<size;j++){

m[i][j]=i+j;

}

x[i]=i;

y[i]=0.0;

}

allsum=0.0;

for(k=0;k<size;k++){

for(i=0;i<size;i++){

sum = 0.0;

for(j=0;j<size;j++){

sum+=m[i][j]\*x[j]+k;

}

y[i]=sum;

allsum+=sum;

}

}

}finally{

//delete[] m;

m=null;

x=null;

y=null;

System.gc();

}

System.out.println("java allsum="+allsum);

}

}

========cmatrix.cpp============

#include <iostream>

using namespace std;

#define size 2000

int main(int argc, char\*\* argv){

double \*x,\*y;

double (\*m)[size];

int i,j,k;

double sum,allsum;

m=new double[size][size];

y=new double[size];

x=new double[size];

for(i=0;i<size;i++){

for(j=0;j<size;j++){

m[i][j]=i+j;

}

x[i]=i;

y[i]=0.0;

}

allsum=0.0;

for(k=0;k<size;k++){

for(i=0;i<size;i++){

sum = 0.0;

for(j=0;j<size;j++){

sum+=m[i][j]\*x[j]+k;

}

y[i]=sum;

allsum+=sum;

}

}

delete[] m;

delete[] x;

delete[] y;

cout<<"c++ allsum="<<allsum<<endl;

}

# Result from http://shootout.alioth.debian.org/



Benchmark:

Fasta, binary-trees, Mandelbrot, n-body, fasta-redux, fannkuch-redux, pidigits, spectral-norm, reverse-complement, k-nucleotide, regex-dna

# A result from a recent paper at scala day 2011:

