



Applied Intelligence
765 Westwood Drive
Suite 10A
Clayton, MO 63105
(314) 863-4770

Case Study: A Compute-Bound Task in C++ and Java

An Applied Intelligence White Paper

May 24, 2000

Charles R. Calkins

© Applied Intelligence, Inc., 2000

The trademarks and registered trademarks of the corporations mentioned in this publication are the property of their respective holders. Unless otherwise noted, the entire contents of this publication are copyrighted by Applied Intelligence, Inc. and may not be reproduced, stored in a retrieval system, or transmitted in any form or by any means without prior written consent of the publisher.

Abstract

This white paper compares a C++ implementation of a Mandelbrot Set image generator to a functionally equivalent Java implementation under both Windows NT and Linux. With respect to the C++ version, a significant performance reduction in the Java implementation on both platforms is demonstrated.

Keywords

- C++
- Java
- computation
- Windows NT
- Linux

Table of Contents

Abstract	i
Keywords.....	i
Executive Summary	1
Introduction	1
The Benchmark.....	1
Windows NT – C++ - Microsoft Visual C++.....	2
Linux – C++ - EGCS.....	2
Windows NT – Java - Sun Microsystems JDK.....	2
Windows NT – Java - Microsoft SDK for Java.....	3
Windows NT – Java – Microsoft Visual J++ 6.0.....	4
Windows NT – Java – IBM Developer Kit for Windows, Java Technology Edition.....	4
Linux – Java - Sun Microsystems JDK.....	5
Linux – Java - Blackdown JDK.....	5
Linux – Java – Kaffe Virtual Machine	6
Linux – Java – IBM Developer Kit for Windows, Java Technology Edition	6
Summary	8
Sources.....	9
Listing 1	10
Listing 2.....	12
Listing 3.....	14

Executive Summary

While Java can be used to solve many problems, it is not suitable for compute-bound tasks as the interpreted nature of the language imposes a severe reduction in performance as compared with a compiled language such as C++.

Introduction

Java has emerged over the last few years as a popular alternative to languages such as C++. The biggest advantage of Java is portability – while code written in C++ must be modified to compile on a new target platform, Java code is unchanged. Java achieves this by not being compiled into a native form for the CPU, but instead to a bytecode which is executed by an interpreter. Provided a compliant Java interpreter (“virtual machine”) exists for a platform, the Java code will execute.

To improve performance, a Just-In-Time compiler can be used. As bytecode sequences are examined by a virtual machine, they can be forwarded to a JIT compiler which converts them to native code before execution. Not all virtual machines have JIT compilers included with them, however, so an application’s performance may vary from VM to VM.

The other major difference between virtual machines that directly affects multithreaded performance is whether the VM simulates threads in user space, known as “green threads”, or whether operating system level threads, known as “native threads” are used. The Java application will behave as if it is executing with multiple threads if green threads are used, but no performance will be gained on machines with more than one CPU. Only VMs that support native threads will see the benefit.

Although Java can be used to create solutions to many problems, the fact that Java is interpreted places a penalty on the execution time of the Java application. While many applications are I/O bound waiting for user input, network or other activity outside of the processor, certain applications are instead compute-bound where use of the CPU is paramount. If Java is to be used to implement solutions to problems of this class, it is important to determine the level of performance loss due to the bytecode interpretation. If a JIT compiler is used, the compilation occurs during runtime so an application does suffer a performance hit when the compilation is in progress, but any subsequent executions of the code during the run execute at native speeds.

An example of a compute-bound task is the calculation of the image of the Mandelbrot Set, essentially the the recurrence, repeated several millions of times over the entire image:

$$z_{n+1}=z_n^2+c \text{ where } z=x+iy$$

This computation is also highly parallelizable as each pixel computes this recurrence independently of every other pixel. For the calculation the CPU is used exclusively and no I/O is needed until the final image is to be saved, making it a useful application to compare its performance as implemented in C++ and Java.

The Benchmark

C++ and Java versions of an application to calculate the Mandelbrot Set were developed and executed on a dual 450MHz Pentium II with 256MB of RAM under various conditions as described below. Command-line parameters to this application allow the range of the set to calculate to be specified, the resulting image size, the number of threads to use during the execution, and whether or not to write out the results to a data file.

Timings were collected by running this application over the full set ((-2.25,-1.5) to (0.75,1.5)) using different image sizes and thread counts. The "noio" option was used to allow the timings to reflect the computation alone without the added delay of disk I/O.

Listing 1 is the application as written in C++ for the Win32 API under Windows NT. The most system-dependent part of this code is the thread creation - Win32 threads are used. Timings were collected from a Release build as compiled by Visual C++ 6.0 SP3.

Listing 2 is a version in C++ that is written for Linux. Calls to the *pthread*s thread library replace Win32 threads, but the application is otherwise unchanged. Timings were collected from a build compiled by egcs 2.91.66 with the O6 optimization level set.

Listing 3 is a corresponding version in Java. Java language keywords replace system library calls that make it able to execute unchanged from VM to VM and from platform to platform.

Windows NT 4.0 SP6a was used for the Windows NT benchmark runs, and Linux 2.2.15 compiled for SMP was used for the Linux benchmark runs.

Windows NT - C++ - Microsoft Visual C++

Timings under NT are consistent with expectations - as the image size doubles the time increases by a factor of four (as the problem size has now quadrupled), and when two threads are used the time spent is half that of one thread.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	0.218	256x256	0.078
512x512	0.640	512x512	0.328
1024x1024	2.562	1024x1024	1.296
2048x2048	10.281	2048x2048	5.156
4096x4096	41.062	4096x4096	20.625

Linux - C++ - EGCS

Even though it's a different operating system, the timings as compared to Windows NT are nearly identical. Its not surprising, as it is pure calculation - no operating system calls needed once the threads are executing, and only differences in the system scheduler would impact performance.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	0.160	256x256	0.080
512x512	0.640	512x512	0.320
1024x1024	2.580	1024x1024	1.290
2048x2048	10.350	2048x2048	5.180
4096x4096	41.410	4096x4096	20.720

Windows NT - Java - Sun Microsystems JDK

The Sun Microsystems reference implementation of Java for NT - the Java 2 Development Kit 1.3.0 - was used for this test. Sun's implementation for NT supports native Win32

threads, but by default their VM does not use a JIT compiler; instead it uses Sun's own HotSpot technology, their proprietary optimizing compiler.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	1.281	256x256	0.656
512x512	3.218	512x512	1.765
1024x1024	11.968	1024x1024	6.078
2048x2048	47.015	2048x2048	23.375
4096x4096	187.296	4096x4096	92.515

The VM can also be invoked with the "-classic" flag, disabling the HotSpot compiler and allowing 3rd party JIT compilers to be used. If none are present, as in this test, the bytecode is interpreted, with a significant performance degradation.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	5.156	256x256	2.562
512x512	19.765	512x512	9.546
1024x1024	78.406	1024x1024	37.515
2048x2048	313.328	2048x2048	149.468
4096x4096	1253.050	4096x4096	597.203

Windows NT – Java - Microsoft SDK for Java

Microsoft has a free Java implementation for NT – the Microsoft SDK for Java 4.0. This implementation includes a JIT compiler and supports and native Win32 threads.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	0.921	256x256	0.531
512x512	3.093	512x512	1.609
1024x1024	11.812	1024x1024	5.937
2048x2048	46.734	2048x2048	23.218
4096x4096	186.312	4096x4096	92.406

The timings using the Microsoft SDK are fractionally better than the Sun JDK, but are nearly identical – a corresponding performance decrease is still present.

Windows NT - Java - Microsoft Visual J++ 6.0

Visual J++ provides a development environment including an editor, debugger and project management features, though it is still the Microsoft Java implementation underneath. As such, it supports native threads, has a JIT compiler, and performance is comparable to the free SDK.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	0.953	256x256	0.578
512x512	3.125	512x512	1.671
1024x1024	11.875	1024x1024	6.078
2048x2048	46.890	2048x2048	23.671
4096x4096	186.921	4096x4096	94.109

Windows NT - Java - IBM Developer Kit for Windows, Java Technology Edition

IBM's Java developer kit version 1.1.8 supports native threads as well as a JIT compiler.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	4.265	256x256	0.781
512x512	15.437	512x512	1.640
1024x1024	60.296	1024x1024	5.046
2048x2048	238.046	2048x2048	12.000
4096x4096	951.125	4096x4096	46.578

It appears that the JIT compiler is not invoked when only one thread in the application is used - the timings for a single are repeatable and consistent with the disabling of the JIT compiler. The compiler is apparently used for two threads however, and the performance is dramatically improved.

The JIT compiler can be disabled with the "-nojit" flag, in a manner similar to Sun's "classic" mode.

One Thread		One Thread	
Size	Time (s)	Size	Time (s)
256x256	4.031	256x256	2.156
512x512	15.109	512x512	7.750
1024x1024	59.625	1024x1024	30.125
2048x2048	237.859	2048x2048	119.796
4096x4096	951.265	4096x4096	478.234

Linux - Java - Sun Microsystems JDK

Sun also has a reference implementation of Java for Linux, though the release at the time of writing is 1.2.2, slightly older than the Windows NT version. Sun's Linux implementation doesn't use native threads - timings for the one thread and two thread cases are identical, so only the one thread case is presented.

One Thread	
Size	Time (s)
256x256	5.030
512x512	19.210
1024x1024	75.930
2048x2048	303.130
4096x4096	1212.110

A note in the documentation included in the distribution stated that using the JDK on an SMP kernel was not supported, but running the same test on 2.2.15 compiled for only a single processor produced results to within a few tenths of a second of the SMP timings.

Linux - Java - Blackdown JDK

Blackdown's Java tools are a port of the Sun JDK to Linux. Version 1.2.2 RC4, although comparable in performance to Sun's in the single threaded case, does support native system threads and is thus faster in the multithreaded case.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	5.470	256x256	2.840
512x512	20.630	512x512	10.110
1024x1024	74.640	1024x1024	39.600
2048x2048	297.620	2048x2048	147.970
4096x4096	1189.790	4096x4096	659.720

Linux - Java - Kaffe Virtual Machine

Kaffe is a virtual machine developed in a “cleanroom” environment – entirely from scratch and without proprietary information from Sun Microsystems. It is only a virtual machine so Java source code must be compiled with a compiler from a complete development environment (Sun’s was used in this case). It includes a JIT compiler but does not use native threads.

One Thread	
Size	Time (s)
256x256	2.720
512x512	9.710
1024x1024	37.660
2048x2048	149.640
4096x4096	597.660

Linux - Java - IBM Developer Kit for Windows, Java Technology Edition

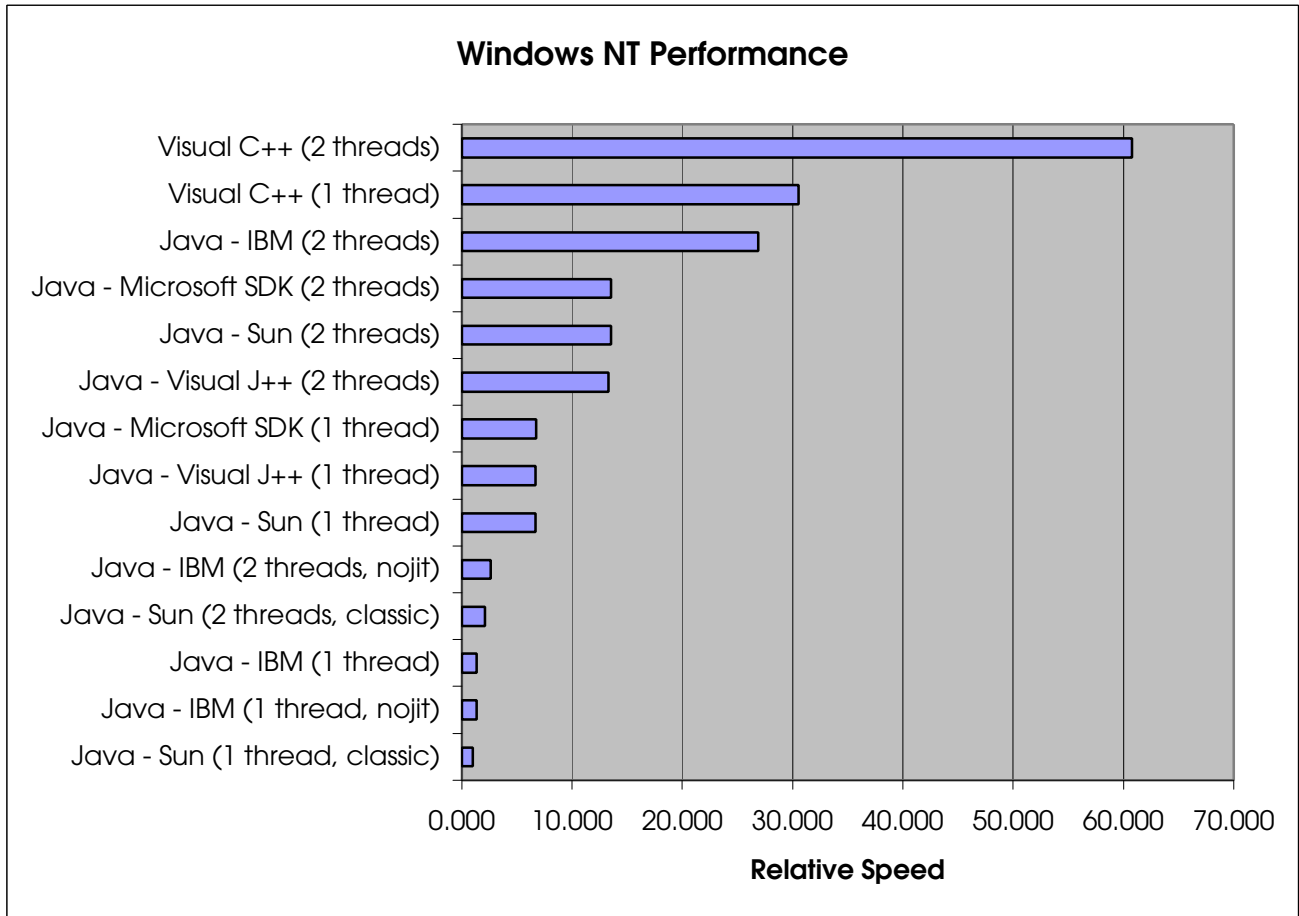
IBM’s Java developer kit version 1.1.8 supports native threads as well as a JIT compiler under Linux.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	0.540	256x256	0.480
512x512	1.090	512x512	0.750
1024x1024	3.280	1024x1024	1.850
2048x2048	12.010	2048x2048	6.220
4096x4096	46.960	4096x4096	23.800

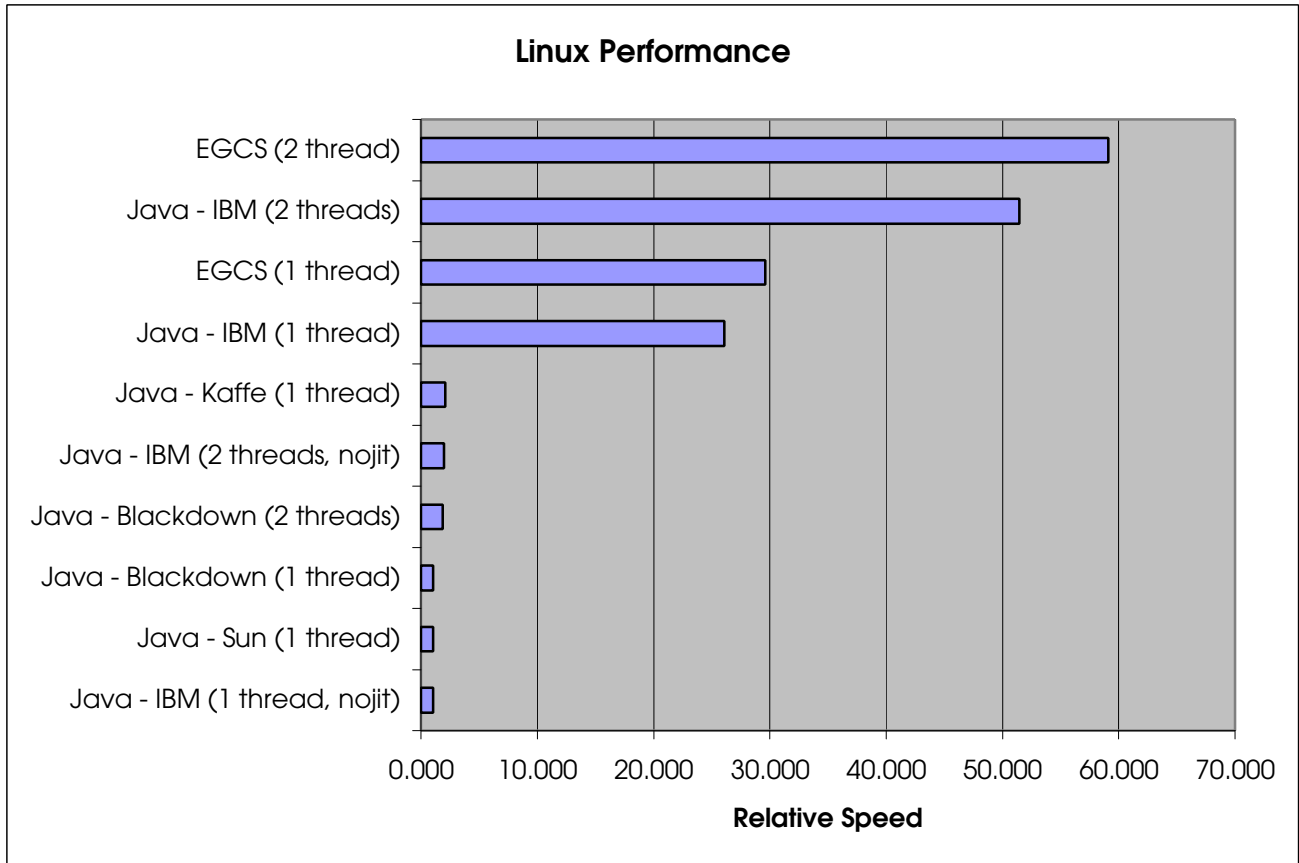
Timings under Linux indicate the JIT compiler is being used in the single threaded case.

As under Windows NT, the JIT compiler can also be disabled by using the “-nojit” flag.

One Thread		Two Threads	
Size	Time (s)	Size	Time (s)
256x256	4.90	256x256	2.53
512x512	19.23	512x512	9.74
1024x1024	76.57	1024x1024	38.59
2048x2048	306.21	2048x2048	154.12
4096x4096	1225.01	4096x4096	616.33



Under Windows NT, Java performance for this application does not come close to C++. The IBM SDK performs the best with two threads of all of the NT Java VMs examined, but even its multithreaded performance is still worse than single threaded C++.



Under Linux, the IBM Java performance is even better than NT, though it still does not reach the level of C++. All other Java VMs surveyed perform much worse than under NT.

Summary

While Java may have many uses, compute-bound applications are not one of them. Compounding the problem, not all Java virtual machines are equal, providing varying levels of performance. It can be seen from the graphs that C++ is the clear leader, though one of the virtual machines surveyed under Linux had performance approaching C++. It can also be seen that some form of compilation, such as a JIT compiler, is a necessity for improved performance – C++ is as much as 30 times faster than purely interpreted Java. Although pure Java as a language is identical from virtual machine to virtual machine, performance varies widely.

Sources

More information on the Java implementations used above can be found at the following locations:

Microsoft SDK for Java
<http://www.microsoft.com/java/sdk/40/start.htm>

Microsoft Visual J++
<http://msdn.microsoft.com/visualj/>

Sun Microsystems Java reference implementation
<http://java.sun.com>

Blackdown Java
<http://www.blackdown.org>

Kaffe virtual machine
<http://www.kaffe.org>

IBM Java developer kits
<http://www.ibm.com/java/jdk/>

Listing 1:

```
#include <windows.h>
#include <fstream.h>
#include <string.h>

double xmin=-2.25;
double xmax=0.75;
double ymin=-1.5;
double ymax=1.5;
unsigned long maxiter=256;
long size=256;
unsigned long nthreads=1;
char filename[256];
bool noio=false;

unsigned char *array=NULL;
long row=-1;

void WINAPI mandelthread(unsigned long *) {
    while (1) {
        int myrow=InterlockedIncrement(&row);
        if (myrow>=size) return;

        double cy=ymin+myrow*(ymax-ymin)/(size-1);
        for (int ix=0; ix<size; ix++) {
            double cx=xmin+ix*(xmax-xmin)/(size-1);

            double x=cx,y=cy;
            double x2=x*x,y2=y*y;
            unsigned long iter=0;

            while ((iter<maxiter) && (x2+y2<10000.0)) {
                y=2*x*y+cy;
                x=x2-y2+cx;
                x2=x*x;
                y2=y*y;
                iter++;
            }

            array[myrow*size+ix]=((iter<maxiter)?0:1);
        }
    }
}

void main(int argc, char *argv[]) {
    strcpy(filename, "mandel.dat");

    int i=1;
    while (i<argc) {
        if (!strcmp(argv[i], "-xmin")) {
            i++;
            xmin=atof(argv[i++]);
        } else
        if (!strcmp(argv[i], "-xmax")) {
            i++;
            xmax=atof(argv[i++]);
        } else
        if (!strcmp(argv[i], "-ymin")) {
            i++;
            ymin=atof(argv[i++]);
        } else
        if (!strcmp(argv[i], "-ymax")) {
            i++;
            ymax=atof(argv[i++]);
        } else
        if (!strcmp(argv[i], "-maxiter")) {
            i++;
        }
    }
}
```

```

    maxiter=atol(argv[i++]);
} else
if (!strcmp(argv[i], "--size")) {
    i++;
    size=atol(argv[i++]);
} else
if (!strcmp(argv[i], "-nthreads")) {
    i++;
    nthreads=atol(argv[i++]);
} else
if (!strcmp(argv[i], "--file")) {
    i++;
    strncpy(filename, argv[i++], 256);
} else
if (!strcmp(argv[i], "--noio")) {
    i++;
    noio=true;
} else
    i++;
}

cout << "(C++) Generating " << size << "x" << size << "(" << xmin << "," << ymin << ") to (" <<
    xmax << "," << ymax << ") using " << maxiter << " maximum iterations and " << nthreads <<
    " threads as " << filename << endl;
array=new unsigned char[size*size];

HANDLE *threads=new HANDLE[nthreads];
for (i=0; i<nthreads; i++)
    threads[i]=CreateThread(0,0, (LPTHREAD_START_ROUTINE)mandelthread, 0,0,0);

WaitForMultipleObjects(nthreads, threads, TRUE, INFINITE);

for (i=0; i<nthreads; i++)
    CloseHandle(threads[i]);

if (!noio) {
    ofstream out(filename, ios::binary);
    out.flags(0);
    out.write((char *)&size, 4);
    out.write(array, size*size);
    out.close();
}

delete [] threads;
delete [] array;
}

```

Listing 2:

```
#include <pthread.h>
#include <fstream.h>
#include <string.h>
#include <stdlib.h>

double xmin=-2.25;
double xmax=0.75;
double ymin=-1.5;
double ymax=1.5;
unsigned long maxiter=256;
long size=256;
unsigned long nthreads=1;
char filename[256];
bool noio=false;

unsigned char *array=NULL;
long row=-1;

pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

void *mandelthread(void *) {
    while (1) {
        pthread_mutex_lock(&mutex);
        int myrow=++row;
        pthread_mutex_unlock(&mutex);
        if (myrow>=size) return 0;

        double cy=ymin+myrow*(ymax-ymin)/(size-1);
        for (int ix=0; ix<size; ix++) {
            double cx=xmin+ix*(xmax-xmin)/(size-1);

            double x=cx,y=cy;
            double x2=x*x,y2=y*y;
            unsigned long iter=0;

            while ((iter<maxiter) && (x2+y2<10000.0)) {
                y=2*x*y+cy;
                x=x2-y2+cx;
                x2=x*x;
                y2=y*y;
                iter++;
            }

            array[myrow*size+ix]=(unsigned char)((iter<maxiter)?0:1);
        }
    }
    return 0;
}

void main(int argc, char *argv[]) {
    strcpy(filename, "mandel.dat");

    int i=1;
    while (i<argc) {
        if (!strcmp(argv[i], "-xmin")) {
            i++;
            xmin=atof(argv[i++]);
        } else
        if (!strcmp(argv[i], "-xmax")) {
            i++;
            xmax=atof(argv[i++]);
        } else
        if (!strcmp(argv[i], "-ymin")) {
            i++;
            ymin=atof(argv[i++]);
        } else
    }
```

```

    if (!strcmp(argv[i], "-ymax")) {
        i++;
        ymax=atof(argv[i++]);
    } else
    if (!strcmp(argv[i], "-maxiter")) {
        i++;
        maxiter=atol(argv[i++]);
    } else
    if (!strcmp(argv[i], "-size")) {
        i++;
        size=atol(argv[i++]);
    } else
    if (!strcmp(argv[i], "-nthreads")) {
        i++;
        nthreads=atol(argv[i++]);
    } else
    if (!strcmp(argv[i], "-file")) {
        i++;
        strncpy(filename, argv[i++], 256);
    } else
    if (!strcmp(argv[i], "-noio")) {
        i++;
        noio=true;
    } else
        i++;
}

cout << "(C++) Generating " << size << "x" << size << "(" << xmin << "," << ymin << ") to (" <<
    xmax << "," << ymax << ") using " << maxiter << " maximum iterations and " << nthreads <<
    " threads as " << filename << endl;

array=new unsigned char[size*size];

pthread_t *threads=new pthread_t[nthreads];
for (i=0; i<nthreads; i++)
    pthread_create(&threads[i], 0, mandelthread, 0);

for (i=0; i<nthreads; i++)
    pthread_join(threads[i], NULL);

if (!noio) {
    ofstream out(filename);
    out.flags(0);
    out.write((char *)&size, 4);
    out.write(array, size*size);
    out.close();
}

delete [] threads;
delete [] array;
}

```

Listing 3:

```
import java.io.*;

class Mandel {
    public static void main(String[] args) {
        MandelGenerator mgen = new MandelGenerator(args);
    }

    static class MandelGenerator {
        double xmin=-2.25;
        double xmax=0.75;
        double ymin=-1.5;
        double ymax=1.5;
        int maxiter=256;
        int size=256;
        int nthreads=1;
        String filename="mandel.dat";
        boolean noio=false;

        int row=-1;
        byte[] array;

        class MandelThread extends Thread {
            public void run() {
                while (true) {
                    int myrow=++row;
                    if (myrow>=size) return;

                    double cy=ymin+myrow*(ymax-ymin)/(size-1);
                    for (int ix=0; ix<size; ix++) {
                        double cx=xmin+ix*(xmax-xmin)/(size-1);

                        double x=cx,y=cy;
                        double x2=x*x,y2=y*y;
                        int iter=0;

                        while ((iter<maxiter) && (x2+y2<10000.0)) {
                            y=2*x*y+cy;
                            x=x2-y2+cx;
                            x2=x*x;
                            y2=y*y;
                            iter++;
                        }

                        if (iter<maxiter)
                            array[myrow*size+ix]=0;
                        else
                            array[myrow*size+ix]=1;
                    }
                }
            }
        }

        MandelGenerator(String args[]) {
            int i=0;
            while (i<args.length) {
                if (args[i].equals("-xmin")) {
                    i++;
                    xmin=Double.valueOf(args[i++]).doubleValue();
                } else
                if (args[i].equals("-xmax")) {
                    i++;
                    xmax=Double.valueOf(args[i++]).doubleValue();
                } else
                if (args[i].equals("-ymin")) {
                    i++;
                    ymin=Double.valueOf(args[i++]).doubleValue();
                } else
                if (args[i].equals("-ymax")) {
```

```

        i++;
        ymax=Double.valueOf(args[i++]).doubleValue();
    } else
    if (args[i].equals("-maxiter")) {
        i++;
        maxiter=Integer.valueOf(args[i++]).intValue();
    } else
    if (args[i].equals("-size")) {
        i++;
        size=Integer.valueOf(args[i++]).intValue();
    } else
    if (args[i].equals("-nthreads")) {
        i++;
        nthreads=Integer.valueOf(args[i++]).intValue();
    } else
    if (args[i].equals("-file")) {
        i++;
        filename = args[i++];
    } else
    if (args[i].equals("-noio")) {
        i++;
        noio=true;
    } else
        i++;
    }

System.out.println("(java) Generating " + String.valueOf(size) + "x" + String.valueOf(size)
+ "(" + String.valueOf(xmin) + "," + String.valueOf(ymin) + ") to (" +
String.valueOf(xmax) + "," + String.valueOf(ymax) +
") using " + String.valueOf(maxiter) + " maximum iterations and " +
String.valueOf(nthreads) + " threads as " + filename);

array=new byte[size*size];

// start threads
MandelThread m[]=new MandelThread[nthreads];
for (i=0; i<nthreads; i++) {
    m[i]=new MandelThread();
    m[i].start();
}

try {
    // wait for completion
    for (i=0; i<nthreads; i++)
        m[i].join();
} catch (InterruptedException e) {
}

if (!noio) {
    // write out the data
    try {
        FileOutputStream fstream=new FileOutputStream(filename);
        DataOutputStream out=new DataOutputStream(fstream);
        out.writeByte(size&0xFF);
        out.writeByte(size>>8); // can't use writeInt as it writes out in bigendian
        out.write(array, 0, size*size);
        out.flush();
    } catch (FileNotFoundException e) {
    } catch (IOException e) {
    }
}
}
}
}

```