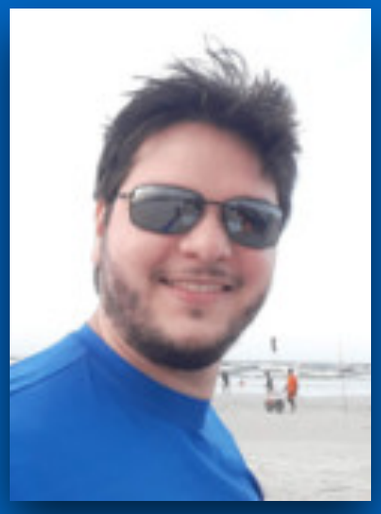
Comprehending Energy Behaviors of Java I/O APIs



Gilson Rocha

Fernando Castor

Gustavo Pinto

esem
2019

@gustavopinto

Uh oh

@gustavopinto


Respect
by battery!
We need
energy
efficient
apps!
Go away
bad
apps!

Waze battery consumption

Moderators: Unholy, Unholy

POSTREPLY Search this topic... Search

16 posts • Page 1 of 2 • 1

Waze battery consumption

QUOTE

by **sfe0** » Sat Jun 02, 2012 10:01 am

Hello all!
Been using Waze for about two weeks now and I really like it. Haven't been stuck in any traffic jam during this time, like I normally do this time of year, but I have received a few warnings from Waze and driven alternative roads with great success. Thank You for that!
But, how come Waze uses so much battery?
I have a SonyEricssonArcS and the SonyEricsson car charger is the only one I've tried this far that can cope with the energy hungry Waze app. All other chargers, 4 of them, can't deliver enough current to keep the phones battery from draining when running Waze. The phone itself gets very hot and from what I've read on the net this isn't a problem just for me.
I also use another similar app in my Xperia and that is the Geocaching app "Neongeo". Neongeo, like Waze, uses GPS, a-GPS, maps, shows me moving around on the map, live internet update and keeps the phones screen always on. Still, it consumes what seems like much less power than Waze and the phone does not get hot at all.
Why?

sfe0

Posts: 4
Joined: Thu May 24, 2012 10:29 am
Has thanked: 0 time
Been thanked: 0 time

Waze battery consumption

Moderators: Unholy, Unholy

POSTREPLY Search this topic... Search

16 posts • Page 1 of 2 • 1

Waze battery consumption

QUOTE

by **sfe0** » Sat Jun 02, 2012 10:01 am

Hello all!
Been using Waze for about two weeks now and I really like it. Haven't been stuck in any traffic jam during this time, like I normally do this time of year, but I have received a few warnings from Waze and driven alternative roads with great success. Thank You for that!
But, how come Waze uses so much battery?
I have a SonyEricssonArcS and the SonyEricsson car charger is the only one I've tried this far that can cope with the energy hungry Waze app. All other chargers, 4 of them, can't deliver enough current to keep the phones battery from draining when running Waze. The phone itself gets very hot and from what I've read on the net this isn't a problem just for me.
I also use another similar app in my Xperia and that is the Geocaching app "Neongeo". Neongeo, like Waze, uses GPS, a-GPS, maps, shows me moving around on the map, live internet update and keeps the phones screen always on. Still, it consumes what seems like much less power than Waze and the phone does not get hot at all.
Why?

sfe0

Posts: 4
Joined: Thu May 24, 2012 10:29 am
Has thanked: 0 time
Been thanked: 0 time

Waze battery consumption

Moderators: Unholy, Unholy

POSTREPLY Search this topic... Search

16 posts • Page 1 of 2 • 1

Waze battery consumption

QUOTE

by **sfe0** » Sat Jun 02, 2012 10:01 am

Hello all!
Been using Waze for about two weeks now and I really like it. Haven't been stuck in any traffic jam during this time, like I normally do this time of year, but I have received a few warnings from Waze and driven alternative roads with great success. Thank You for that!
But, how come Waze uses so much battery?
I have a SonyEricssonArcS and the SonyEricsson car charger is the only one I've tried this far that can cope with the energy hungry Waze app. All other chargers, 4 of them, can't deliver enough current to keep the phones battery from draining when running Waze. The phone itself gets very hot and from what I've read on the net this isn't a problem just for me.
I also use another similar app in my Xperia and that is the Geocaching app "Neongeo". Neongeo, like Waze, uses GPS, a-GPS, maps, shows me moving around on the map, live internet update and keeps the phones screen always on. Still, it consumes what seems like much less power than Waze and the phone does not get hot at all.
Why?

sfe0

Posts: 4
Joined: Thu May 24, 2012 10:29 am
Has thanked: 0 time
Been thanked: 0 time

I have no idea on how to make this code more energy efficient 😢

@gustavopinto

Source of Java I/O APIs

java.io

Class Writer

java.lang.Object
    java.io.Writer

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:

BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

---

```
public abstract class Writer
extends Object
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

java.io

Class Reader

java.lang.Object
    java.io.Reader

All Implemented Interfaces:

Closeable, AutoCloseable, Readable

Direct Known Subclasses:

BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

---

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

java.io

Class InputStream

java.lang.Object
    java.io.InputStream

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

---

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:

JDK1.0

java.io

Class OutputStream

java.lang.Object
    java.io.OutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

---

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:

JDK1.0

@gustavopinto

Source of Java I/O APIs

java.io

Class Writer

java.lang.Object
    java.io.Writer

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:

BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

---

```
public abstract class Writer
extends Object
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

java.io

Class Reader

java.lang.Object
    java.io.Reader

All Implemented Interfaces:

Closeable, AutoCloseable, Readable

Direct Known Subclasses:

BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

---

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

java.io

Class InputStream

java.lang.Object
    java.io.InputStream

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

---

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:

JDK1.0

java.io

Class OutputStream

java.lang.Object
    java.io.OutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

---

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

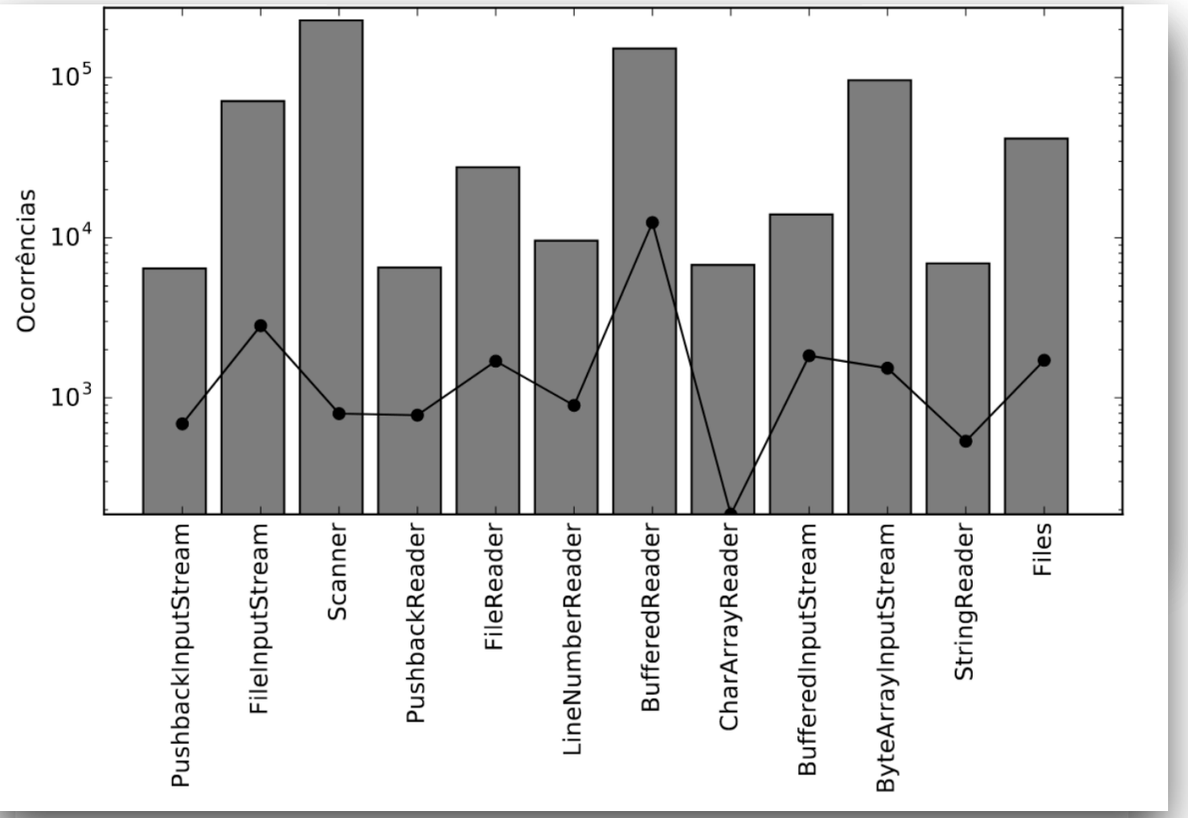
Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:

JDK1.0

@gustavopinto

Source of Java I/O APIs



100K+ projects use Java IO APIs
(as of sept 2015)

java.io

Class Reader

java.lang.Object
    java.io.Reader

All Implemented Interfaces:

Closeable, AutoCloseable, Readable

Direct Known Subclasses:

BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

---

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

java.io

Class OutputStream

java.lang.Object
    java.io.OutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

---

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:

JDK1.0

@gustavopinto

```
BufferedReader reader = new BufferedReader(new
FileReader(“file.txt”));


try {
     StringBuilder sb = new StringBuilder();
     String line = reader.readLine();

     while (line != null) {
          sb.append(line);
          sb.append(System.lineSeparator());
          line = reader.readLine();
     }
     String everything = sb.toString();
} finally {
     br.close();
}
```

```
BufferedReader reader = new BufferedReader(new
FileReader(“file.txt”));


try {
    StringBuilder sb = new StringBuilder();
    String line = reader.readLine();

    while (line != null) {
        sb.append(line);
        sb.append(System.lineSeparator());
        line = reader.readLine();
    }
    String everything = sb.toString();
} finally {
    br.close();
}
```

```
LineNumberReader reader = new LineNumberReader(new
FileReader(“file.txt”));


try {
    StringBuilder sb = new StringBuilder();
    String line = reader.readLine();

    while (line != null) {
        sb.append(line);
        sb.append(System.lineSeparator());
        line = reader.readLine();
    }
    String everything = sb.toString();
} finally {
    br.close();
}
```

```
CharArrayReader reader = new CharArrayReader(new
FileReader(“file.txt”));


try {
    StringBuilder sb = new StringBuilder();
    String line = reader.readLine();

    while (line != null) {
        sb.append(line);
        sb.append(System.lineSeparator());
        line = reader.readLine();
    }
    String everything = sb.toString();
} finally {
    br.close();
}
```

```
FilterReader reader = new FilterReader(new
FileReader(“file.txt”));


try {
    StringBuilder sb = new StringBuilder();
    String line = reader.readLine();

    while (line != null) {
        sb.append(line);
        sb.append(System.lineSeparator());
        line = reader.readLine();
    }
    String everything = sb.toString();
} finally {
    br.close();
}
```

```
BufferedReader reader = new BufferedReader(new
FileReader("file.txt"));

LineNumberReader reader = new LineNumberReader(new
FileReader("file.txt"));

CharArrayReader reader = new CharArrayReader(new
FileReader("file.txt"));

FilterReader reader = new FilterReader(new
FileReader("file.txt"));
```

Similar design choices

Extremely used

```
BufferedReader reader = new BufferedReader(new
FileReader(“file.txt”));

LineNumberReader reader = new LineNumberReader(new
FileReader(“file.txt”));

CharArrayReader reader = new CharArrayReader(new
FileReader(“file.txt”));

FilterReader reader = new FilterReader(new
FileReader(“file.txt”));
```

Similar design choices

Extremely used

Energy usage?

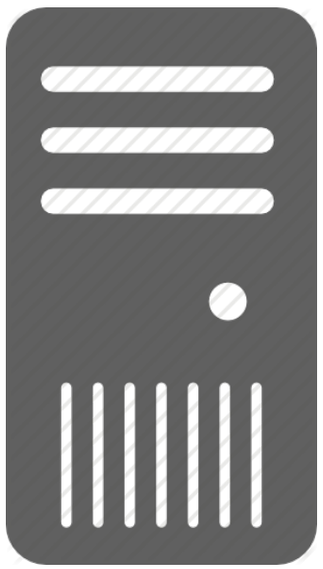
Research Questions

RQ1: What is the energy consumption behavior of the Java I/O APIs?

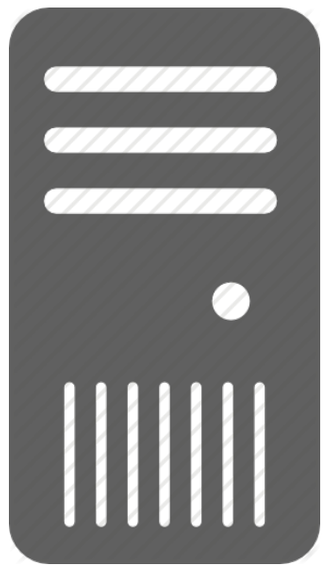
RQ2: Can we improve energy consumption by refactoring the use of Java I/O APIs?

@gustavopinto

2 environments

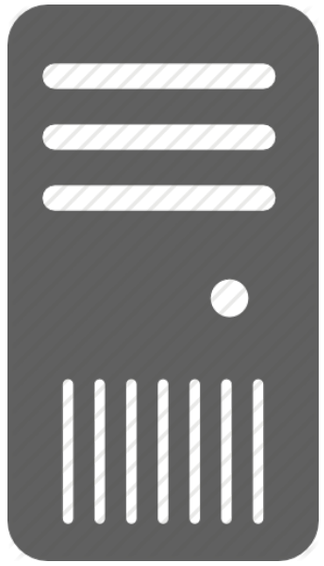


Intel CPU: A 4-core, running Ubuntu, 2.2 GHz, 16GB of memory, JDK version 1.8.0, build 151.



Intel CPU: A 40-core, running Ubuntu, 2.20GHz, with 251GB of memory, JDK version 1.8.0, build 151.

Software-based energy measurement



Intel CPU: A 4-core, running Ubuntu, 2.2 GHz, 16GB of memory, JDK version 1.8.0, build 151.

jRAPL -- A framework for profiling energy consumption of Java programs

What is jRAPL?

jRAPL is framework for profiling Java programs running on CPUs with Running Average Power Limit (RAPL) support.

But, what is RAPL?

RAPL is a set of low-level interfaces with the ability to monitor, control, and get notifications of energy and power consumption data of different hardware levels.

Originally designed by Intel for enabling chip-level power management, RAPL is widely supported in today's Intel architectures, including Xeon server-level CPUs and the popular i5 and i7.

https://github.com/kliu20/jRAPL

K. Liu, G. Pinto, and Y. D. Liu, “Data-oriented characterization of application-level energy optimization,” in Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering, ser. FASE’15, 2015.

22 Java IO APIs

Writer
BufferedWriter
FileWriter
StringWriter
PrintWriter
CharArrayWriter

Reader
BufferedReader
LineNumberReader
CharArrayReader
PushbackReader
FileReader
StringReader

OutputStream
FileOutputStream
ByteArrayOutputStream
BufferedOutputStream
PrintStream

InputStream
FileInputStream
BufferedInputStream
PushbackInputStream
ByteArrayInputStream

Benchmarks

Micro benchmarks

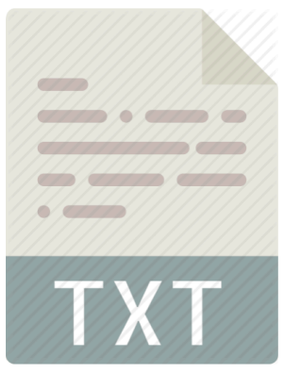
Macro benchmarks

Optimized benchmarks

Micro benchmarks

```
BufferedInputStream reader = new
BufferedInputStream(new
    FileInputStream(FILE_READER));
int value = 0, fake = 0;
while ((value = reader.read()) != -1) fake = value;
reader.close()
```

FILE_READER =

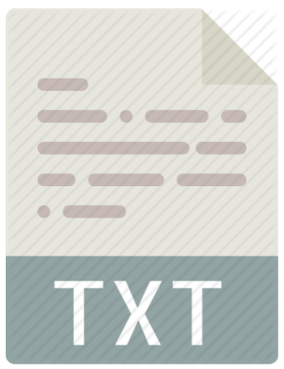


20mb

Micro benchmarks

```
BufferedInputStream reader = new
BufferedInputStream(new
    FileInputStream(FILE_READER));
int value = 0, fake = 0;
while ((value = reader.read()) != -1) fake = value;
reader.close()
```

FILE_READER =

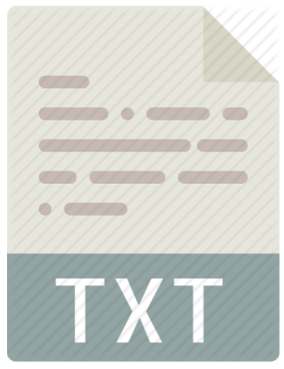


20mb

Micro benchmarks

```
BufferedOutputStream fileWriter = new
BufferedOutputStream(new
    FileOutputStream(new File(OUT_WRITER +
UUID.randomUUID().toString())));
fileWriter.write(data);
fileWriter.close();
```

OUT_WRITER =

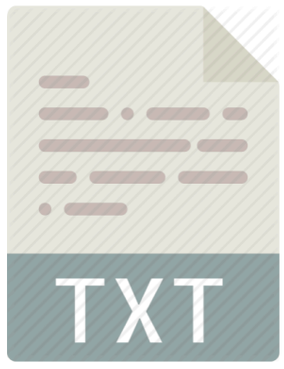


20mb

Micro benchmarks

```
BufferedOutputStream fileWriter = new
BufferedOutputStream(new
    FileOutputStream(new File(OUT_WRITER +
UUID.randomUUID().toString())));
fileWriter.write(data);
fileWriter.close();
```

OUT_WRITER =



20mb

Optimized benchmarks

The Computer Language Benchmarks Game

"Which programming language is fastest?"

Should we care? How could we know?

"It's important to be realistic: most people don't care about program performance most of the time."

"By instrumenting the … runtime, we measure the JavaScript behavior of … web applications… Our results show that real web applications behave very differently from the benchmarks…"

Fasta

K-nucleotide

Reverse-complement

Source code and performance measurements available

```
import java.io.IOException;
import java.io.OutputStream;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.atomic.AtomicInteger;

public class fasta {

    static final int LINE_LENGTH = 60;
    static final int LINE_COUNT = 1024;
    static final NucleotideSelector[] WORKERS
            = new NucleotideSelector[
                    Runtime.getRuntime().availableProcessors() > 1
                    ? Runtime.getRuntime().availableProcessors() - 1
                    : 1];
    static final AtomicInteger IN = new AtomicInteger();
    static final AtomicInteger OUT = new AtomicInteger();
    static final int BUFFERS_IN_PLAY = 6;
    static final int IM = 139968;
    static final int IA = 3877;
    static final int IC = 29573;
    static final float ONE_OVER_IM = 1f / IM;
    static int last = 42;

    public static void main(String[] args) {
        int n = 1000;

        if (args.length > 0) {
            n = Integer.parseInt(args[0]);
        }
        for (int i = 0; i < WORKERS.length; i++) {
            WORKERS[i] = new NucleotideSelector();
            WORKERS[i].setDaemon(true);
            WORKERS[i].start();
        }
        try (OutputStream writer = System.out;) {
            final int bufferSize = LINE_COUNT * LINE_LENGTH;

            for (int i = 0; i < BUFFERS_IN_PLAY; i++) {
                lineFillALU(
                            final byte[] sapienChars = new byte[]{
            'a',
            'c',
            'g',
            't'};
        final double[] sapienProbs = new double[]{
            0.3029549426680,
            0.1979883004921,
            0.1975473066391,
            0.3015094502008};
        final float[] probs;
        final float[] randoms;
        final int charsInFullLines;

        public Buffer(final boolean isIUB
                , final int lineLength
                , final int nChars) {
            super(lineLength, nChars);
            double cp = 0;
            final double[] dblProbs = isIUB ? iubProbs : sapienProbs;

            chars = isIUB ? iubChars : sapienChars;
            probs = new float[dblProbs.length];
            for (int i = 0; i < probs.length; i++) {
                cp += dblProbs[i];
                probs[i] = (float) cp;
            }
            probs[probs.length - 1] = 2f;
            randoms = new float[nChars];
            charsInFullLines = (nChars / lineLength) * lineLength;
        }

        @Override
        public void selectNucleotides() {
            int i, j, m;
            float r;
            int k;

            for (i = 0, j = 0; i < charsInFullLines; j++) {
                for (k = 0; k < LINE_LENGTH; k++) {
                    r = randoms[i++];
                    for (m = 0; probs[m] < r; m++) {
                    }
                    nucleotides[j++] = chars[m];
                }
            }
            for (k = 0; k < CHARS_LEFTOVER; k++) {
                r = randoms[i++];
                for (m = 0; probs[m] < r; m++) {
                }
                nucleotides[j++] = chars[m];
            }
        }
    }
}
```

Fasta (325 loc)

```java
import java.io.IOException;
import java.io.OutputStream;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.atomic.AtomicInteger;

public class fasta {

    static final int LINE_LENGTH = 60;
    static final int LINE_COUNT = 1024;
    static final NucleotideSelector[] WORKERS
            = new NucleotideSelector[
                    Runtime.getRuntime().availableProcessors() > 1
                    ? Runtime.getRuntime().availableProcessors() - 1
                    : 1];
    static final AtomicInteger IN = new AtomicInteger();
    static final AtomicInteger OUT = new AtomicInteger();
    static final int BUFFERS_IN_PLAY = 6;
    static final int IM = 139968;
    static final int IA = 3877;
    static final int IC = 29573;
    static final float ONE_OVER_IM = 1f / IM;
    static int last = 42;

    public static void main(String[] args) {
        int n = 1000;

        if (args.length > 0) {
            n = Integer.parseInt(args[0]);
        }
        for (int i = 0; i < WORKERS.length; i++) {
            WORKERS[i] = new NucleotideSelector();
            WORKERS[i].setDaemon(true);
            WORKERS[i].start();
        }
        try (OutputStream writer = System.out;) {
            final int bufferSize = LINE_COUNT * LINE_LENGTH;

            for (int i = 0; i < BUFFERS_IN_PLAY; i++) {
                lineFillALU(
                                final byte[] sapienChars = new byte[]{
            'a',
            'c',
            'g',
            't'};
        final double[] sapienProbs = new double[]{
            0.3029549426680,
            0.1979883004921,
            0.1975473066391,
            0.3015094502008};
        final float[] probs;
        final float[] randoms;
        final int charsInFullLines;

        public Buffer(final boolean isIUB
                , final int lineLength
                , final int nChars) {
            super(lineLength, nChars);
            double cp = 0;
            final double[] dblProbs = isIUB ? iubProbs : sapienProbs;

            chars = isIUB ? iubChars : sapienChars;
            probs = new float[dblProbs.length];
            for (int i = 0; i < probs.length; i++) {
                cp += dblProbs[i];
                probs[i] = (float) cp;
            }
            probs[probs.length - 1] = 2f;
            randoms = new float[nChars];
            charsInFullLines = (nChars / lineLength) * lineLength;
        }

        @Override
        public void selectNucleotides() {
            int i, j, m;
            float r;
            int k;

            for (i = 0, j = 0; i < charsInFullLines; j++) {
                for (k = 0; k < LINE_LENGTH; k++) {
                    r = randoms[i++];
                    for (m = 0; probs[m] < r; m++) {
                    }
                    nucleotides[j++] = chars[m];
                }
            }
            for (k = 0; k < CHARS_LEFTOVER; k++) {
                r = randoms[i++];
                for (m = 0; probs[m] < r; m++) {
                }
                nucleotides[j++] = chars[m];
            }
        }
    }
}
```

Fasta (325 loc)

Output →

```
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCT
CTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACT
CGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCC
GAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAG
GCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCG
GATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTC
TACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTC
GGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCG
AGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGG
CCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGG
ATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCT
ACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCG
GGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGA
GATCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGC
CGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGA
TCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTA
CTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGG
GAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAG
ATCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCC
GGGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGAT
CACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTAC
TAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCCG
GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATC
ACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTACT
AAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCCG
GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATC
ACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTACT
AAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCCG
GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATC
ACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTACT
AAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCCG
GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATC
ACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTACT
AAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCCG
GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATC
ACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTACT
AAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCCG
GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATC
ACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTACT
AAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAAGGCCG
GGCGCGGTGGCTCACGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGCGGATC
ACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTCTCTACT
AAAAATACAAAAATTAGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTACTCGGG
AGGCTGAGGCAGGAGAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGCCGAGA
TCGCGCCACTGCACTCCAGCCTGGGCGACAGAGCGAGACTCCGTGTAGGTAGGATAGT
```

```java
import java.io.IOException;
import java.io.OutputStream;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.atomic.AtomicInteger;

public class fasta {

    static final int LINE_LENGTH = 60;
    static final int LINE_COUNT = 1024;
    static final NucleotideSelector[] WORKERS
            = new NucleotideSelector[
                    Runtime.getRuntime().availableProcessors() > 1
                    ? Runtime.getRuntime().availableProcessors() - 1
                    : 1];
    static final AtomicInteger IN = new AtomicInteger();
    static final AtomicInteger OUT = new AtomicInteger();
    static final int BUFFERS_IN_PLAY = 6;
    static final int IM = 139968;
    static final int IA = 3877;
    static final int IC = 29573;
    static final float ONE_OVER_IM = 1f / IM;
    static int last = 42;

    public static void main(String[] args) {
        int n = 1000;

        if (args.length > 0) {
            n = Integer.parseInt(args[0]);
        }
        for (int i = 0; i < WORKERS.length; i++) {
            WORKERS[i] = new NucleotideSelector();
            WORKERS[i].setDaemon(true);
            WORKERS[i].start();
        }
        try (OutputStream writer = System.out;) {
            final int bufferSize = LINE_COUNT * LINE_LENGTH;

            for (int i = 0; i < BUFFERS_IN_PLAY; i++) {
                lineFillALU(
                            final byte[] sapienChars = new byte[]{
            'a',
            'c',
            'g',
            't'};
        final double[] sapienProbs = new double[]{
            0.3029549426680,
            0.1979883004921,
            0.1975473066391,
            0.3015094502008};
        final float[] probs;
        final float[] randoms;
        final int charsInFullLines;

        public Buffer(final boolean isIUB
                , final int lineLength
                , final int nChars) {
            super(lineLength, nChars);
            double cp = 0;
            final double[] dblProbs = isIUB ? iubProbs : sapienProbs;

            chars = isIUB ? iubChars : sapienChars;
            probs = new float[dblProbs.length];
            for (int i = 0; i < probs.length; i++) {
                cp += dblProbs[i];
                probs[i] = (float) cp;
            }
            probs[probs.length - 1] = 2f;
            randoms = new float[nChars];
            charsInFullLines = (nChars / lineLength) * lineLength;
        }

        @Override
        public void selectNucleotides() {
            int i, j, m;
            float r;
            int k;

            for (i = 0, j = 0; i < charsInFullLines; j++) {
                for (k = 0; k < LINE_LENGTH; k++) {
                    r = randoms[i++];
                    for (m = 0; probs[m] < r; m++) {
                    }
                    nucleotides[j++] = chars[m];
                }
            }
            for (k = 0; k < CHARS_LEFTOVER; k++) {
                r = randoms[i++];
                for (m = 0; probs[m] < r; m++) {
                }
                nucleotides[j++] = chars[m];
            }
        }
    }
}
```

Fasta (325 loc)

Output

```java
import it.unimi.dsi.fastutil.longs.Long2IntOpenHashMap;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Map.Entry;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class knucleotide {
    static final byte[] codes = { -1, 0, -1, 1, 3, -1, -1, 2 };
    static final char[] nucleotides = { 'A', 'C', 'G', 'T' };

    static class Result {
        Long2IntOpenHashMap map = new Long2IntOpenHashMap();
        int keyLength;

        public Result(int keyLength) {
            this.keyLength = keyLength;
        }
    }

    static ArrayList<Callable<Result>> createFragmentTasks(final byte[] sequence,
            int[] fragmentLengths) {
        ArrayList<Callable<Result>> tasks = new ArrayList<>();
        for (int fragmentLength : fragmentLengths) {
            for (int index = 0; index < fragmentLength; index++) {
                int offset = index;
                tasks.add(() -> createFragmentMap(sequence, offset, fragmentLength));
            }
        }
        return tasks;
    }

    static Result createFragmentMap(byte[] sequence, int offset, int fragmentLength) {
        Result res = new Result(fragmentLength);
        Long2IntOpenHashMap map = res.map;
        int lastIndex = sequence.length - fragmentLength + 1;
        for (int index = offset; index < lastIndex; index += fragmentLength) {
            map.addTo(getKey(sequence, index, fragmentLength), 1);
        }

        return res;
    }

    /**
     * Convert given byte array (limiting to given length) containing acgtACGT
     * to codes (0 = A, 1 = C, 2 = G, 3 = T) and returns new array
     */
    static byte[] toCodes(byte[] sequence, int length) {
        byte[] result = new byte[length];
        for (int i = 0; i < length; i++) {
            result[i] = codes[sequence[i] & 0x7];
        }
        return result;
    }


        byte[] bytes = new byte[1048576];
        int position = 0;
        while ((line = in.readLine()) != null && line.charAt(0) != '>') {
            if (line.length() + position > bytes.length) {
                byte[] newBytes = new byte[bytes.length * 2];
                System.arraycopy(bytes, 0, newBytes, 0, position);
                bytes = newBytes;
            }
            for (int i = 0; i < line.length(); i++)
                bytes[position++] = (byte) line.charAt(i);
        }

        return toCodes(bytes, position);
    }

    public static void main(String[] args) throws Exception {
        byte[] sequence = read(System.in);

        ExecutorService pool = Executors.newFixedThreadPool(Runtime.getRuntime()
                .availableProcessors());
        int[] fragmentLengths = { 1, 2, 3, 4, 6, 12, 18 };
        List<Future<Result>> futures = pool.invokeAll(createFragmentTasks(sequence,
                fragmentLengths));
        pool.shutdown();

        StringBuilder sb = new StringBuilder();

        sb.append(writeFrequencies(sequence.length, futures.get(0).get()));
        sb.append(writeFrequencies(sequence.length - 1,
                sumTwoMaps(futures.get(1).get(), futures.get(2).get())));

        String[] nucleotideFragments = { "GGT", "GGTA", "GGTATT", "GGTATTTTAATT",
                "GGTATTTTAATTTATAGT" };
        for (String nucleotideFragment : nucleotideFragments) {
            sb.append(writeCount(futures, nucleotideFragment));
        }

        System.out.print(sb);
    }
}
```

k-nucleotide (174 loc)

```
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
```

Input →

```java
import it.unimi.dsi.fastutil.longs.Long2IntOpenHashMap;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Map.Entry;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class knucleotide {
    static final byte[] codes = { -1, 0, -1, 1, 3, -1, -1, 2 };
    static final char[] nucleotides = { 'A', 'C', 'G', 'T' };

    static class Result {
        Long2IntOpenHashMap map = new Long2IntOpenHashMap();
        int keyLength;

        public Result(int keyLength) {
            this.keyLength = keyLength;
        }
    }

    static ArrayList<Callable<Result>> createFragmentTasks(final byte[] sequence,
            int[] fragmentLengths) {
        ArrayList<Callable<Result>> tasks = new ArrayList<>();
        for (int fragmentLength : fragmentLengths) {
            for (int index = 0; index < fragmentLength; index++) {
                int offset = index;
                tasks.add(() -> createFragmentMap(sequence, offset, fragmentLength));
            }
        }
        return tasks;
    }

    static Result createFragmentMap(byte[] sequence, int offset, int fragmentLength) {
        Result res = new Result(fragmentLength);
        Long2IntOpenHashMap map = res.map;
        int lastIndex = sequence.length - fragmentLength + 1;
        for (int index = offset; index < lastIndex; index += fragmentLength) {
            map.addTo(getKey(sequence, index, fragmentLength), 1);
        }

        return res;
    }

    /**
     * Convert given byte array (limiting to given length) containing acgtACGT
     * to codes (0 = A, 1 = C, 2 = G, 3 = T) and returns new array
     */
    static byte[] toCodes(byte[] sequence, int length) {
        byte[] result = new byte[length];
        for (int i = 0; i < length; i++) {
            result[i] = codes[sequence[i] & 0x7];
        }
        return result;
    }


        byte[] bytes = new byte[1048576];
        int position = 0;
        while ((line = in.readLine()) != null && line.charAt(0) != '>') {
            if (line.length() + position > bytes.length) {
                byte[] newBytes = new byte[bytes.length * 2];
                System.arraycopy(bytes, 0, newBytes, 0, position);
                bytes = newBytes;
            }
            for (int i = 0; i < line.length(); i++)
                bytes[position++] = (byte) line.charAt(i);
        }

        return toCodes(bytes, position);
    }

    public static void main(String[] args) throws Exception {
        byte[] sequence = read(System.in);

        ExecutorService pool = Executors.newFixedThreadPool(Runtime.getRuntime()
                .availableProcessors());
        int[] fragmentLengths = { 1, 2, 3, 4, 6, 12, 18 };
        List<Future<Result>> futures = pool.invokeAll(createFragmentTasks(sequence,
                fragmentLengths));
        pool.shutdown();

        StringBuilder sb = new StringBuilder();

        sb.append(writeFrequencies(sequence.length, futures.get(0).get()));
        sb.append(writeFrequencies(sequence.length - 1,
                sumTwoMaps(futures.get(1).get(), futures.get(2).get())));

        String[] nucleotideFragments = { "GGT", "GGTA", "GGTATT", "GGTATTTTAATT",
                "GGTATTTTAATTTATAGT" };
        for (String nucleotideFragment : nucleotideFragments) {
            sb.append(writeCount(futures, nucleotideFragment));
        }

        System.out.print(sb);
    }
}
```

k-nucleotide (174 loc)

```
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
```

Input →

```java
import it.unimi.dsi.fastutil.longs.Long2IntOpenHashMap;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Map.Entry;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class knucleotide {
    static final byte[] codes = { -1, 0, -1, 1, 3, -1, -1, 2 };
    static final char[] nucleotides = { 'A', 'C', 'G', 'T' };

    static class Result {
        Long2IntOpenHashMap map = new Long2IntOpenHashMap();
        int keyLength;

        public Result(int keyLength) {
            this.keyLength = keyLength;
        }
    }

    static ArrayList<Callable<Result>> createFragmentTasks(final byte[] sequence,
            int[] fragmentLengths) {
        ArrayList<Callable<Result>> tasks = new ArrayList<>();
        for (int fragmentLength : fragmentLengths) {
            for (int index = 0; index < fragmentLength; index++) {
                int offset = index;
                tasks.add(() -> createFragmentMap(sequence, offset, fragmentLength));
            }
        }
        return tasks;
    }

    static Result createFragmentMap(byte[] sequence, int offset, int fragmentLength) {
        Result res = new Result(fragmentLength);
        Long2IntOpenHashMap map = res.map;
        int lastIndex = sequence.length - fragmentLength + 1;
        for (int index = offset; index < lastIndex; index += fragmentLength) {
            map.addTo(getKey(sequence, index, fragmentLength), 1);
        }

        return res;
    }

    /**
     * Convert given byte array (limiting to given length) containing acgtACGT
     * to codes (0 = A, 1 = C, 2 = G, 3 = T) and returns new array
     */
    static byte[] toCodes(byte[] sequence, int length) {
        byte[] result = new byte[length];
        for (int i = 0; i < length; i++) {
            result[i] = codes[sequence[i] & 0x7];
        }
        return result;
    }


        byte[] bytes = new byte[1048576];
        int position = 0;
        while ((line = in.readLine()) != null && line.charAt(0) != '>') {
            if (line.length() + position > bytes.length) {
                byte[] newBytes = new byte[bytes.length * 2];
                System.arraycopy(bytes, 0, newBytes, 0, position);
                bytes = newBytes;
            }
            for (int i = 0; i < line.length(); i++)
                bytes[position++] = (byte) line.charAt(i);
        }

        return toCodes(bytes, position);
    }

    public static void main(String[] args) throws Exception {
        byte[] sequence = read(System.in);

        ExecutorService pool = Executors.newFixedThreadPool(Runtime.getRuntime()
                .availableProcessors());
        int[] fragmentLengths = { 1, 2, 3, 4, 6, 12, 18 };
        List<Future<Result>> futures = pool.invokeAll(createFragmentTasks(sequence,
                fragmentLengths));
        pool.shutdown();

        StringBuilder sb = new StringBuilder();

        sb.append(writeFrequencies(sequence.length, futures.get(0).get()));
        sb.append(writeFrequencies(sequence.length - 1,
                sumTwoMaps(futures.get(1).get(), futures.get(2).get())));

        String[] nucleotideFragments = { "GGT", "GGTA", "GGTATT", "GGTATTTTAATT",
                "GGTATTTTAATTTATAGT" };
        for (String nucleotideFragment : nucleotideFragments) {
            sb.append(writeCount(futures, nucleotideFragment));
        }

        System.out.print(sb);
    }
}
```

k-nucleotide (174 loc)

Output →

```
A 30.295
T 30.151
C 19.800
G 19.754

AA 9.177
TA 9.132
AT 9.131
TT 9.091
CA 6.002
AC 6.001
AG 5.987
GA 5.984
CT 5.971
TC 5.971
GT 5.957
TG 5.956
CC 3.917
GC 3.911
CG 3.909
GG 3.902

1471758    GGT
446535     GGTA
47336      GGTATT
893   GGTATTTTAATT
```

```
import java.io.Closeable;
import java.io.FileDescriptor;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class revcomp {

    public static void main(String[] args) throws Exception {
        try (Strand strand = new Strand();
                FileInputStream standIn = new
FileInputStream(FileDescriptor.in);
                FileOutputStream standOut = new
FileOutputStream(FileDescriptor.out);;) {

            while (strand.readOneStrand(standIn) >= 0) {
                strand.reverse();
                strand.write(standOut);
                strand.reset();
            }


class Strand implements Closeable {

    private static final byte NEW_LINE = '\n';
    private static final byte ANGLE = '>';
    private static final int LINE_LENGTH = 61;

    private static final byte[] map = new byte[128];
    static {
        for (int i = 0; i < map.length; i++) {
            map[i] = (byte) i;
        }

        map['t'] = map['T'] = 'A';
        map['a'] = map['A'] = 'T';
        map['g'] = map['G'] = 'C';
        map['c'] = map['C'] = 'G';
        map['v'] = map['V'] = 'B';
        map['h'] = map['H'] = 'D';
        map['r'] = map['R'] = 'Y';
        map['m'] = map['M'] = 'K';
        map['y'] = map['Y'] = 'R';
        map['k'] = map['K'] = 'M';
        map['b'] = map['B'] = 'V';
        map['d'] = map['D'] = 'H';
        map['u'] = map['U'] = 'A';
    }

    private static int NCPU = Runtime.getRuntime().availableProcessors();
    private ExecutorService executor = Executors.newFixedThreadPool(NCPU);

    private int chunkCount = 0;

    private final ArrayList<Chunk> chunks = new ArrayList<Chunk>();

    private void ensureSize() {
        if (chunkCount == chunks.size()) {
            chunks.add(new Chunk());
        }
    }

    private boolean isLastChunk(Chunk chunk) {
        return chunk.
            if (leftIndex <= leftEndIndex) {
                byte lByte = leftBytes[leftIndex];
                byte rByte = rightBytes[rightIndex];

                leftBytes[leftIndex++] = map[rByte];
                rightBytes[rightIndex--] = map[lByte];
            }
        }

    }

    private int ceilDiv(int a, int b) {
        return (a + b - 1) / b;
    }

    private int getSumLength() {
        int sumLength = 0;

        for (int i = 0; i < chunkCount; i++) {
            sumLength += chunks.get(i).length;
        }

        return sumLength;
    }
```

revcomp (296 loc)

@gustavopinto

```
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
```

Input →

```java
import java.io.Closeable;
import java.io.FileDescriptor;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class revcomp {

    public static void main(String[] args) throws Exception {
        try (Strand strand = new Strand();
                FileInputStream standIn = new
FileInputStream(FileDescriptor.in);
                FileOutputStream standOut = new
FileOutputStream(FileDescriptor.out);) {

            while (strand.readOneStrand(standIn) >= 0) {
                strand.reverse();
                strand.write(standOut);
                strand.reset();
            }


class Strand implements Closeable {

    private static final byte NEW_LINE = '\n';
    private static final byte ANGLE = '>';
    private static final int LINE_LENGTH = 61;

    private static final byte[] map = new byte[128];
    static {
        for (int i = 0; i < map.length; i++) {
            map[i] = (byte) i;
        }

        map['t'] = map['T'] = 'A';
        map['a'] = map['A'] = 'T';
        map['g'] = map['G'] = 'C';
        map['c'] = map['C'] = 'G';
        map['v'] = map['V'] = 'B';
        map['h'] = map['H'] = 'D';
        map['r'] = map['R'] = 'Y';
        map['m'] = map['M'] = 'K';
        map['y'] = map['Y'] = 'R';
        map['k'] = map['K'] = 'M';
        map['b'] = map['B'] = 'V';
        map['d'] = map['D'] = 'H';
        map['u'] = map['U'] = 'A';
    }

    private static int NCPU = Runtime.getRuntime().availableProcessors();
    private ExecutorService executor = Executors.newFixedThreadPool(NCPU);

    private int chunkCount = 0;

    private final ArrayList<Chunk> chunks = new ArrayList<Chunk>();

    private void ensureSize() {
        if (chunkCount == chunks.size()) {
            chunks.add(new Chunk());
        }
    }

    private boolean isLastChunk(Chunk chunk) {
        return chunk.
            if (leftIndex <= leftEndIndex) {
                byte lByte = leftBytes[leftIndex];
                byte rByte = rightBytes[rightIndex];

                leftBytes[leftIndex++] = map[rByte];
                rightBytes[rightIndex--] = map[lByte];
            }
        }

    }

    private int ceilDiv(int a, int b) {
        return (a + b - 1) / b;
    }

    private int getSumLength() {
        int sumLength = 0;

        for (int i = 0; i < chunkCount; i++) {
            sumLength += chunks.get(i).length;
        }

        return sumLength;
    }
```

revcomp (296 loc)

Input

```
import java.io.Closeable;
import java.io.FileDescriptor;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class revcomp {

    public static void main(String[] args) throws Exception {
        try (Strand strand = new Strand();
                FileInputStream standIn = new
FileInputStream(FileDescriptor.in);
                FileOutputStream standOut = new
FileOutputStream(FileDescriptor.out);) {

            while (strand.readOneStrand(standIn) >= 0) {
                strand.reverse();
                strand.write(standOut);
                strand.reset();
            }


class Strand implements Closeable {

    private static final byte NEW_LINE = '\n';
    private static final byte ANGLE = '>';
    private static final int LINE_LENGTH = 61;

    private static final byte[] map = new byte[128];
    static {
        for (int i = 0; i < map.length; i++) {
            map[i] = (byte) i;
        }

        map['t'] = map['T'] = 'A';
        map['a'] = map['A'] = 'T';
        map['g'] = map['G'] = 'C';
        map['c'] = map['C'] = 'G';
        map['v'] = map['V'] = 'B';
        map['h'] = map['H'] = 'D';
        map['r'] = map['R'] = 'Y';
        map['m'] = map['M'] = 'K';
        map['y'] = map['Y'] = 'R';
        map['k'] = map['K'] = 'M';
        map['b'] = map['B'] = 'V';
        map['d'] = map['D'] = 'H';
        map['u'] = map['U'] = 'A';
    }

    private static int NCPU = Runtime.getRuntime().availableProcessors();
    private ExecutorService executor = Executors.newFixedThreadPool(NCPU);

    private int chunkCount = 0;

    private final ArrayList<Chunk> chunks = new ArrayList<Chunk>();

    private void ensureSize() {
        if (chunkCount == chunks.size()) {
            chunks.add(new Chunk());
        }
    }

    private boolean isLastChunk(Chunk chunk) {
        return chunk.
            if (leftIndex <= leftEndIndex) {
                byte lByte = leftBytes[leftIndex];
                byte rByte = rightBytes[rightIndex];

                leftBytes[leftIndex++] = map[rByte];
                rightBytes[rightIndex--] = map[lByte];
            }
        }

    }

    private int ceilDiv(int a, int b) {
        return (a + b - 1) / b;
    }

    private int getSumLength() {
        int sumLength = 0;

        for (int i = 0; i < chunkCount; i++) {
            sumLength += chunks.get(i).length;
        }

        return sumLength;
    }
```

revcomp (296 loc)

Output

GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC

TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT

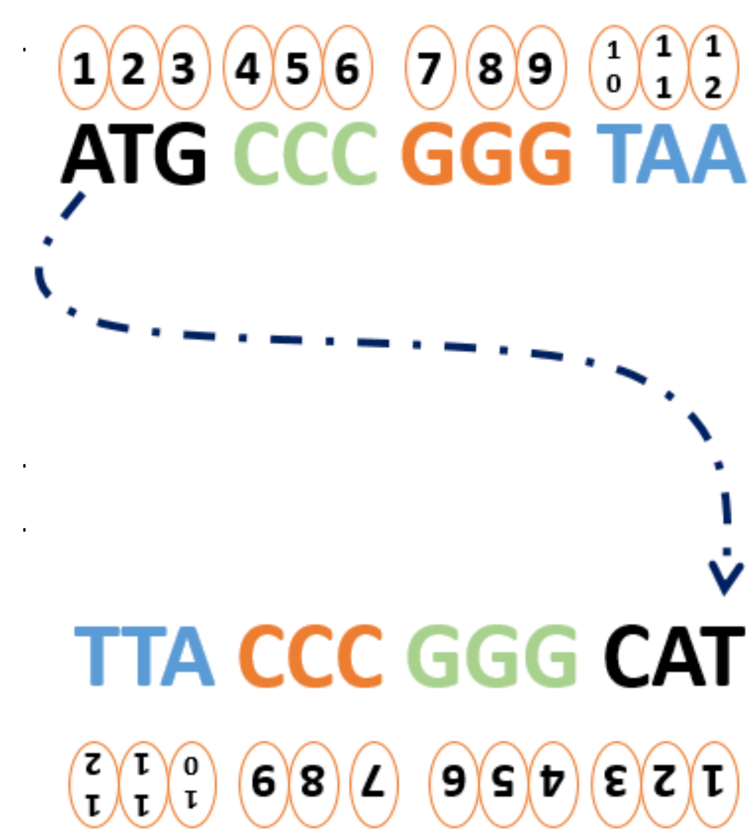
Input →

```java
import java.io.Closeable;
import java.io.FileDescriptor;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class revcomp {

    public static void main(String[] args) throws Exception {
        try (Strand strand = new Strand();
                FileInputStream standIn = new
FileInputStream(FileDescriptor.in);
                FileOutputStream standOut = new
FileOutputStream(FileDescriptor.out);) {

            while (strand.readOneStrand(standIn) >= 0) {
                strand.reverse();
                strand.write(standOut);
                strand.reset();
            }


class Strand implements Closeable {

    private static final byte NEW_LINE = '\n';
    private static final byte ANGLE = '>';
    private static final int LINE_LENGTH = 61;

    private static final byte[] map = new byte[128];
    static {
        for (int i = 0; i < map.length; i++) {
            map[i] = (byte) i;
        }

        map['t'] = map['T'] = 'A';
        map['a'] = map['A'] = 'T';
        map['g'] = map['G'] = 'C';
        map['c'] = map['C'] = 'G';
        map['v'] = map['V'] = 'B';
        map['h'] = map['H'] = 'D';
        map['r'] = map['R'] = 'Y';
        map['m'] = map['M'] = 'K';
        map['y'] = map['Y'] = 'R';
        map['k'] = map['K'] = 'M';
        map['b'] = map['B'] = 'V';
        map['d'] = map['D'] = 'H';
        map['u'] = map['U'] = 'A';
    }

    private static int NCPU = Runtime.getRuntime().availableProcessors();
    private ExecutorService executor = Executors.newFixedThreadPool(NCPU);

    private int chunkCount = 0;

    private final ArrayList<Chunk> chunks = new ArrayList<Chunk>();

    private void ensureSize() {
        if (chunkCount == chunks.size()) {
            chunks.add(new Chunk());
        }
    }

    private boolean isLastChunk(Chunk chunk) {
        return chunk.
            if (leftIndex <= leftEndIndex) {
                byte lByte = leftBytes[leftIndex];
                byte rByte = rightBytes[rightIndex];

                leftBytes[leftIndex++] = map[rByte];
                rightBytes[rightIndex--] = map[lByte];
            }
        }

    }

    private int ceilDiv(int a, int b) {
        return (a + b - 1) / b;
    }

    private int getSumLength() {
        int sumLength = 0;

        for (int i = 0; i < chunkCount; i++) {
            sumLength += chunks.get(i).length;
        }

        return sumLength;
    }
```

revcomp (296 loc)

Output →

GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC



GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT
TTGGGAGGCCGAGGCGGGCGGATCACCTGAGGTCAGGA
GTTCGAGACCAGCCTGGCCAACATGGTGAAACCCCGTC
TCTACTAAAAATACAAAAATTAGCCGGGCGTGGTGGCGCG
CGCCTGTAATCCCAGCTACTCGGGAGGCTGAGGCAGGA
GAATCGCTTGAACCCGGGAGGCGGAGGTTGCAGTGAGC
CGAGATCGCGCCACTGCACTCCAGCCTGGGCGACAGAG
CGAGACTCCGTCTCAAAAAGGCCGGGCGCGGTGGCTCA
CGCCTGTAATCCCAGCACTTTGGGAGGCCGAGGCGGGC
GGATCACCTGAGGTCAGGAGTTCGAGACCAGCCTGGCC
AACATGGTGAAACCCCGTCTCTACTAAAAATACAAAAATT
AGCCGGGCGTGGTGGCGCGCCTGTAATCCCAGCTAC
TCGGGAGGCTGAGGCAGGAGAATCGCTTGAACCCGGGA
GGCGGAGGTTGCAGTGAGCCGAGATCGCGCCACTGCAC
TCCAGCCTGGGCGACAGAGCGAGACTCCGTCTCAAAAA
GGCCGGGCGCGGTGGCTCACGCCTGTAATCCCAGCACT

Macro benchmarks

Xalan™
XSLT
Apache Software Foundation

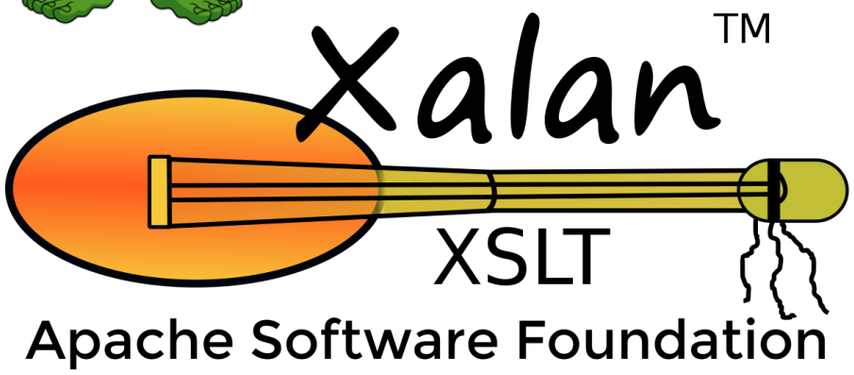
Apache Commons™
http://commons.apache.org/

FOP

batik
JAVA SVG TOOLKIT

PGJDBC

Macro benchmarks



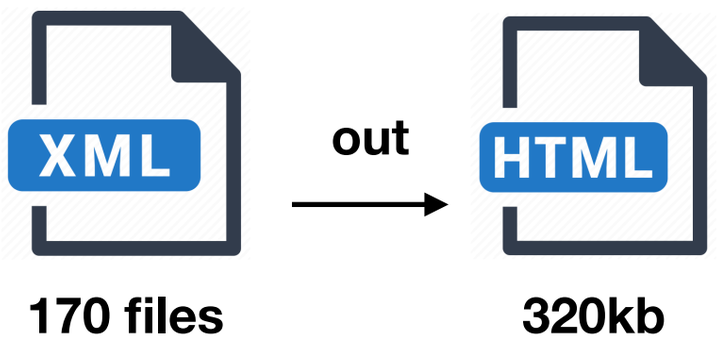
- Parses XML in HTML documents
- More than 188K lines of Java code
- More than 40 FileInputStream

Macro benchmarks

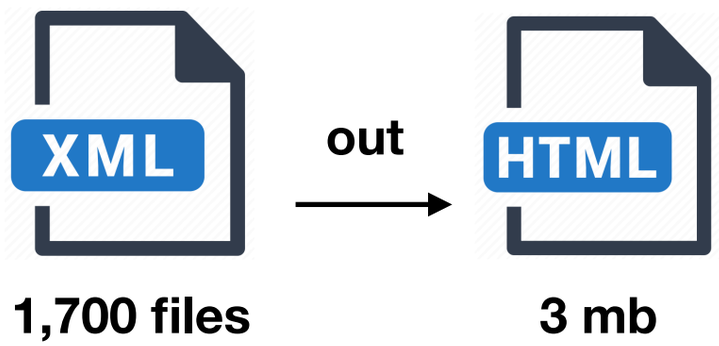
Xalan™ XSLT
Apache Software Foundation

- > Parses XML in HTML documents
- > More than 188K lines of Java code
- > More than 40 FileInputStream

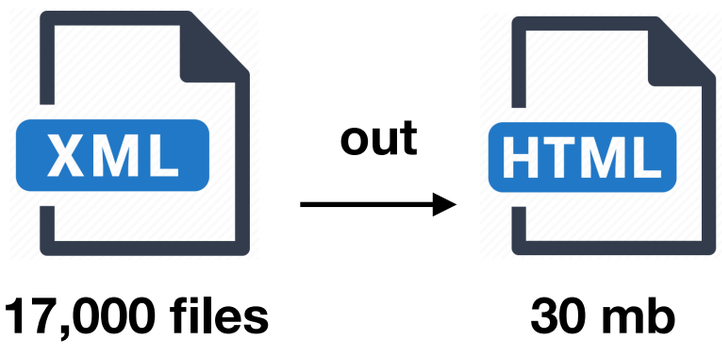
3 workloads



Small

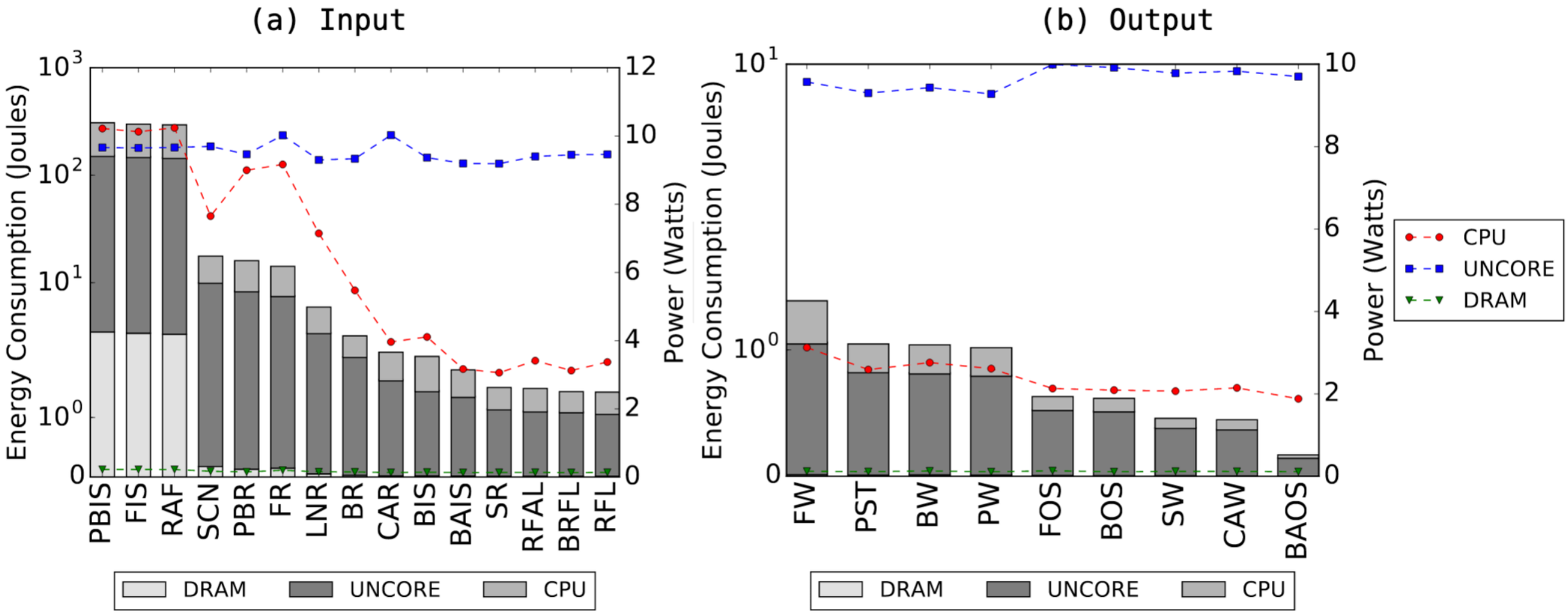


Default

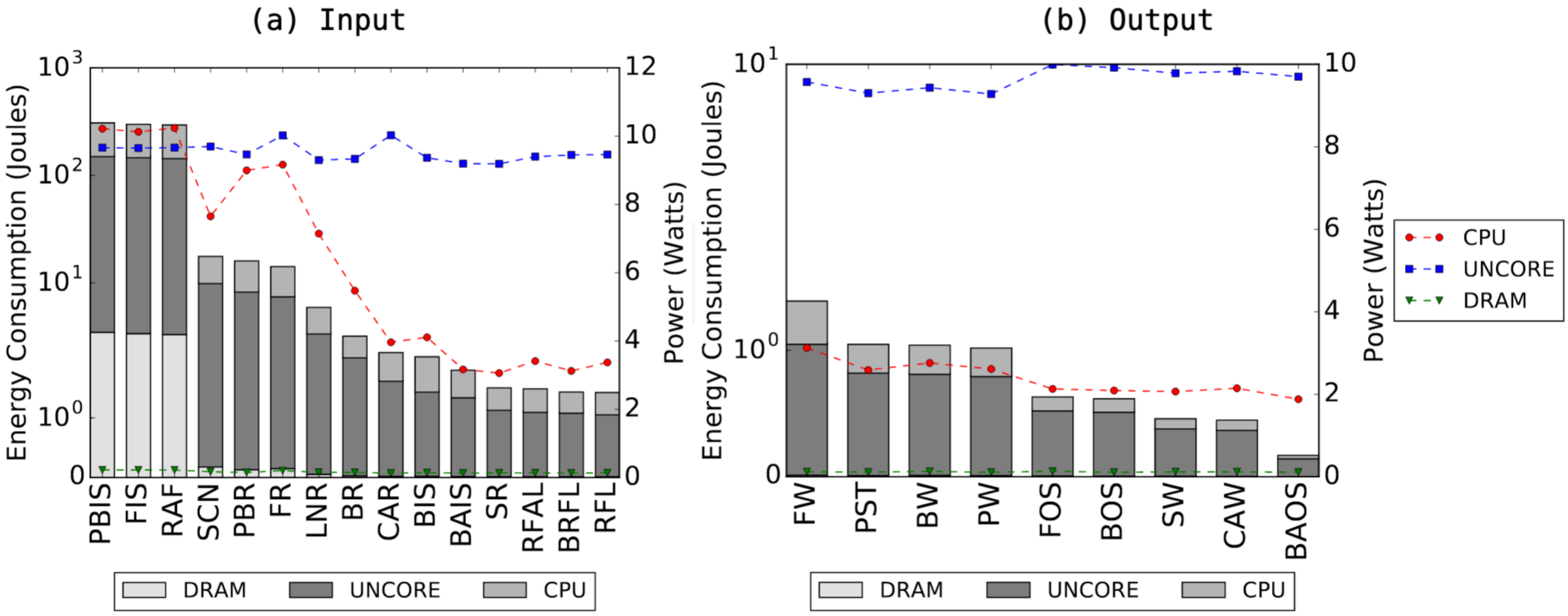


Large

RQ1: Energy behaviors

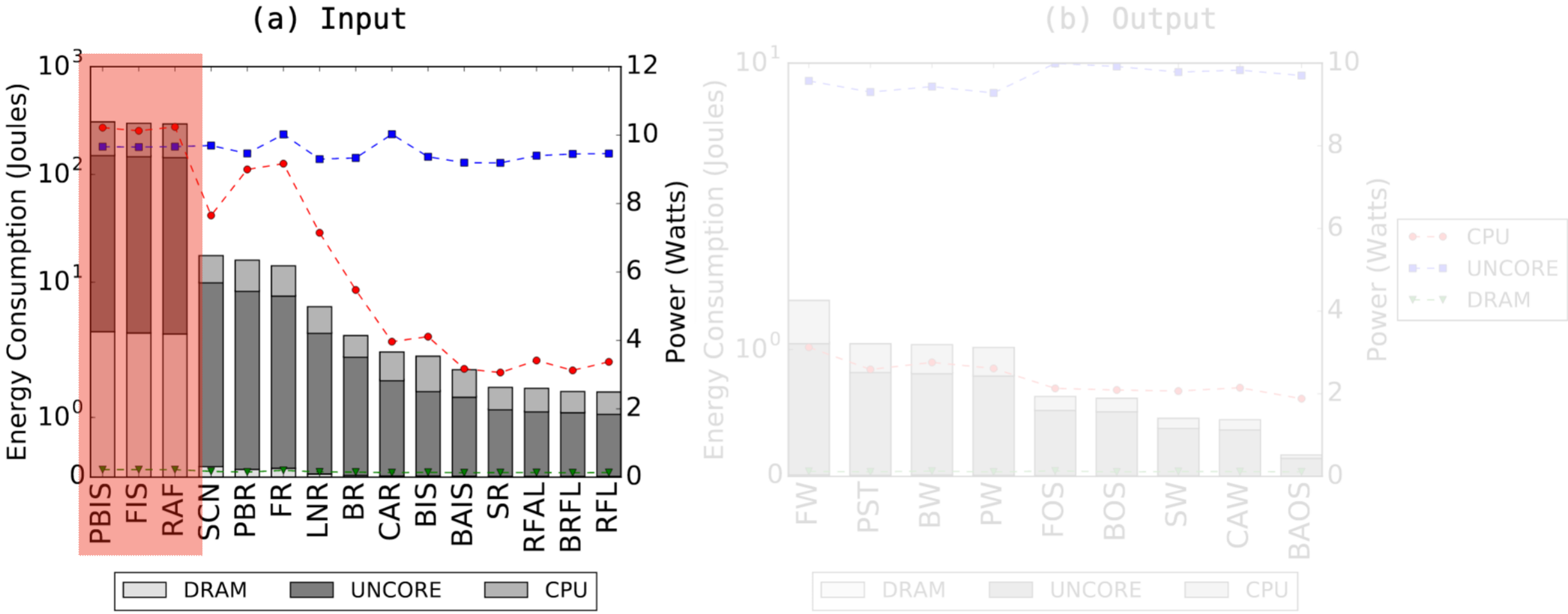


RQ1: Energy behaviors



Reading consume ~3x more than writing operations

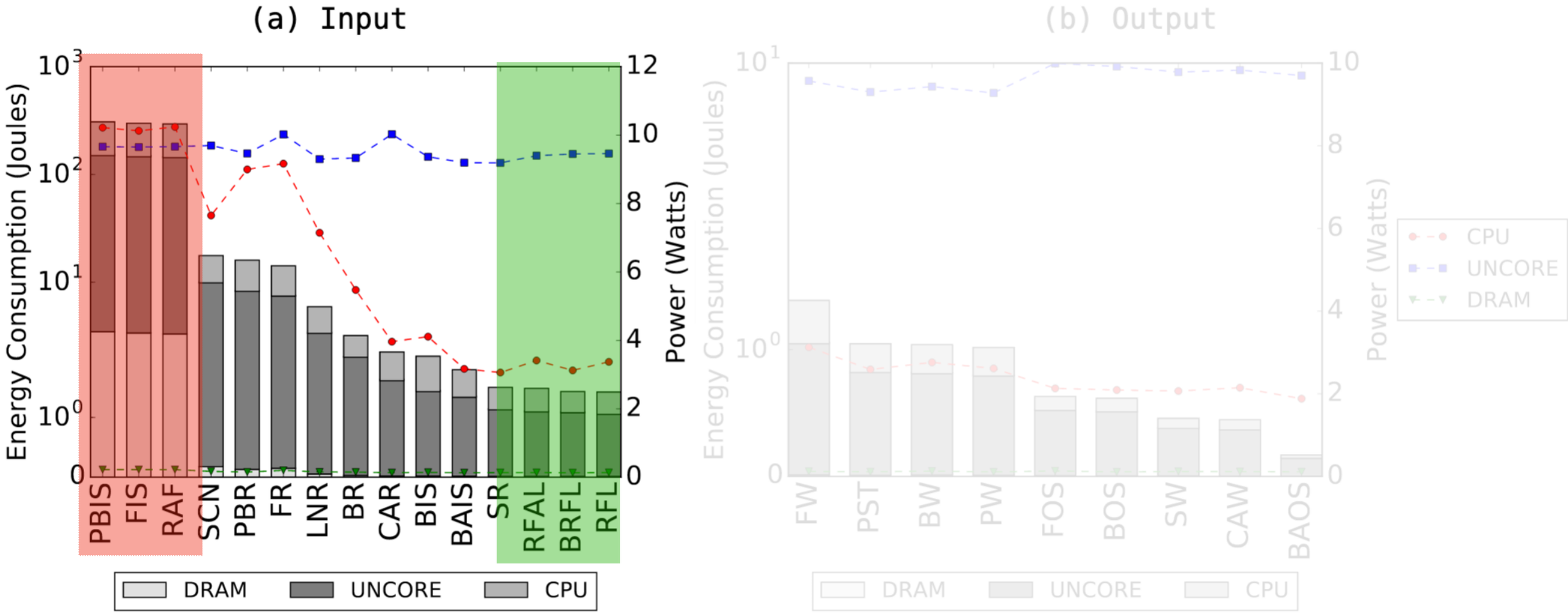
RQ1: Energy behaviors



PBIS: PushBackInputStream
FIS: FileInputStream
RAF: RandomAccessFile

@gustavopinto

RQ1: Energy behaviors

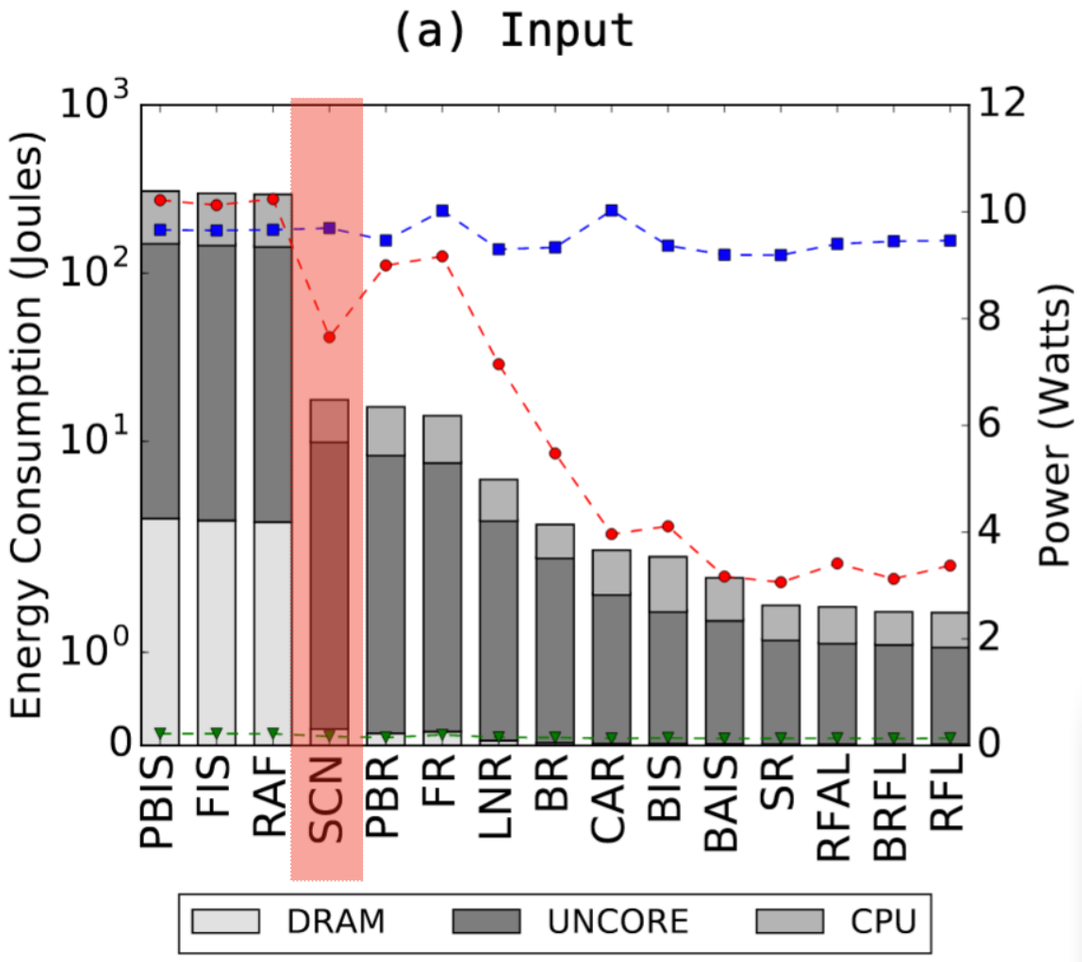


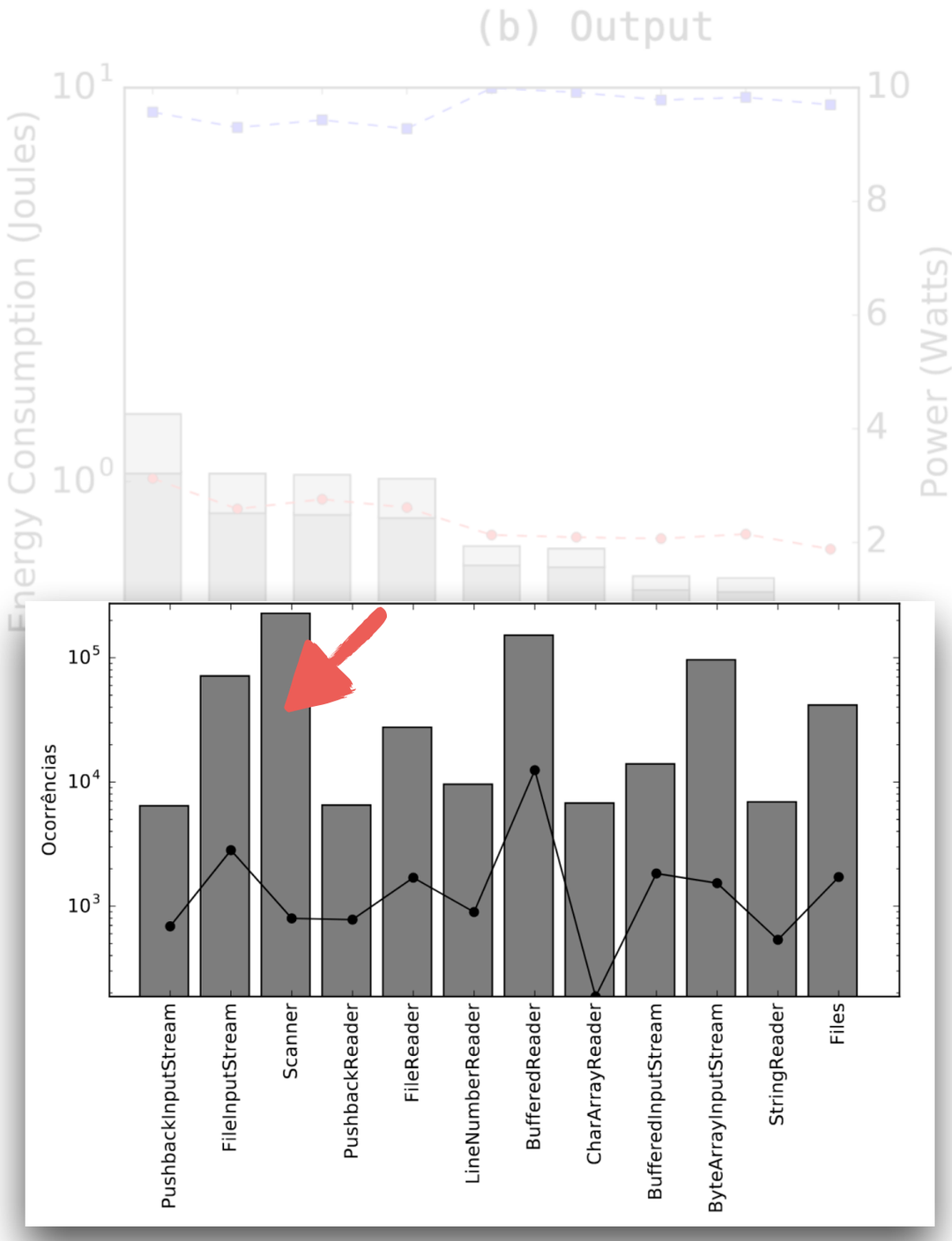
PBIS: PushBackInputStream
FIS: FileInputStream
RAF: RandomAccessFile

RFAL: Files.readAllLines()
BRFL: Files.newBufferedReader()
RFAL: Files.lines()

@gustavopinto

RQ1: Energy behaviors





SCN: Scanner

The most used Java I/O API

@gustavopinto

RQ2: Does refactoring play a role?

1. We identified all instances of Java I/O APIs

2. We refactored these instances to other Java I/O APIs that inherit from the same parent class

3. We made sure it compile and does not raise runtime errors

4. We benchmarked again

RQ2: Does refactoring play a role?

1. We identified all instances of Java I/O APIs

2. We refactored these instances to other Java I/O APIs that inherit from the same parent class

3. We made sure it compile and does not raise runtime errors

4. We benchmarked again

22 manual refactorings performed

RQ2: Does refactoring play a role?

Classes	optimized benchmarks			macro-benchmarks				
	FA	KN	RC	XALAN	FOP	BATIK	COMMONS-IO	PGJDBC
`FileInputStream`	✖	☹	</>	</>	✖	✖	☹	✖
`BufferedInputStream`	✖	☹	+16.31%	-17.19%	✖	✖	☹	✖
`BufferedReader`	✖	</>	☹	☹	✖	✖	</>	✖
`LineNumberReader`	✖	-7.52%	☹	☹	✖	✖	-16.47%	✖
`Files.readAllLines`	✖	-0.87%	☹	☹	✖	✖	-9.5%	✖
`Scanner`	✖	+60.77%	☹	☹	✖	✖	+486.28%	✖
`PrintStream`	+19.13%	✖	+15.24%	-7.96%	+48.38%	-3.68%	✖	+811.41%
`FileOutputStream`	+29.13%	✖	</>	</>	+73.91%	</>	✖	☹
`DataOutputStream`	☹	✖	☹	☹	☹	☹	✖	+883.22%
`ByteArrayOutputStream`	+36.24%	✖	☹	☹	☹	☹	✖	☹
`BufferedOutputStream`	+19.23%	✖	+25.96%	-11.71%	</>	-3.1%	✖	</>
`System.out`	</>	✖	☹	☹	☹	☹	✖	☹

RQ2: Does refactoring play a role?

Classes	optimized benchmarks			macro-benchmarks				
	FA	KN	RC	XALAN	FOP	BATIK	COMMONS-IO	PGJDBC
FileInputStream	✖	☹	</>	</>	✖	✖	☹	✖
BufferedInputStream	✖	☹	+16.31%	-17.19%	✖	✖	☹	✖
BufferedReader	✖	</>	☹	☹	✖	✖	</>	✖
LineNumberReader	✖	-7.52%	☹	☹	✖	✖	-16.47%	✖
Files.readAllLines	✖	-0.87%	☹	☹	✖	✖	-9.5%	✖
Scanner	✖	+60.77%	☹	☹	✖	✖	+486.28%	✖
PrintStream	+19.13%	✖	+15.24%	-7.96%	+48.38%	-3.68%	✖	+811.41%
FileOutputStream	+29.13%	✖	</>	</>	+73.91%	</>	✖	☹
DataOutputStream	☹	✖	☹	☹	☹	☹	✖	+883.22%
ByteArrayOutputStream	+36.24%	✖	☹	☹	☹	☹	✖	☹
BufferedOutputStream	+19.23%	✖	+25.96%	-11.71%	</>	-3.1%	✖	</>
System.out	</>	✖	☹	☹	☹	☹	✖	☹

Optimized benchmarks **Macro benchmarks**

RQ2: Does refactoring play a role?

Classes	optimized benchmarks			macro-benchmarks				
	FA	KN	RC	XALAN	FOP	BATIK	COMMONS-IO	PGJDBC
`FileInputStream`	✖	☹	</>	</>	✖	✖	☹	✖
`BufferedInputStream`	✖	☹	+16.31%	-17.19%	✖	✖	☹	✖
`BufferedReader`	✖	</>	☹	☹	✖	✖	</>	✖
`LineNumberReader`	✖	-7.52%	☹	☹	✖	✖	-16.47%	✖
`Files.readAllLines`	✖	-0.87%	☹	☹	✖	✖	-9.5%	✖
`Scanner`	✖	+60.77%	☹	☹	✖	✖	+486.28%	✖
`PrintStream`	+19.13%	✖	+15.24%	-7.96%	+48.38%	-3.68%	✖	+811.41%
`FileOutputStream`	+29.13%	✖	</>	</>	+73.91%	</>	✖	☹
`DataOutputStream`	☹	✖	☹	☹	☹	☹	✖	+883.22%
`ByteArrayOutputStream`	+36.24%	✖	☹	☹	☹	☹	✖	☹
`BufferedOutputStream`	+19.23%	✖	+25.96%	-11.71%	</>	-3.1%	✖	</>
`System.out`	</>	✖	☹	☹	☹	☹	✖	☹

Default implementation

Not able to change

@gustavopinto

RQ2: Does refactoring play a role?

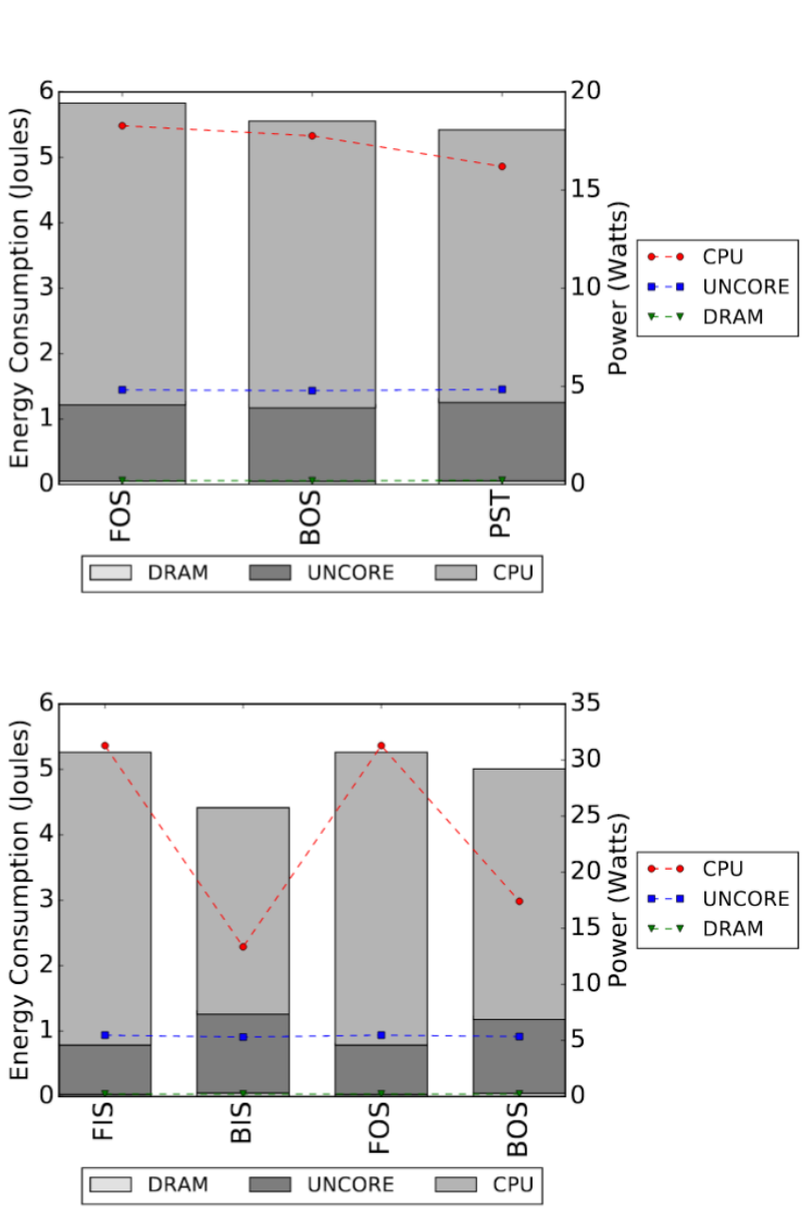
Classes	optimized benchmarks			macro-benchmarks				
	FA	KN	RC	XALAN	FOP	BATIK	COMMONS-IO	PGJDBC
`FileInputStream`	✖	☹	</>	</>	✖	✖	☹	✖
`BufferedInputStream`	✖	☹	+16.31%	-17.19%	✖	✖	☹	✖
`BufferedReader`	✖	</>	☹	☹	✖	✖	</>	✖
`LineNumberReader`	✖	-7.52%	☹	☹	✖	✖	-16.47%	✖
`Files.readAllLines`	✖	-0.87%	☹	☹	✖	✖	-9.5%	✖
`Scanner`	✖	+60.77%	☹	☹	✖	✖	+486.28%	✖
`PrintStream`	+19.13%	✖	+15.24%	-7.96%	+48.38%	-3.68%	✖	+811.41%
`FileOutputStream`	+29.13%	✖	</>	</>	+73.91%	</>	✖	☹
`DataOutputStream`	☹	✖	☹	☹	☹	☹	✖	+883.22%
`ByteArrayOutputStream`	+36.24%	✖	☹	☹	☹	☹	✖	☹
`BufferedOutputStream`	+19.23%	✖	+25.96%	-11.71%	</>	-3.1%	✖	</>
`System.out`	</>	✖	☹	☹	☹	☹	✖	☹

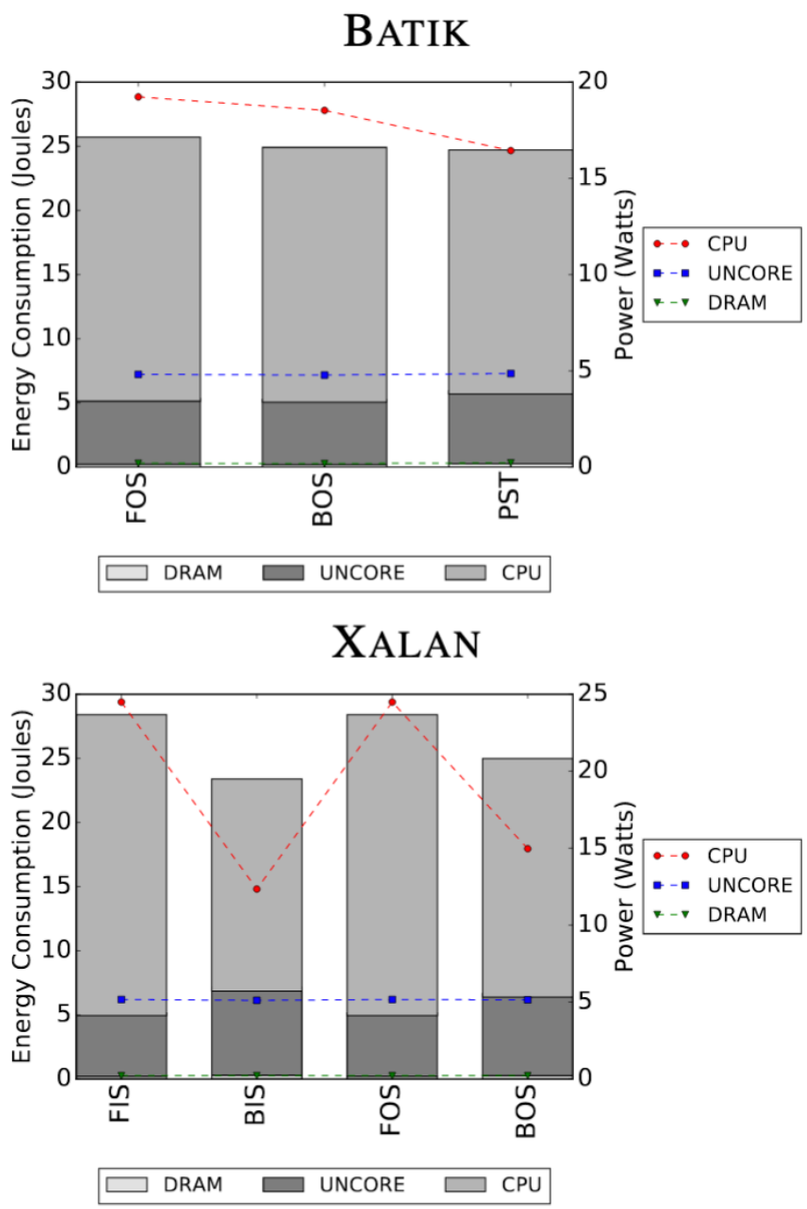
RQ2: Does refactoring play a role?

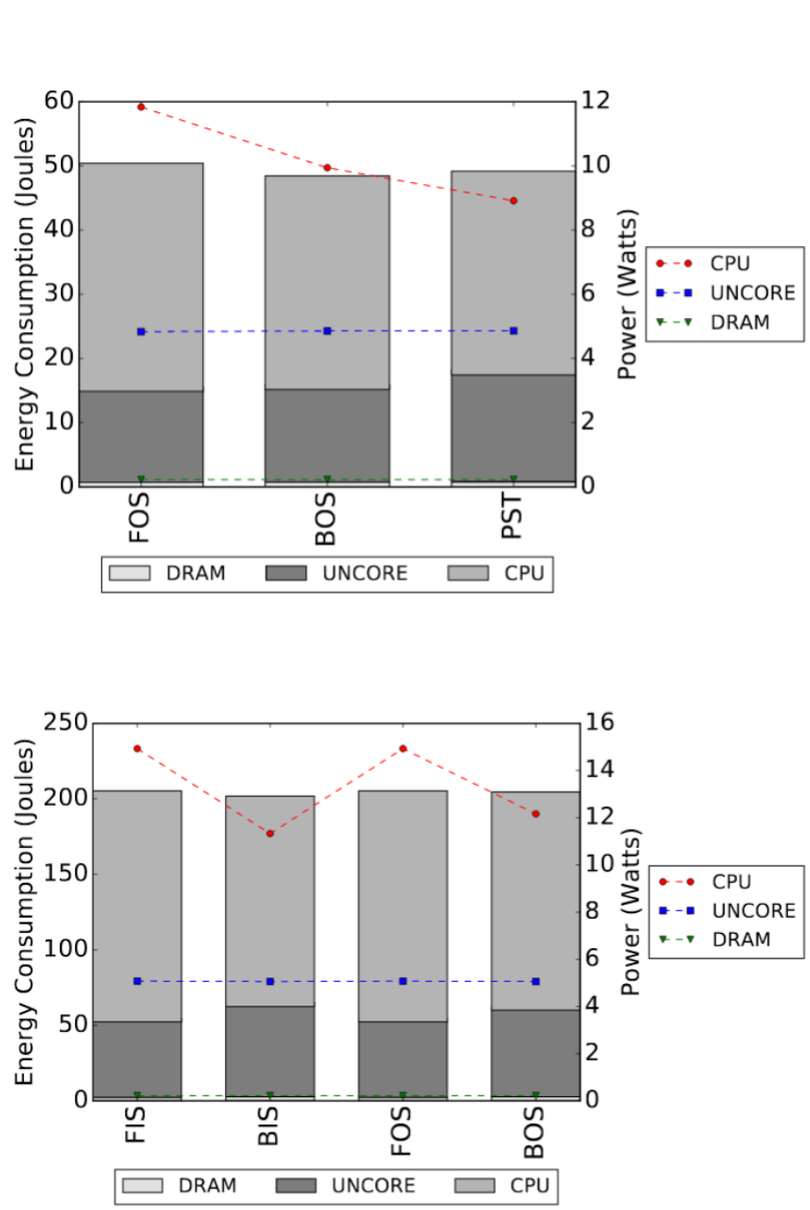
Classes	optimized benchmarks			macro-benchmarks				
	FA	KN	RC	XALAN	FOP	BATIK	COMMONS-IO	PGJDBC
`FileInputStream`	✖	☹	</>	</>	✖	✖	☹	✖
`BufferedInputStream`	✖	☹	+16.31%	-17.19%	✖	✖	☹	✖
`BufferedReader`	✖	</>	☹	☹	✖	✖	</>	✖
`LineNumberReader`	✖	-7.52%	☹	☹	✖	✖	-16.47%	✖
`Files.readAllLines`	✖	-0.87%	☹	☹	✖	✖	-9.5%	✖
`Scanner`	✖	+60.77%	☹	☹	✖	✖	+486.28%	✖
`PrintStream`	+19.13%	✖	+15.24%	-7.96%	+48.38%	-3.68%	✖	+811.41%
`FileOutputStream`	+29.13%	✖	</>	</>	+73.91%	</>	✖	☹
`DataOutputStream`	☹	✖	☹	☹	☹	☹	✖	+883.22%
`ByteArrayOutputStream`	+36.24%	✖	☹	☹	☹	☹	✖	☹
`BufferedOutputStream`	+19.23%	✖	+25.96%	-11.71%	</>	-3.1%	✖	</>
`System.out`	</>	✖	☹	☹	☹	☹	✖	☹

We improved the energy consumption in 36% of the cases (from 0.8% to 17%)

Does the input size matter?

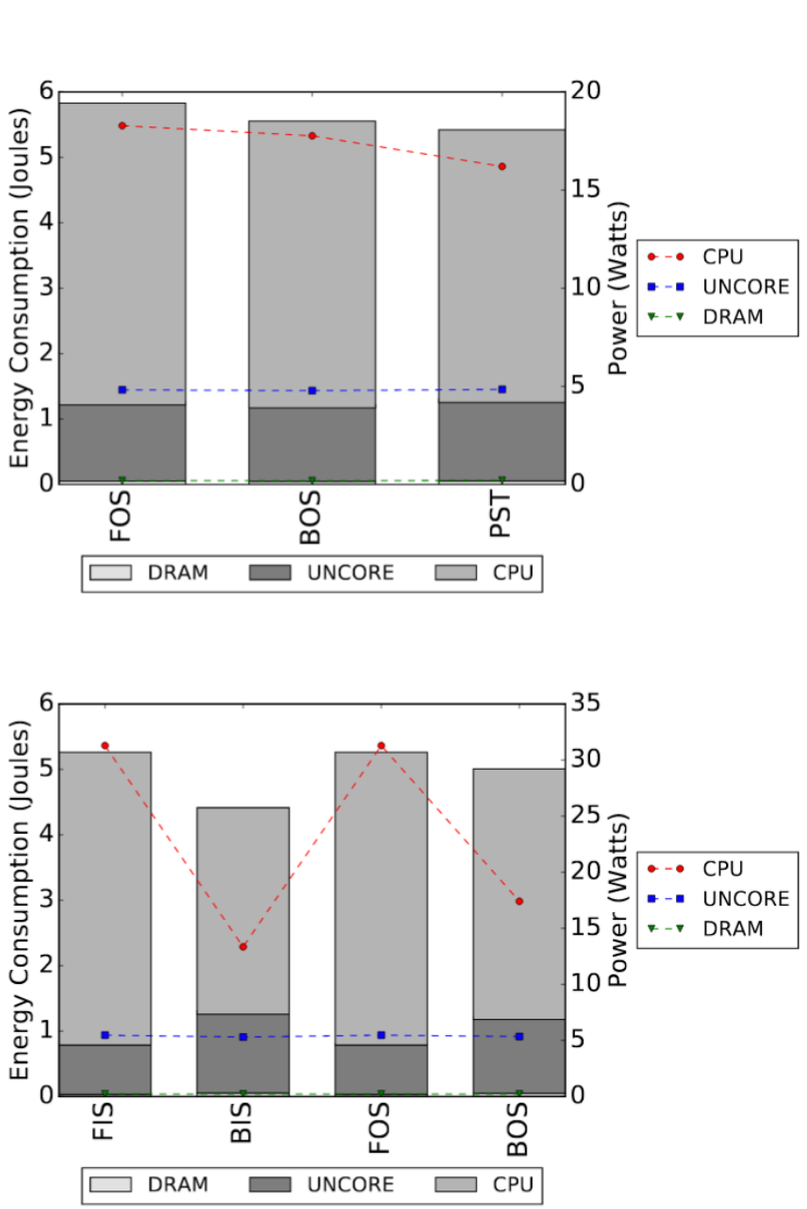


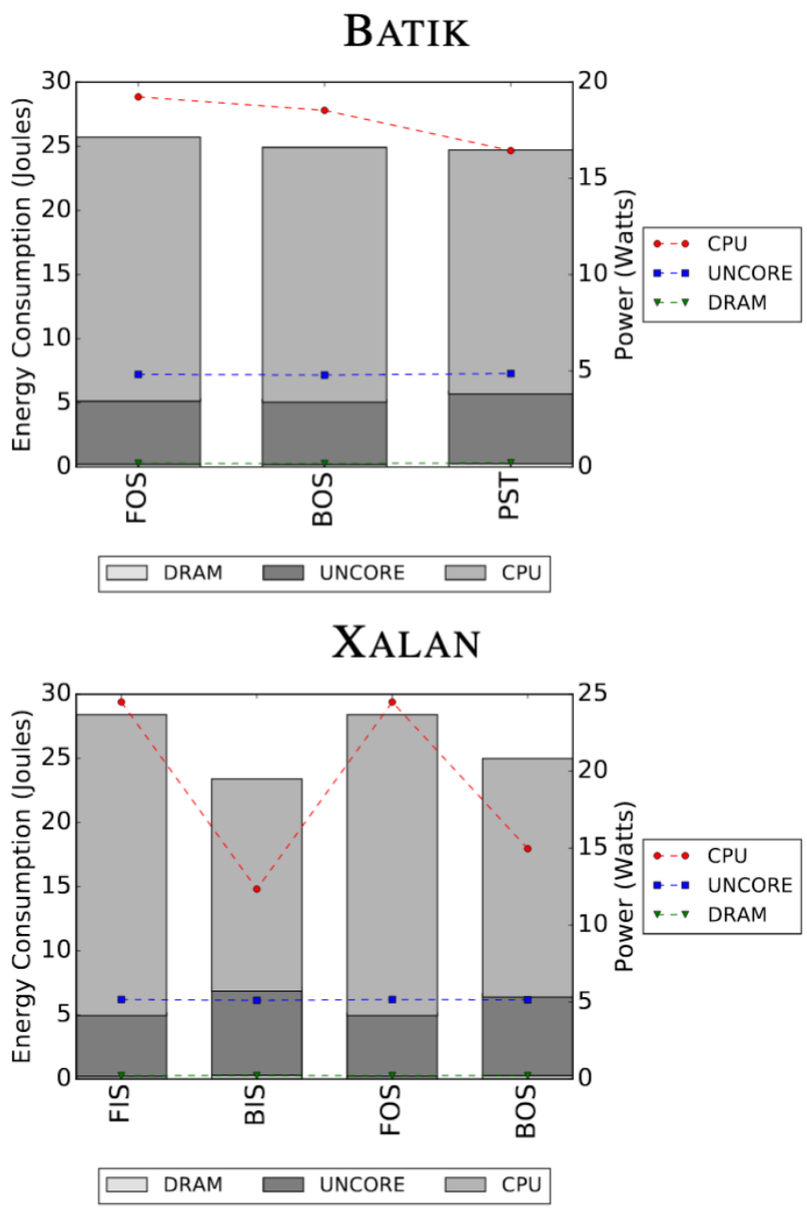


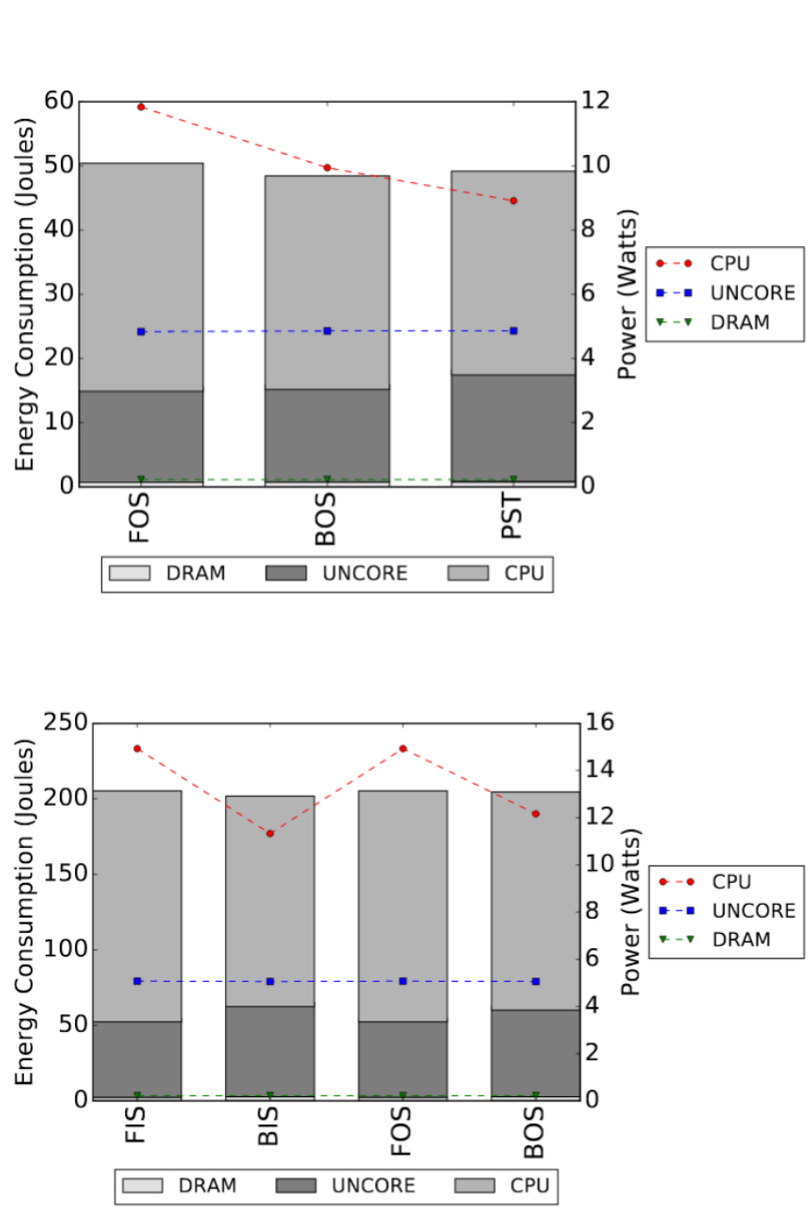


@gustavopinto

Does the input size matter?







Changing the input size did not change (much) the overall results

Source of Java I/O APIs

java.io

Class Writer

java.lang.Object
  java.io.Writer

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:

BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

```
public abstract class Writer
extends Object
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

java.io

Class Reader

java.lang.Object
  java.io.Reader

All Implemented Interfaces:

Closeable, AutoCloseable, Readable

Direct Known Subclasses:

BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1

java.io

Class InputStream

java.lang.Object
  java.io.InputStream

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:

JDK1.0

java.io

Class OutputStream

java.lang.Object
  java.io.OutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

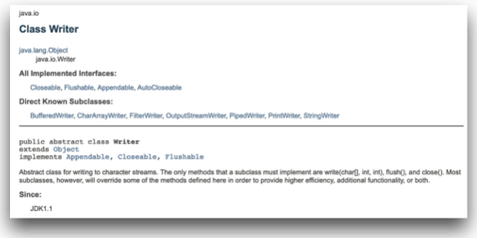
Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:

JDK1.0

@gustavopinto

Source of Java I/O APIs



java.io

Class Writer

java.lang.Object
  java.io.Writer

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:

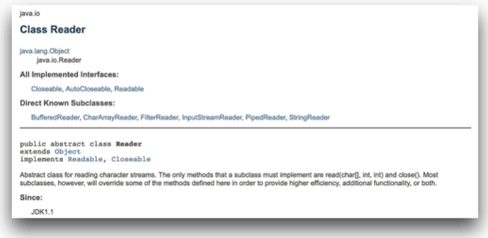
BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

```
public abstract class Writer
extends Object
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1



java.io

Class Reader

java.lang.Object
  java.io.Reader

All Implemented Interfaces:

Closeable, AutoCloseable, Readable

Direct Known Subclasses:

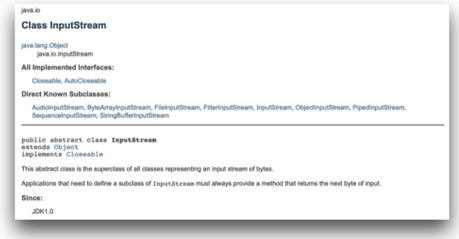
BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:

JDK1.1



java.io

Class InputStream

java.lang.Object
  java.io.InputStream

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:

JDK1.0

java.io

Class OutputStream

java.lang.Object
  java.io.OutputStream

All Implemented Interfaces:

Closeable, Flushable, AutoCloseable

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:

JDK1.0

@gustavopinto

22 Java IO APIs

Writer
BufferedWriter
FileWriter
StringWriter
PrintWriter
CharArrayWriter

Reader
BufferedReader
LineNumberReader
CharArrayReader
PushbackReader
FileReader
StringReader

OutputStream
FileOutputStream
ByteArrayOutputStream
BufferedOutputStream
PrintStream

InputStream
FileInputStream
BufferedInputStream
PushbackInputStream
ByteArrayInputStream

RandomAccessFile

Files

Scaner

@gustavopinto

25

Source of Java I/O APIs

java.io
Class Writer

java.lang.Object
  java.io.Writer

All Implemented Interfaces:
Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:
BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

```
public abstract class Writer
extends Object
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:
JDK1.1

java.io
Class Reader

java.lang.Object
  java.io.Reader

All Implemented Interfaces:
Closeable, AutoCloseable, Readable

Direct Known Subclasses:
BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:
JDK1.1

java.io
Class InputStream

java.lang.Object
  java.io.InputStream

All Implemented Interfaces:
Closeable, AutoCloseable

Direct Known Subclasses:
AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:
JDK1.0

java.io
Class OutputStream

java.lang.Object
  java.io.OutputStream

All Implemented Interfaces:
Closeable, Flushable, AutoCloseable

Direct Known Subclasses:
ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:
JDK1.0

22 Java IO APIs

Writer
BufferedWriter
FileWriter
StringWriter
PrintWriter
CharArrayWriter

Reader
BufferedReader
LineNumberReader
CharArrayReader
PushbackReader
FileReader
StringReader

OutputStream
FileOutputStream
ByteArrayOutputStream
BufferedOutputStream
PrintStream

InputStream
FileInputStream
BufferedInputStream
PushbackInputStream
ByteArrayInputStream

RandomAccessFile

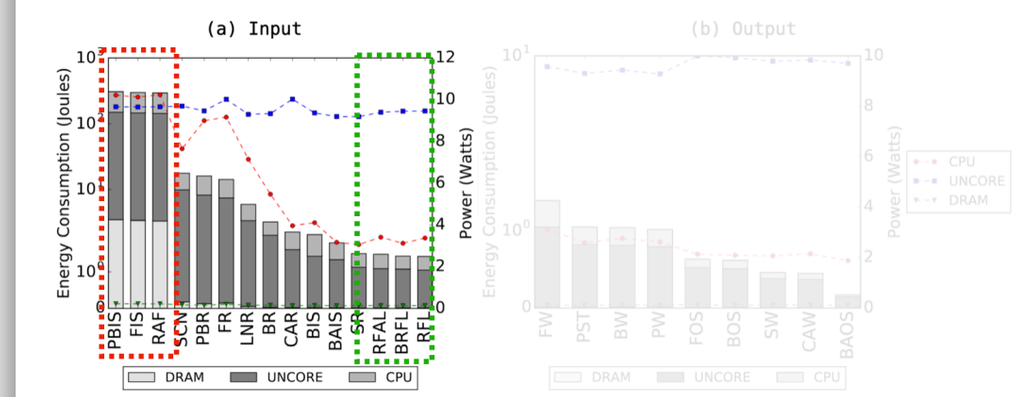
Files

Scaner

@gustavopinto

25

RQ1: Energy behaviors



PBIS: PushBackInputStream
FIS: FileInputStream
RAF: RandomAccessFile

RFAL: Files.readAllLines()
BRFL: Files.newBufferedReader()
RFAL: Files.lines()

@gustavopinto

48

Source of Java I/O APIs

java.io
Class Writer

java.lang.Object
  java.io.Writer

All Implemented Interfaces:
Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:
BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

```
public abstract class Writer
extends Object
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:
JDK1.1

java.io
Class Reader

java.lang.Object
  java.io.Reader

All Implemented Interfaces:
Closeable, AutoCloseable, Readable

Direct Known Subclasses:
BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:
JDK1.1

java.io
Class InputStream

java.lang.Object
  java.io.InputStream

All Implemented Interfaces:
Closeable, AutoCloseable

Direct Known Subclasses:
AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:
JDK1.0

java.io
Class OutputStream

java.lang.Object
  java.io.OutputStream

All Implemented Interfaces:
Closeable, Flushable, AutoCloseable

Direct Known Subclasses:
ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:
JDK1.0

22 Java IO APIs

Writer
BufferedWriter
FileWriter
StringWriter
PrintWriter
CharArrayWriter

Reader
BufferedReader
LineNumberReader
CharArrayReader
PushbackReader
FileReader
StringReader

OutputStream
FileOutputStream
ByteArrayOutputStream
BufferedOutputStream
PrintStream

InputStream
FileInputStream
BufferedInputStream
PushbackInputStream
ByteArrayInputStream

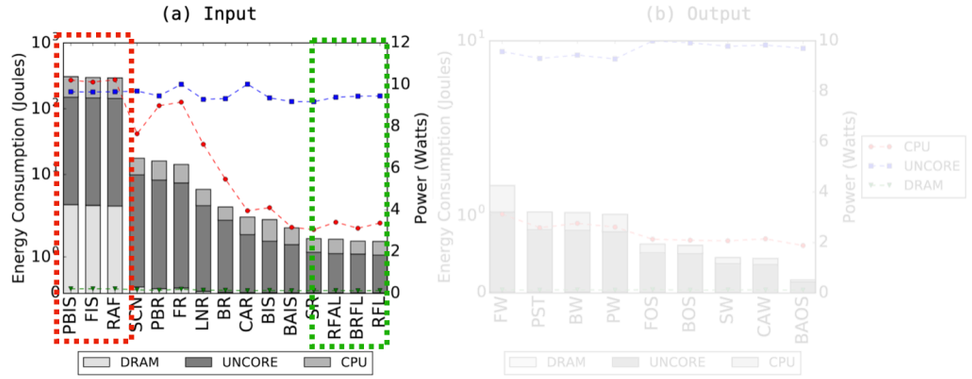
RandomAccessFile

Files

Scanner

25

RQ1: Energy behaviors



PBIS: PushBackInputStream
FIS: FileInputStream
RAF: RandomAccessFile

RFAL: Files.readAllLines()
BRFL: Files.newBufferedReader()
RFAL: Files.lines()

@gustavopinto

48

RQ2: Does refactoring play a role?

Classes	optimized benchmarks			macro-benchmarks				
	FA	KN	RC	XALAN	FOP	BATIK	COMMONS-IO	PGJDBC
FileInputStream	✖	☹	</>	</>	✖	✖	☹	✖
BufferedInputStream	✖	☹	+16.31%	-17.19%	✖	✖	☹	✖
BufferedReader	✖	</>	☹	☹	✖	✖	</>	✖
LineNumberReader	✖	-7.52%	☹	☹	✖	✖	-16.47%	✖
Files.readAllLines	✖	-0.87%	☹	☹	✖	✖	-9.5%	✖
Scanner	✖	+60.77%	☹	☹	✖	✖	+486.28%	✖
PrintStream	+19.13%	✖	+15.24%	-7.96%	+48.38%	-3.68%	✖	+811.41%
FileOutputStream	+29.13%	✖	</>	</>	+73.91%	</>	✖	☹
DataOutputStream	☹	✖	☹	☹	☹	☹	✖	+883.22%
ByteArrayOutputStream	+36.24%	✖	☹	☹	☹	☹	✖	☹
BufferedOutputStream	+19.23%	✖	+25.96%	-11.71%	</>	-3.1%	✖	</>
System.out	</>	✖	☹	☹	☹	☹	✖	☹

We improved the energy consumption in 36% of the cases (from 0.8% to 17%)

@gustavopinto

54

Source of Java I/O APIs

java.io
Class Writer

java.lang.Object
  java.io.Writer

All Implemented Interfaces:
Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:
BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter

```
public abstract class Writer
extends Object
implements Appendable, Closeable, Flushable
```

Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:
JDK1.1

java.io
Class Reader

java.lang.Object
  java.io.Reader

All Implemented Interfaces:
Closeable, AutoCloseable, Readable

Direct Known Subclasses:
BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

```
public abstract class Reader
extends Object
implements Readable, Closeable
```

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

Since:
JDK1.1

java.io
Class InputStream

java.lang.Object
  java.io.InputStream

All Implemented Interfaces:
Closeable, AutoCloseable

Direct Known Subclasses:
AudioInputStream, ByteArrayInputStream, FileInputStream, FilterInputStream, InputStream, ObjectInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream

```
public abstract class InputStream
extends Object
implements Closeable
```

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of `InputStream` must always provide a method that returns the next byte of input.

Since:
JDK1.0

java.io
Class OutputStream

java.lang.Object
  java.io.OutputStream

All Implemented Interfaces:
Closeable, Flushable, AutoCloseable

Direct Known Subclasses:
ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

```
public abstract class OutputStream
extends Object
implements Closeable, Flushable
```

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one byte of output.

Since:
JDK1.0

22 Java IO APIs

Writer
BufferedWriter
FileWriter
StringWriter
PrintWriter
CharArrayWriter

Reader
BufferedReader
LineNumberReader
CharArrayReader
PushbackReader
FileReader
StringReader

OutputStream
FileOutputStream
ByteArrayOutputStream
BufferedOutputStream
PrintStream

InputStream
FileInputStream
BufferedInputStream
PushbackInputStream
ByteArrayInputStream

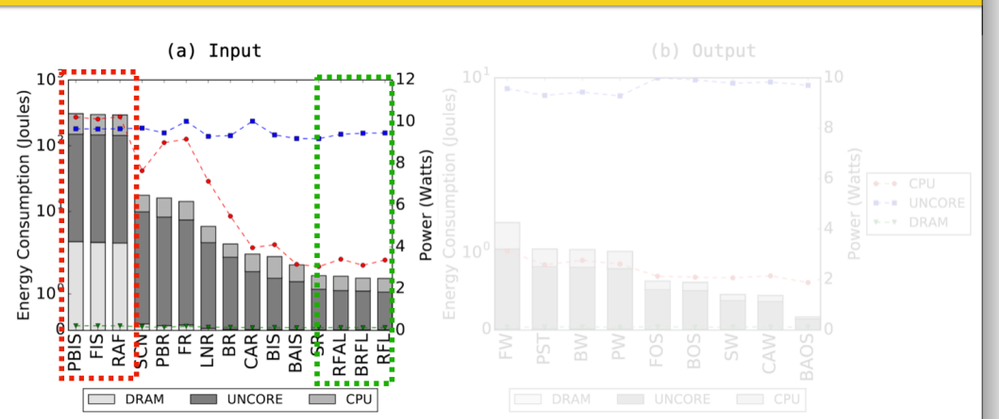
RandomAccessFile

Files

Scanner

25

RQ1: Energy behaviors



PBIS: PushBackInputStream
FIS: FileInputStream
RAF: RandomAccessFile

RFAL: Files.readAllLines()
BRFL: Files.newBufferedReader()
RFAL: Files.lines()

@gustavopinto

48

RQ2: Does refactoring play a role?

Classes	optimized benchmarks			macro-benchmarks				
	FA	KN	RC	XALAN	FOP	BATIK	COMMONS-IO	PGJDBC
FileInputStream	✖	☹	</>	</>	✖	✖	☹	✖
BufferedInputStream	✖	☹	+16.31%	-17.19%	✖	✖	☹	✖
BufferedReader	✖	</>	☹	☹	✖	✖	</>	✖
LineNumberReader	✖	-7.52%	☹	☹	✖	✖	-16.47%	✖
Files.readAllLines	✖	-0.87%	☹	☹	✖	✖	-9.5%	✖
Scanner	✖	+60.77%	☹	☹	✖	✖	+486.28%	✖
PrintStream	+19.13%	✖	+15.24%	-7.96%	+48.38%	-3.68%	✖	+811.41%
FileOutputStream	+29.13%	✖	</>	</>	+73.91%	</>	✖	☹
DataOutputStream	☹	✖	☹	☹	☹	☹	✖	+883.22%
ByteArrayOutputStream	+36.24%	✖	☹	☹	☹	☹	✖	☹
BufferedOutputStream	+19.23%	✖	+25.96%	-11.71%	</>	-3.1%	✖	</>
System.out	</>	✖	☹	☹	☹	☹	✖	☹

We improved the energy consumption in 36% of the cases (from 0.8% to 17%)

@gustavopinto

54

Email
Twitter
mail@gustavopinto.org
Web site