Java Thread Affinity library

When you need stable performance
from a few threads.

Peter Lawrey, 2012

Tuning the system - Isolating cores

Intel i7 with 4 cores and hyper-threading.

All Cores scheduled

	Thread 0	Thread 1
Core 0	CPU 0	CPU 4
Core 1	CPU 1	CPU 5
Core 2	CPU 2	CPU 6
Core 3	CPU 3	CPU 7

Cores isolated

	Thread 0	Thread 1
Core 0	CPU 0	CPU 4
Core 1	CPU 1	CPU 5
Core 2	CPU 2	CPU 6
Core 3	CPU 3	CPU 7

Using the library to allocate threads

```
Assigning cpu 7 to Thread[main,5,main]
Assigning cpu 6 to Thread[reader,5,main]
Assigning cpu 3 to Thread[writer,5,main]
Releasing cpu 7 from Thread[main,5,main]
Assigning cpu 7 to Thread[engine,5,main]

The assignment of CPUs is
0: General use CPU
1: General use CPU
2: Reserved for this application
3: Thread[writer,5,main] alive=true
4: General use CPU
5: General use CPU
6: Thread[reader,5,main] alive=true
7: Thread[engine,5,main] alive=true

Releasing cpu 6 from Thread[reader,5,main]
Releasing cpu 3 from Thread[writer,5,main]
Releasing cpu 7 from Thread[engine,5,main]
```

The high level contract

```
public static void main(String... args) throws InterruptedException {
    AffinityLock al = AffinityLock.acquireLock();
    try {
        new Thread(new SleepRunnable(), "reader").start();
        new Thread(new SleepRunnable(), "writer").start();
        Thread.sleep(200);
    } finally {
        al.release();
    }
    new Thread(new SleepRunnable(), "engine").start();

    Thread.sleep(200);
    System.out.println("\nThe assignment of CPUs is\n" + AffinityLock.dumpLocks());
}

private static class SleepRunnable implements Runnable {
    public void run() {
        AffinityLock al = AffinityLock.acquireLock();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        } finally {
            al.release();
        }
    }
}
```

Tuning the system - Determining the layout.

Intel i7 with 4 cores and hyper-threading.
"reader" and "writer" to be on the same core.
"engine" to be on a core of its own.

	Thread 0	Thread 1
Core 0	CPU 0 General Use	CPU 4 General Use
Core 1	CPU 1 General Use	CPU 5 General Use
Core 2	CPU 2 writer thread	CPU 6 reader thread
Core 3	CPU 3 engine thread	CPU 7 engine thread*

Allocating threads and cores

```
Assigning cpu 7 to Thread[main,5,main]
Assigning cpu 6 to Thread[reader,5,main]
Assigning cpu 2 to Thread[writer,5,main]
Releasing cpu 7 from Thread[main,5,main]
Assigning core 3: cpus 3, 7 to Thread[engine,5,main]

The assignment of CPUs is
0: General use CPU
1: General use CPU
2: Thread[writer,5,main] alive=true
3: Thread[engine,5,main] alive=true
4: General use CPU
5: General use CPU
6: Thread[reader,5,main] alive=true
7: Thread[engine,5,main] alive=true

Releasing cpu 6 from Thread[reader,5,main]
Releasing cpu 2 from Thread[writer,5,main]
Releasing cpu 3 from Thread[engine,5,main]
Releasing cpu 7 from Thread[engine,5,main]
```

The high level contract, locks and cores

```
public static void main(String... args) throws InterruptedException {
    AffinityLock al = AffinityLock.acquireLock();
    try {
        // find a cpu on a different socket, otherwise a different core.
        AffinityLock readerLock = al.acquireLock(DIFFERENT_SOCKET, DIFFERENT_CORE);
        new Thread(new SleepRunnable(readerLock, false), "reader").start();
        // find a cpu on the same core, or the same socket, or any free cpu.
        AffinityLock writerLock = readerLock.acquireLock(SAME_CORE, SAME_SOCKET, ANY);
        new Thread(new SleepRunnable(writerLock, false), "writer").start();
        Thread.sleep(200);
    } finally {
        al.release();
    }
    // allocate a whole core to the engine so it doesn't have to compete for resources.
    al = AffinityLock.acquireCore(false);
    new Thread(new SleepRunnable(al, true), "engine").start();

    Thread.sleep(200);
    System.out.println("\nThe assignment of CPUs is\n" + AffinityLock.dumpLocks());
}
```

The high level contract, locks and cores

```
static class SleepRunnable implements Runnable {
    private final AffinityLock affinityLock;
    private final boolean wholeCore;

    SleepRunnable(AffinityLock affinityLock, boolean wholeCore) {
        this.affinityLock = affinityLock;
        this.wholeCore = wholeCore;
    }

    public void run() {
        affinityLock.bind(wholeCore);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        } finally {
            affinityLock.release();
        }
    }
}
```

The underlying contract

```
public interface IAffinity {
    /**
* @return returns affinity mask for current thread
*/
    public long getAffinity();

    /**
* @param affinity sets affinity mask of current thread to specified value
*/
    public void setAffinity(final long affinity);
}
```

JNI Implementation (Java)

```
public enum NativeAffinity implements IAffinity {
    INSTANCE;

    public static final boolean LOADED;
    private static final Logger LOGGER = Logger.getLogger(NativeAffinity.class.getName());

    static {
        boolean loaded;
        try {
            System.loadLibrary("affinity");
            loaded = true;
        } catch (UnsatisfiedLinkError ule) {
            if (LOGGER.isLoggable(Level.FINE))
                LOGGER.fine("Unable to find libaffinity in "
+ System.getProperty("java.library.path") + " " + ule);
            loaded = false;
        }
        LOADED = loaded;
    }

    private native static long getAffinity0();

    private native static void setAffinity0(long affinity);

    @Override
    public long getAffinity() { return getAffinity0(); }

    @Override
    public void setAffinity(long affinity) { setAffinity0(affinity); }
}
```

JNI Implementation (C)

```
/*
* Class: vanilla_java_affinity_impl_NativeAffinity
* Method: getAffinity0
* Signature: ()J
*/
JNIEXPORT jlong JNICALL Java_vanilla_java_affinity_impl_NativeAffinity_getAffinity0
  (JNIEnv *env, jclass c) {
    cpu_set_t mask;
    int ret = sched_getaffinity(0, sizeof(mask), &mask);
    if (ret < 0) return ~0LL;
    long long mask2 = 0, i;
    for(i=0;i<sizeof(mask2)*8;i++)
        if (CPU_ISSET(i, &mask))
            mask2 |= 1L << i;
    return (jlong) mask2;
}
/*
* Class: vanilla_java_affinity_NativeAffinity
* Method: setAffinity0
* Signature: (J)V
*/
JNIEXPORT void JNICALL Java_vanilla_java_affinity_impl_NativeAffinity_setAffinity0
  (JNIEnv *env, jclass c, jlong affinity) {
    int i;
    cpu_set_t mask;
    CPU_ZERO(&mask);
    for(i=0;i<sizeof(affinity)*8;i++)
        if ((affinity >> i) & 1)
            CPU_SET(i, &mask);
    sched_setaffinity(0, sizeof(mask), &mask);
}
```

JNA Implementation

```
public enum PosixJNAAffinity implements IAffinity {
    INSTANCE;
    private static final Logger LOGGER = Logger.getLogger(PosixJNAAffinity.class.getName());

    public static final boolean LOADED;

    private static final String LIBRARY_NAME = "c";

    /**
* @author BegemoT
*/
    private interface CLibrary extends Library {
        public static final CLibrary INSTANCE = (CLibrary)
                Native.loadLibrary(LIBRARY_NAME, CLibrary.class);

        public int sched_setaffinity(final int pid,
                                     final int cpusetsize,
                                     final PointerType cpuset) throws LastErrorException;

        public int sched_getaffinity(final int pid,
                                     final int cpusetsize,
                                     final PointerType cpuset) throws LastErrorException;
    }
```

JNA Implementation (cont)

```
    @Override
    public long getAffinity() {
        final CLibrary lib = CLibrary.INSTANCE;
        // TODO where are systems with 64+ cores...
        final LongByReference cpuset = new LongByReference(0L);
        try {
            final int ret = lib.sched_getaffinity(0, Long.SIZE / 8, cpuset);
            if (ret < 0)
                throw new IllegalStateException("sched_getaffinity((" + Long.SIZE / 8 + ") , &(" + cpuset +
") ) return " + ret);
            return cpuset.getValue();
        } catch (LastErrorException e) {
            throw new IllegalStateException("sched_getaffinity((" + Long.SIZE / 8 + ") , &(" + cpuset + ") )
errorNo=" + e.getErrorCode(), e);
        }
    }

    @Override
    public void setAffinity(final long affinity) {
        final CLibrary lib = CLibrary.INSTANCE;
        try {
            //fixme: where are systems with more then 64 cores...
            final int ret = lib.sched_setaffinity(0, Long.SIZE / 8, new LongByReference(affinity));
            if (ret < 0) {
                throw new IllegalStateException("sched_setaffinity((" + Long.SIZE / 8 + ") , &(" + affinity
+ ") ) return " + ret);
            }
        } catch (LastErrorException e) {
            throw new IllegalStateException("sched_getaffinity((" + Long.SIZE / 8 + ") , &(" + affinity + ")
) errorNo=" + e.getErrorCode(), e);
        }
    }
```

Using affinity with one thread

BusyAccountingMain - transfers money between accounts with changes persisted to disk.
Without affinity.
Throughput was **1.630** million transfers per second
Latencies 50/avg/99/99.9/99.99%tile were 540/560/**1,140/3,440/5,310** us
Throughput was **1.689** million transfers per second
Latencies 50/avg/99/99.9/99.99%tile were 540/561/**1,140/3,500/5,050** us
Throughput was **1.686** million transfers per second
Latencies 50/avg/99/99.9/99.99%tile were 540/561/**1,210/3,560/5,420** us

With thread affinity.
Throughput was **1.695** million transfers per second
Latencies 50/avg/99/99.9/99.99%tile were 540/559/**850/3,370/4,580** us
Throughput was **1.696** million transfers per second
Latencies 50/avg/99/99.9/99.99%tile were 540/558/**920/3,380/4,560** us
Throughput was **1.700** million transfers per second
Latencies 50/avg/99/99.9/99.99%tile were 540/557/**840/3,350/4,450** us

Note: You can get much greater improvements when you have tight interactions between multiple threads. (Example to follow)

Java Thread Affinity Library

The Java Thread Affinity Library on GitHub
https://github.com/peter-lawrey/Java-Thread-Affinity

Example: Busy Accounting
https://github.com/peter-lawrey/Example-Busy-Accounting

Vanilla #Java blog

http://vanillajava.blogspot.com/