Requirements

- ASM Bytecode Instrumentation Library 3.3:
  - http://www.cs.tufts.edu/research/redline/elephantTracks/asm-all-3.3.jar
- IBM's thread building blocks (including header files):
  - http://threadingbuildingblocks.org/download#stable-releases

Building Elephant Tracks

1. First, copy Makefile.inc to Makefile.local; in Makefile.local you will set local build parameters for your machine, in particular, you must set the following
   - INSTALL_DIR -- Where you would like the Elephant Tracks binaries be installed
   - ASMJAR -- Location of the asm.jar (asm version > 3.3)
   - JAVA_PATH -- Must point to the home directory for java, not the actually java binary. (ie, the 'java' binary should be in JAVA_PATH/bin/java)
2. make; make install

Known Issue

There is disagreement between different versions of jni.h on the const-ness of certain parameters. This can lead to the following error when attempting to build Elephant Tracks:

ETCallBackHandler.cpp: In function '_jobject* instNewObjectA(JNIEnv*, jclass, jmethodID, const jvalue*)':
ETCallBackHandler.cpp:1385:66: warning: invalid conversion from 'const jvalue*' to 'jvalue*'

This can be resolved by removing the -Werror g++ flag,
and adding -fpermissive.

Running Elephant Tracks

First, make sure that INSTALL_DIR is in the LD_LIBRARY_PATH

Then run any Java program with Elephant Tracks like so:

```
java -classpath <other-paths>:$ASMJAR -Xbootclasspath/a:$INSTALL_DIR \
-agentlib:ElephantTracks=<ElephantTracks Options>
```

Options:

Are represented as name=value pairs, and may be given on the command like so:

```
-agentlib:ElephantTracks=:=name1=value1@name2=value2@...@nameN=valueN
```

Note that JVMs often impose an (undocumented) limit on the length of the command strings passed to a JVMTI agent such as Elephant Tracks, and will silently truncate if this is limit is exceeded. For this reason, it is recommend that infrequently changing options are stored in an option file, and specify that option file on the command line (see below).

- optionsFile=<option file>
  - This is a path to a file containing options (the same as may be passed on the command line), one option per line.
- classPath=<path>

  This is the path ElephantTracks will use to start its own Java process (not the one running your program). It must include INSTALL_DIR, INSTALL_DIR/elephantTracksRewriter.jar, and the asm-3.3 jar file.

- javaPath=<path to java executable>
  - This is the path to the actual java binary, not merely the directory
    it is in. (for example, /usr/bin/java, not /usr/bin).
- namesFile=<file name>
  - The file in which to output the names information (see below)
- traceFile=<file name>
  - The file in which to output the trace. You may also redirect trace output to a shell command with this syntax:

traceFile=>(shell command)

For example:

traceFile=>(gzip > foo.trace.gz)

Would pipe the trace output to gzip, and redirect the output from gzip to foo.trace.gz

- bufferSize=<number>
  - How many records to hold in Elephant Tracks' internal buffer; larger values generally give better performance, but will use more memory (approximately 40 bytes per record).

Record Types

Names File Records

The entries in the names file map names of methods to numerical ids used in the trace.

Comment

Lines that start with # are comments.

C 0xcccc class-name [0xssss] (I:0xiiii)*

cccc = class id; ssss = superclass id;
iiii = superinterface id

I 0xiiii interface-name [0xssss] (I:0xiiii)*

C or I will be output when the id is assigned, which is on first *mention*, and will be output again, possibly with more info, such as superclass / superinterfaces, when *processed*. We could perhaps distinguish a mention from a definition if you think it would be easier ...

E 0xcccc class-name

marks end of processing a class/interface *definition*

(so now you know all the methods and fields)

F I/S 0xffff name 0xcccc class-name field-type
I = instance; S = static
ffff = field id; cccc = declaring class id
field-type is a descriptor (I, etc., or Jclassname; etc.)

N 0xmmmm 0xcccc class-name method-name descriptor flags

mmmm = method id; cccc = class id
flags can include I or S for instance or static,
with N added if native

S 0xmmmm 0xcccc 0xssss descriptor dims

For allocation sites
mmmm = method id; cccc = declaring class id;
ssss = site id (unique in run)
descriptor = type of thing allocated
dims = number of dimensions (0 for a scalar object)

Trace File Records

The entries in the trace file represent events in the program execution.

Comment

Lines that start with # are comments.

Allocation:

There are several different kinds of allocation record, based on how the object was allocated. They all have the same general form:

<Type Character> <object-id> <size> <type> <site> <length> <thread-id>

Types:

A - Array allocation
I - Initial Heap Allocation (Allocated before Elephant Tracks started)
N - Allocated via the NEW byte code in the traced java program

P - Preexisting object; these are objects for which Elephant Tracks missed the actual object allocation, but discovered later.This can be due to objects from the constant pool, or VM bugs.

V - Objects allocated by the virtual machine.

The new object has ID object-id, which is used to refer to the object in later events; the size in bytes; the type (a Java type as a string); length field is 0 for non-arrays, and the length of the array for arrays, the ID of the allocating thread.

The size is the size of the object in bytes as reported by the VM (this includes object headers and possibly other VM structures).

Death:
D <object-id> <thread-id>

Object-id died, and its death occurred in thread_id.

Field update:
U <old-target-id> <object-id> <new-target-id> <field-id> <thread-id>

The field field-id in object object-id that used to point to old-target-id now points at new-target-id, and this update occurred in thread-id.

An object-id of 0 indicates that this is an update to a static field.

Method entry:
M <method-id> <receiver-object-id>

<thread-id>

A call to the method method-id with receiver object receiver-object-id in thread thread-id.

A origin receiver of 0 indicates that this was a static method

Method exit:
E <method-id> <receiver-object-id> <thread-id>

Return from method method-id with receiver object receiver-object-id in thread thread-id.

A receiver of 0 indicates this was a static method.

X <method-id> <receiver-object-id> <exception-id> <thread-id>

Exceptional exit from a method method-id, with receiver object receiver-object-id, exception object exception-id, in thread thread-id

Exception Throw

T <method-id> <receiver-id> <exception-object-id> <thread-id>

An exception was thrown in the given method, with the given receiver, and with the exception object itself having the given id.

Exception Handled

H <method-id> <receiver-id>

<exception-object-id> <thread-id>

An exception was handled in the given method, with the given receiver, and with the exception object itself having the given id.

R <root object ID> <thread id>

Object-ID was a root in thread-id