The HSAIL Simulator

The HSA Foundation has open sourced the following repositories on github

- libHSAIL (tools for parsing, assembling and disassembling HSAIL)
- an HSAIL Instruction Set Simulator
- OKRA interface to the HSAIL Simulator

The HSAIL-enabled Sumatra can run using the OKRA interface to the HSAIL simulator. Similarly, graal with the new HSAIL backend can run using this OKRA interface. For graal, you can either

- use a pre-built OKRA and simulator that is part of the graal libraries (found in graal/lib/okra-1.x-with-sim.jar). This prebuilt simulator allows junit tests to be run against the simulator without any additional okra build steps or path setup.
- or you can build the simulator with its OKRA interface using the instructions below and then add the build ouput directory to your PATH and LD_LIBRARY_PATH. By setting up the PATH and LD_LIBRARY_PATH, you can control whether you use a simulator-based OKRA or a hardware-based OKRA. Note that if no okra library can be found in the PATH or LD_LIBRARY_PATH a pre-built simulator OKRA described above is used. While this option involves more setup, it will have to be used eventually when you move to a hardware based OKRA.

Building the Simulator OKRA from sources

The HSAIL simulator and associated OKRA interface are currently limited to Linux. The build process for the OKRA interface will build both the simulator and the assembler from libHSAIL which is used to create BRIG binaries for the simulator. To build the simulator with its OKRA interface, follow the directions at https://github.com/HSAFoundation/Okra-Interface-to-HSAIL-Simulator#okra-interface-to-hsail-simulator. This procedure will build the assembler, simulator and okra interface. Remember to add okra/dist/bin from this build to PATH and to LD_LIBRARY_PATH. Run the indicated OKRA sanity tests on that page to confirm a proper build.

The sanity tests above are small tests with hand-coded HSAIL. They test both the C++ interface and the Java JNI interface. The following are a few other clients that can be run against the Java OKRA interface to the HSAIL Simulator.

Running the Graal Junit Tests on the Simulator

The graal trunk supports an HSAIL backend.

You can run junit tests in the graal HSAIL-enabled backend by doing the following steps. Your JAVA_HOME can be either JDK7 or a recent JDK8 binary.

- hg clone http://hg.openjdk.java.net/graal/graal
- cd graal (or whatever you named the clone you just made)
- Build graal using the command "mx --vm server --vmbuild product build".
- The following command runs a single junit test (the one that is described in Vasanth's HSAIL compiler blog). The command should be run on a single line:

```
    mx --vm server unittest -XX:-UseGraalClassLoader -G:Log=CodeGen,~HostGraph,~Stub hsail.test.
IntSquaredTest
```

- Note: the -G:Log=CodeGen is optional but can be used if you want to see the generated HSAIL code on stdout. Further debug logging can be enabled with -XX:+TraceGPUInteraction

- By removing the specifier IntSquaredTest from the above command line, you can run all the junit tests in the hsail.test directory. You can experiment with different -Xmx heap sizes which will lead to different compressed reference encodings and see that these are handled in the HSAIL code. (You can also turn off compressed oops if desired).

A note about the Graal Junit Tests

Each graal junit test specifies a java method which takes either:

- a final int argument which is treated as a "workitemId". Also specified is a Range from 0 to N over which the workitemId will iterate. Most of the tests fall into this category.
- a final Object argument. In this case an array or ArrayList of Objects is supplied. Each workitem will receive a different Object from the array or list. The "workitem range" in this case is over the length of the array or list.

In either case, the method under test is dispatched twice:

- once using straight java on the host machine, sequentially, once for each workitemid in the Range.
- in a separate class instance, the method under test is compiled to HSAIL and the resulting kernel is dispatched using OKRA across the specified Range (with the library paths specified above this will be executed on the simulator).

The fields marked with @Result from the two dispatch instances are then compared for equality.

Internally, the junit tests force a few graal options to needed values using graal's "scoped options" feature:

- RemoveNeverExecutedCode is turned off because the current HSAIL backend cannot handle the deoptimization that would be required when a previously never executed path is then taken.
- InlineEverything is turned on used because the HSAIL backend's support for function calls is still in progress.

Stepping thru the HSAIL code in gdb

The HSAIL simulator supports debugging of the HSAIL source with gdb. For more details see the section "GDB-Based Debugging Interface" in the HSAIL Simulator README. In the particular context of debugging HSAIL generated for the graal junit tests, note that the kernel entry point is always called &run, so a kernel entry breakpoint can be set with "break run". The hsail source file will always be temp_hsa.hsail, which is a temporary file created by the OKRA interface. You will be able to step by HSAIL instructions, and inspect HSAIL registers as described in the README. Be sure to set the environment variables SIMNOOPT=1 and SIMTHREADS=1 as described there. In addition, set the environment variable OKRA_SAVEHSAILSOURCE=1. This prevents the normal action which is to delete the hsail file after it is converted to BRIG. This allows the file to be displayed as a source file by gdb.

Building and Running a Sumatra JDK to offload Stream.forEach lambdas thru HSAIL

A Sumatra JDK can be built which is then used to build a Graal Server JVM. This combination can then be used to offload certain JDK 8 Stream API parallel streams terminating in forEach() to HSA APU/GPUs or the HSAIL Simulator. Follow the instructions at Sumatra JDK build instructions.