HiPE

HiPE 3.17.1
March 26, 2018

March 26, 2018

1 Reference Manual

HiPE

Application

The normal way to native-compile an Erlang module using HiPE is to include the atom native in the Erlang compiler options, as in:

```
1> c(my_module, [native]).
```

Options to the HiPE compiler are then passed as follows:

```
1> c(my_module, [native,{hipe,Options}]).
```

For on-line help in the Erlang shell, call `hipe:help()`. Details on HiPE compiler options are given by `hipe:help_options()`.

Feature Limitations

The HiPE compiler is in general compliant with the normal BEAM compiler, with respect to semantic behavior. There are however features in the BEAM compiler and the runtime system that have limited or no support for HiPE compiled modules.

Stack traces

Stack traces returned from *`erlang:get_stacktrace/0`* or as part of `'EXIT'` terms can look incomplete if HiPE compiled functions are involved. Typically a stack trace will contain only BEAM compiled functions or only HiPE compiled functions, depending on where the exception was raised.

Source code line numbers in stack traces are also not supported by HiPE compiled functions.

Tracing

Erlang call trace is not supported by HiPE. Calling *`erlang:trace_pattern({M,F,A}, ...)`* does not have any effect on HiPE compiled modules.

NIFs

Modules compiled with HiPE can not call *`erlang:load_nif/2`* to load NIFs.

-on_load

Modules compiled with HiPE can not use *`-on_load()`* directives.

Performance Limitations

The HiPE compiler does in general produce faster code than the BEAM compiler. There are however some situation when HiPE compiled code will perform worse than BEAM code.

Mode switches

Every time a process changes from executing code in a HiPE compiled module to a BEAM compiled module (or vice versa), it will do a mode switch. This involves a certain amount of CPU overhead which can have a negative net impact if the process is switching back and forth without getting enough done in each mode.

Optimization for `receive` with unique references

The BEAM compiler can do an optimization when a receive statement is only waiting for messages containing a reference created before the receive. All messages that existed in the queue when the reference was created will

be bypassed, as they cannot possibly contain the reference. HiPE currently has an optimization similar this, but it is not guaranteed to bypass all messages. In the worst case scenario it, cannot bypass any messages at all.

An example of this is when `gen_server:call()` waits for the reply message.

Stability Issues

Not yielding in `receive` statements

HiPE will not yield in `receive` statements where appropriate. If a process have lots of signals in its signal queue and execute a HiPE compiled `receive` statement, the scheduler thread performing the execution may be stuck in the `receive` statement for a very long time. This can in turn cause various severe issues such as for example prevent the runtime system from being able to release memory.

SEE ALSO

c(3), *compile(3)*