

HiPE

Copyright © 2006-2023 Ericsson AB. All Rights Reserved.
HiPE 3.19.2
May 9, 2023

Copyright © 2006-2023 Ericsson AB. All Rights Reserved.
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License. Ericsson AB. All Rights Reserved..

May 9, 2023

1 Reference Manual

HiPE

Application

> **Note:**
>
> HiPE and execution of HiPE compiled code only have limited support by the OTP team at Ericsson. The OTP team only does limited maintenance of HiPE and does not actively develop HiPE. HiPE is mainly supported by the HiPE team at Uppsala University.

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.

Binary matching

The HiPE compiler will crash on modules containing binary matching.

try/catch

The HiPE compiler will crash on modules containing 'try' or 'catch'.

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 cannot call *`erlang:load_nif/2`* to load NIFs.

-on_load

Modules compiled with HiPE cannot 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.

Garbage collection after BIFs

> The condition for determining whether a garbage collection is needed or not has changed in later releases. HiPE has not been updated regarding this which may cause premature garbage collections after BIF calls.

Stability Issues

Not checking reduction count on function returns

> BEAM checks the reduction count and schedules out the executing process if needed both when calling a function and when returning from a function call that was not called using a tail call. HiPE only checks the reduction count when calling a function.
>
> The runtime system might need to schedule out a process in order to reclaim memory. If the process isn't scheduled out soon after the process has entered this state, memory consumption will quickly grow. Maintaining this state is also quite expensive performance wise.
>
> Processes executing code that performs large recursions and produce data after returning from recursive calls may have to be scheduled out when returning from a function call. Since HiPE does not check reductions on returns, processes executing such HiPE compiled code may cause huge peeks in memory consumption as well as severe performance degradation.

Not bumping appropriate amount of reductions in `receive` statements

> The process signaling improvements made in ERTS version 10.0 moved potentially significant amounts of work into the receive statement from other places. In order to account for this work, the reduction count should be bumped on the executing process. Reductions are not bumped when entering the `receive` statement from HiPE compiled code.

SEE ALSO

c(3), compile(3)