Intel® Software Development Products

Profiling Runtime Generated and Interpreted Code with Intel® VTune™ Amplifier

Intel® VTune™ Amplifier 2013 for Windows* OS

User's Guide

Document Number: 329381-001

Legal Information

Legal Information

JIT Profiling API

About JIT Profiling API

The JIT (Just-In-Time) Profiling API provides functionality to report information about just-in-time generated code that can be used by performance tools. You need to insert JIT Profiling API calls in the code generator to report information before JIT-compiled code goes to execution. This information is collected at runtime and used by tools like Intel® VTune™ Amplifier to display performance metrics associated with JIT-compiled code.

You can use the JIT Profiling API to profile such environments as dynamic JIT compilation of JavaScript code traces, JIT execution in OpenCL™ applications, Java*/.NET* managed execution environments, and custom ISV JIT engines.

The standard VTune Amplifier installation contains a static part (as a static library and source files) and a profiler object. The JIT engine generating code during runtime communicates with a profiler object through the static part. During runtime, the JIT engine reports the information about JIT-compiled code stored in a trace file by the profiler object. After collection, the VTune Amplifier uses the generated trace file to resolve the JIT-compiled code. If the VTune Amplifier is not installed, profiling is disabled.

Use the JIT Profiling API to:

- Profile trace-based and method-based JIT-compiled code
- Analyze split functions
- Explore inline functions

Profiling Trace-based and Method-based JIT-compiled Code

This is the most common scenario for using JIT Profiling API to profile trace-based and method-based JIT-compiled code:

```
#include <jitprofiling.h>

if (iJIT_IsProfilingActive != iJIT_SAMPLING_ON) {
    return;
}

iJIT_Method_Load jmethod = {0};
jmethod.method_id = iJIT_GetNewMethodID();
jmethod.method_name = "method_name";
jmethod.class_file_name = "class_name";
jmethod.source_file_name = "source_file_name";
jmethod.method_load_address = code_addr;
jmethod.method_size = code_size;

iJIT_NotifyEvent(iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED,
      (void*)&jmethod);
iJIT_NotifyEvent(iJVM_EVENT_TYPE_SHUTDOWN, NULL);
```

Usage Tips

- If any `iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED` event overwrites an already reported method, then such a method becomes invalid and its memory region is treated as unloaded. VTune Amplifier displays the metrics collected by the method until it is overwritten.
- If supplied line number information contains multiple source lines for the same assembly instruction (code location), then VTune Amplifier picks up the first line number.
- Dynamically generated code can be associated with a module name. Use the `iJIT_Method_Load_V2` structure.

    - If you register a function with the same method ID multiple times, specifying different module names, then the VTune Amplifier picks up the module name registered first. If you want to distinguish the same function between different JIT engines, supply different method IDs for each function. Other symbolic information (for example, source file) can be identical.

Analyzing Split Functions

You can use the JIT Profiling API to analyze split functions (multiple joint or disjoint code regions belonging to the same function) including re-JITting with potential overlapping of code regions in time, which is common in resource-limited environments.

```
#include <jitprofiling.h>

unsigned int method_id = iJIT_GetNewMethodID();

iJIT_Method_Load a = {0};
a.method_id = method_id;
a.method_load_address = 0x100;
a.method_size = 0x20;

iJIT_Method_Load b = {0};
b.method_id = method_id;
b.method_load_address = 0x200;
b.method_size = 0x30;

iJIT_NotifyEvent(iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED, (void*)&a);
iJIT_NotifyEvent(iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED, (void*)&b)
```

Usage Tips

- If a `iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED` event overwrites an already reported method, then such a method becomes invalid and its memory region is treated as unloaded.
- All code regions reported with the same method ID are considered as belonging to the same method. Symbolic information (method name, source file name) will be taken from the first notification, and all subsequent notifications with the same method ID will be processed only for line number table information. So, the VTune Amplifier will map samples to a source line using the line number table from the current notification while taking the source file name from the very first one.
    - If you register a second code region with a different source file name and the same method ID, this information will be saved and will not be considered as an extension of the first code region, but VTune Amplifier will use the source file of the first code region and map performance metrics incorrectly.
    - If you register a second code region with the same source file as for the first region and the same method ID, the source file will be discarded but VTune Amplifier will map metrics to the source file correctly.
    - If you register a second code region with a null source file and the same method ID, provided line number info will be associated with the source file of the first code region.

Exploring Inline Functions

You can use the JIT Profiling API to explore inline functions including multi-level hierarchy of nested inline methods that shows how performance metrics are distributed through them.

```
#include <jitprofiling.h>

  //                                    method_id   parent_id
  //   [-- c --]                          3000        2000
  //                  [---- d -----]      2001        1000
  //  [---- b ----]                       2000        1000
  // [------------ a ----------------]    1000         n/a

iJIT_Method_Load a = {0};
a.method_id = 1000;

iJIT_Method_Inline_Load b = {0};
b.method_id = 2000;
```

```
b.parent_method_id = 1000;

iJIT_Method_Inline_Load c = {0};
c.method_id = 3000;
c.parent_method_id = 2000;

iJIT_Method_Inline_Load d = {0};
d.method_id = 2001;
d.parent_method_id = 1000;

iJIT_NotifyEvent(iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED, (void*)&a);
iJIT_NotifyEvent(iJVM_EVENT_TYPE_METHOD_INLINE_LOAD_FINISHED, (void*)&b);
iJIT_NotifyEvent(iJVM_EVENT_TYPE_METHOD_INLINE_LOAD_FINISHED, (void*)&c);
iJIT_NotifyEvent(iJVM_EVENT_TYPE_METHOD_INLINE_LOAD_FINISHED, (void*)&d);
```

Usage Tips

- Each inline (`iJIT_Method_Inline_Load`) method should be associated with two method IDs: one for itself; one for its immediate parent.
- Address regions of inline methods of the same parent method cannot overlap each other.
- Execution of the parent method must not be started until it and all its inline methods are reported.
- In case of nested inline methods an order of `iJVM_EVENT_TYPE_METHOD_INLINE_LOAD_FINISHED` events is not important.
- If any event overwrites either inline method or top parent method, then the parent, including inline methods, becomes invalid and its memory region is treated as unloaded.

See Also

JIT Profiling API Reference
Using JIT Profiling API

Using JIT Profiling API

To include JIT Profiling support, do one of the following:

- Include the following files to your source tree:
  - `jitprofiling.h`, located under `<install-dir>\include`
  - `ittnotify_config.h`, `ittnotify_types.h` and `jitprofiling.c`, located under `<install-dir>\sdk\src\ittnotify`

  NOTE
  The default installation directory is `C:\[Program Files]\Intel\VTune Amplifier XE 2013`

- Use the static library provided with the product:

  1. Include `jitprofiling.h` file, located under the `<install-dir>\include` directory, in your code. This header file provides all API function prototypes and type definitions.
  2. Link to `jitprofiling.lib`, located under `<install-dir>\lib32`or `<install-dir>\lib64`.

Use This Primitive	To Do This
`int iJIT_NotifyEvent( iJIT_JVM_EVENT event_type, void *EventSpecificData );`	Use this API to send a notification of `event_type` with the data pointed by `EventSpecificData` to the agent. The reported information is used to attribute samples obtained from any Intel® VTune™ Amplifier collector.

Use This Primitive	To Do This
`unsigned int iJIT_GetNewMethodID ( void );`	Generate a new method ID. You must use this function to assign unique and valid method IDs to methods reported to the profiler. This API returns a new unique method ID. When out of unique method IDs, this API function returns 0.
`iJIT_IsProfilingActiveFlags iJIT_IsProfilingActive( void );`	Returns the current mode of the profiler: off, or sampling, using the `iJIT_IsProfilingActiveFlags` enumeration. This API returns `iJIT_SAMPLING_ON` by default, indicating that Sampling is running. It returns `iJIT_NOTHING_RUNNING` if no profiler is running.

Lifetime of Allocated Data

You send an event notification to the agent (VTune Amplifier) with event-specific data, which is a structure. The pointers in the structure refer to memory you allocated and you are responsible for releasing it. The pointers are used by the `iJIT_NotifyEvent` method to copy your data in a trace file, and they are not used after the `iJIT_NotifyEvent` method returns.

JIT Profiling API Sample Application

VTune Amplifier is installed with a sample application in the `jitprofiling_vtune_amp_xe.zip` that emulates the creation and execution of dynamic code. In addition, it uses the JIT profiling API to notify the VTune Amplifier when it transfers execution control to dynamic code.

To install and set up the sample code:

1. Copy the `jitprofiling_vtune_amp_xe.zip` file from the `<install-dir>\samples\<locale>\C++` directory to a writable directory or share on your system.
2. Extract the sample from the .zip file.

See Also

About JIT Profiling API
JIT Profiling API Reference

JIT Profiling API Reference

iJIT_NotifyEvent

Reports information about JIT-compiled code to the agent.

Syntax

```
int iJIT_NotifyEvent( iJIT_JVM_EVENT event_type, void *EventSpecificData );
```

Description

The `iJIT_NotifyEvent` function sends a notification of `event_type` with the data pointed by `EventSpecificData` to the agent. The reported information is used to attribute samples obtained from any Intel® VTune™ Amplifier collector. This API needs to be called after JIT compilation and before the first entry into the JIT-compiled code.

Input Parameters

Parameter	Description
`iJIT_JVM_EVENT event_type`	Notification code sent to the agent. See a complete list of event types below.
`void` `*EventSpecificData`	Pointer to event specific data.

The following values are allowed for `event_type`:

`iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED`	Send this notification after a JITted method has been loaded into memory, and possibly JIT compiled, but before the code is executed. Use the `iJIT_Method_Load` structure for `EventSpecificData`. The return value of `iJIT_NotifyEvent` is undefined.
`iJVM_EVENT_TYPE_SHUTDOWN`	Send this notification to terminate profiling. Use NULL for `EventSpecificData`. `iJIT_NotifyEvent` returns 1 on success.
`JVM_EVENT_TYPE_METHOD_UPDATE`	Send this notification to provide new content for a previously reported dynamic code. The previous content will be invalidated starting from the time of the notification. Use the `iJIT_Method_Load` structure for `EventSpecificData` with the following required fields: • `method_id` to identify the code to update • `method_load_address` to specify the start address within an identified code range where the update should be started • `method_size` to specify the length of an updated code range
`JVM_EVENT_TYPE_METHOD_INLINE_LOAD_FINISHED`	Send this notification when an inline dynamic code is JIT compiled and loaded into memory by the JIT engine, but before the parent code region starts executing. Use the `iJIT_Method_Inline_Load` structure for `EventSpecificData`.
`iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED_V2`	Send this notification when a dynamic code is JIT compiled and loaded into memory by the JIT engine, but before the code is executed. Use the `iJIT_Method_Load_V2` structure for `EventSpecificData`.

The following structures can be used for `EventSpecificData`:

iJIT_Method_Inline_Load Structure

When you use the `iJIT_Method_Inline_Load` structure to describe the JIT compiled method, use `iJVM_EVENT_TYPE_METHOD_INLINE_LOAD_FINISHED` as an event type to report it. The `iJIT_Method_Inline_Load` structure has the following fields:

Field	Description
unsigned int method_id	Unique method ID. Method ID cannot be smaller than 999. You must either use the API function iJIT_GetNewMethodID to get a valid and unique method ID, or else manage ID uniqueness and correct range by yourself.
unsigned int parent_method_id	Unique immediate parent's method ID. Method ID may not be smaller than 999. You must either use the API function iJIT_GetNewMethodID to get a valid and unique method ID, or else manage ID uniqueness and correct range by yourself.
char *method_name	The name of the method, optionally prefixed with its class name and appended with its complete signature. This argument cannot be set to NULL.
void *method_load_address	The base address of the method code. Can be NULL if the method is not JITted.
unsigned int method_size	The virtual address on which the method is inlined. If NULL, then data provided with the event are not accepted.
unsigned int line_number_size	The number of entries in the line number table. 0 if none.
pLineNumberInfo line_number_table	Pointer to the line numbers info array. Can be NULL if line_number_size is 0. See LineNumberInfo structure for a description of a single entry in the line number info array.
char *class_file_name	Class name. Can be NULL.
char *source_file_name	Source file name. Can be NULL.

iJIT_Method_Load Structure

When you use the iJIT_Method_Load structure to describe the JIT compiled method, use iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED as an event type to report it. The iJIT_Method_Load structure has the following fields:

Field	Description
unsigned int method_id	Unique method ID. Method ID cannot be smaller than 999. You must either use the API function iJIT_GetNewMethodID to get a valid and unique method ID, or else manage ID uniqueness and correct range by yourself.
char *method_name	The name of the method, optionally prefixed with its class name and appended with its complete signature. This argument cannot be set to NULL.
void *method_load_address	The base address of the method code. Can be NULL if the method is not JITted.
unsigned int method_size	The virtual address on which the method is inlined. If NULL, then data provided with the event are not accepted.
unsigned int line_number_size	The number of entries in the line number table. 0 if none.

Field	Description
pLineNumberInfo line_number_table	Pointer to the line numbers info array. Can be NULL if line_number_size is 0. See LineNumberInfo structure for a description of a single entry in the line number info array.
unsigned int class_id	This field is obsolete.
char *class_file_name	Class name. Can be NULL.
char *source_file_name	Source file name. Can be NULL.
void *user_data	This field is obsolete.
unsigned int user_data_size	This field is obsolete.
iJDEnvironmentType env	This field is obsolete.

iJIT_Method_Load_V2 Structure

When you use the iJIT_Method_Load_V2 structure to describe the JIT compiled method, use iJVM_EVENT_TYPE_METHOD_LOAD_FINISHED_V2 as an event type to report it. The iJIT_Method_Load_V2 structure has the following fields:

Field	Description
unsigned int method_id	Unique method ID. Method ID cannot be smaller than 999. You must either use the API function iJIT_GetNewMethodID to get a valid and unique method ID, or else manage ID uniqueness and correct range by yourself.
char *method_name	The name of the method, optionally prefixed with its class name and appended with its complete signature. This argument cannot be set to NULL.
void *method_load_address	The base address of the method code. Can be NULL if the method is not JITted.
unsigned int method_size	The virtual address on which the method is inlined. If NULL, then data provided with the event are not accepted.
unsigned int line_number_size	The number of entries in the line number table. 0 if none.
pLineNumberInfo line_number_table	Pointer to the line numbers info array. Can be NULL if line_number_size is 0. See LineNumberInfo structure for a description of a single entry in the line number info array.
char *class_file_name	Class name. Can be NULL.
char *source_file_name	Source file name. Can be NULL.
char *module_name	Module name. Can be NULL. The module name can be useful for distinguishing among different JIT engines. VTune Amplifier will display reported methods grouped by specific module.

LineNumberInfo Structure

Use the `LineNumberInfo` structure to describe a single entry in the line number information of a code region. A table of line number entries provides information about how the reported code region is mapped to source file. VTune Amplifier uses line number information to attribute the samples (virtual address) to a line number. It is acceptable to report different code addresses for the same source line:

Offset	Line Number
1	2
12	4
15	2
18	1
21	30

VTune Amplifier constructs the following table using the client data:

Code sub-range	Line number
0-1	2
1-12	4
12-15	2
15-18	1
18-21	30

The `LineNumberInfo` structure has the following fields:

Field	Description
`unsigned int Offset`	Opcode byte offset from the beginning of the method.
`unsigned int LineNumber`	Matching source line number offset (from beginning of source file).

Return Values

The return values are dependent on the particular `iJIT_JVM_EVENT`.

See Also

About JIT Profiling API
Using JIT Profiling API

iJIT_IsProfilingActive

Returns the current mode of the agent.

Syntax

```
iJIT_IsProfilingActiveFlags JITAPI iJIT IsProfilingActive ( void )
```

Description

The `iJIT_IsProfilingActive` function returns the current mode of the agent.

Input Parameters

None

Return Values

`iJIT_SAMPLING_ON`, indicating that agent is running, or `iJIT_NOTHING_RUNNING` if no agent is running.

See Also

About JIT Profiling API
Using JIT Profiling API

iJIT_ GetNewMethodID

Generates a new unique method ID.

Syntax

```
unsigned int iJIT_GetNewMethodID(void);
```

Description

The `iJIT_GetNewMethodID` function generates new method ID upon each call. Use this API to obtain unique and valid method IDs for methods or traces reported to the agent if you do not have your own mechanism to generate unique method IDs.

Input Parameters

None

Return Values

A new unique method ID. When out of unique method IDs, this API function returns 0.

See Also

About JIT Profiling API
Using JIT Profiling API