

HTC Programmers Guide

February 5, 2015

Version 0.1

901-000013-000

This work is licensed under the Creative Commons AttributionShareAlike 4.0 International License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

This document is adapted in part from OpenMP 4.0 Specification and is covered by the following: Copyright 1997-208 OpenMP Architecture Review Board

**Trademarks**

The following are trademarks of Convey Computer Corporation:

The Convey Computer Logo: 

Convey Computer

HC-1

HC-1ex

HC-2

HC-2ex

Trademarks of other companies

Intel is a registered trademark of Intel Corporation

Adobe and Adobe Reader are registered trademarks of Adobe Systems Incorporated

Linux is a registered trademark of Linus Torvalds

Revisions

| Version | Description |
| --- | --- |
| 0.1 | Preliminary |

Table of Contents

[1 Overview 5](#_Toc413142520)

[1.1 Introduction 5](#_Toc413142521)

[1.2 Document Content 5](#_Toc413142522)

[1.3 Related Documents 5](#_Toc413142523)

[1.3.1 Intended Audience 5](#_Toc413142524)

[2 HT OpenMP 6](#_Toc413142525)

[2.1 Directives 6](#_Toc413142526)

[parallel 6](#_Toc413142527)

[loop 7](#_Toc413142528)

[sections 7](#_Toc413142529)

[single 8](#_Toc413142530)

[target 8](#_Toc413142531)

[declare target 8](#_Toc413142532)

[teams 8](#_Toc413142533)

[distribute 9](#_Toc413142534)

[distribute parallel for 9](#_Toc413142535)

[parallel loop 9](#_Toc413142536)

[parallel sections 10](#_Toc413142537)

[target teams 10](#_Toc413142538)

[teams distribute 11](#_Toc413142539)

[target teams distribute 11](#_Toc413142540)

[teams distribute parallel 11](#_Toc413142541)

[target teams distribute parallel for 12](#_Toc413142542)

[master 12](#_Toc413142543)

[critical 12](#_Toc413142544)

[barrier 12](#_Toc413142545)

[atomic 12](#_Toc413142546)

[2.2 Runtime Library Routines 13](#_Toc413142547)

[2.2.1 Execution Environment Routines 13](#_Toc413142548)

[omp\_get\_num\_threads 13](#_Toc413142549)

[omp\_get\_max\_threads 13](#_Toc413142550)

[omp\_get\_thread\_num 13](#_Toc413142551)

[omp\_get\_num\_devices 13](#_Toc413142552)

[omp\_get\_num\_teams 14](#_Toc413142553)

[omp\_get\_team\_num 14](#_Toc413142554)

[omp\_is\_initial\_device 14](#_Toc413142555)

[2.2.2 Lock Routines 14](#_Toc413142556)

[Initialize Lock 14](#_Toc413142557)

[Destroy Lock 14](#_Toc413142558)

[Set Lock 14](#_Toc413142559)

[Unset Lock 14](#_Toc413142560)

[Test Lock 14](#_Toc413142561)

[2.3 Clauses 15](#_Toc413142562)

[2.3.1 Data Sharing Attribute Clauses 15](#_Toc413142563)

[default[shared|none] 15](#_Toc413142564)

[shared[*list*] 15](#_Toc413142565)

[private[*list*] 15](#_Toc413142566)

[firstprivate[*list*] 15](#_Toc413142567)

[lastprivate[*list*] 15](#_Toc413142568)

[reduction[*reduction-identifier:list*] 15](#_Toc413142569)

[2.4 C Language Limitations 16](#_Toc413142570)

[2.5 HTC Extensions 16](#_Toc413142571)

[2.5.1 Intrinsics 16](#_Toc413142572)

[2.5.2 HTC Specific Pragmas 17](#_Toc413142573)

[2.6 Coding and Optimization Strategies 18](#_Toc413142574)

[2.6.1 Thread Synchronization 18](#_Toc413142575)

[2.6.2 Static Schedules 18](#_Toc413142576)

[2.6.3 Designating HTC Implementation Area 18](#_Toc413142577)

# 

# Overview

## Introduction

The OpenHT OpenMP Application Program Interface (API) gives parallel programmers a simple and flexible interface for developing portable parallel applications in C/C++, which can be compiled for an FPGA based coprocessor.

The OpenMP target construct(s) will be targeted for an FPGA coprocessor. The construct will be translated to HT instructions which are then compiled into Verilog. The Verilog is compiled using FPGA vendor specific tool for the target FPGA.

## Document Content

This document describes the subset of the OpenMP Application Program Interface implemented in the OpenHT compiler (HTC).

## Related Documents

The OpenMP Application Program Interface, Version 4.0 contains a complete description of compiler directives, library routines and environment variables defined in the OpenMP API.

The HT Reference Guide contains an overview of HT designs. It is recommended that users have an understanding of the HT Architecture before using the HTC Programmers Guide.

The HT Programmers Guide describes application development of both the host and coprocessor portions of an HTdesign.

### Intended Audience

This document is intended for users interested in developing parallel application to run on an FPGA based coprocessor. While it does raise the level of abstraction involved in programming the personality, it is recommended that the user have some experience with FPGA development. The user should also be capable of writing and debugging applications in C/C++.

# HT OpenMP

The HT OpenMP C/C++ compiler supports a subset of the OpenMP 4.0 API. The supported directives, runtime library routines and clauses are described in this document. Environment variables described in the OpenMP 4.0 API are not supported.

## Directives

An HT OpenMP executable directive applies to the succeeding structured block or and OpenMP construct. Each directive starts with #pragma omp. The remainder of the direct follows the conventions of the C and C++ standards for compiler directives.

The OpenMP target construct(s) will be targeted for an FPGA coprocessor. The construct(s) will be translated to HT instructions which are then compiled into Verilog. The Verilog is compiled using FPGA vendor specific tool for the coprocessor FPGA. Execution will occur on the host processor when the coprocessor is not available or is not supported.

parallel

Forms a team of threads and starts parallel execution.

#pragma omp parallel *[clause [, clause]…]*

*structured-block*

where *clause* is one of the following*:*

**if***(scalar -expression)*

**num\_threads***(integer-expression)*

**default(shared | none)**

**private***(list)*

**firstprivate***(list)*

**shared***(list)*

**reduction***(reduction-identifier: list)*

loop

Specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.

#pragma omp for *[clause [, clause]…]*

*for-loops*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**lastprivate***(list)*

**reduction***(reduction-identifier: list)*

**schedule***(***static***[, chunk\_size])*

**nowait**

sections

A non-iterative work sharing construct that contains a set of structured blocks that are to be distributed among and executed by threads in a team.

#pragma omp section *[clause [, clause]…]*

{

*[#pragma omp section]*

*structured block*

*[#pragma omp section*

*structured block]*

}

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**lastprivate***(list)*

**reduction***(reduction-identifier: list)*

**nowait**

single

Specifies that the associated structured block is executed by only one of the threads in the team.

#pragma omp single *[clause [, clause]…]*

*structured block*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**nowait**

target

Creates a device data environment and executes the construct on the same device.

#pragma omp target

*structured block*

declare target

A declarative directive that specifies that variables and functions are mapped to the FPGA.

#pragma omp declare target

*declarations-definition sequence*

#pragma omp end declare target

teams

Creates a league of thread teams where the master thread of each team executes the region. Each team is an HT Unit.

#pragma omp teams *[clause [, clause]…]*

*structured-block*

where *clause* is one of the following*:*

**num\_teams***(integer-expression)*

**thread\_limit***(integer-expression)*

**default(shared | none)**

**private***(list)*

**firstprivate***(list)*

**shared***(list)*

distribute

Specifies loops which are executed by the thread teams.

#pragma omp distribute *[clause [, clause]…]*

*for loops*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**dist\_schedule***(***static***[, chunk\_size])*

distribute parallel for

Specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams.

#pragma omp distribute parallel for*[clause [, clause]…]*

*for loops*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**dist\_schedule***(***static***[, chunk\_size])*

parallel loop

Shortcut for specifying a **parallel** construct containing one or more associated loops and no other statements

#pragma omp parallel for*[clause [, clause]…]*

*for loops*

where *clause* is one of the following*:*

**if***(scalar -expression)*

**num\_threads***(integer-expression)*

**default(shared | none)**

**private***(list)*

**firstprivate***(list)*

**lastprivate***(list)*

**reduction***(reduction-identifier: list)*

**schedule***(***static***[, chunk\_size])*

parallel sections

Shortcut for specifying a **parallel** construct containing one sections construct and no other statements.

#pragma omp parallel section*[clause [, clause]…]*

{

*[#pragma omp section]*

*structured-block*

*[#pragma omp section*

*structured-block]*

}

where *clause* is one of the following*:*

**if***(scalar -expression)*

**num\_threads***(integer-expression)*

**default(shared | none)**

**private***(list)*

**firstprivate***(list)*

**lastprivate***(list)*

**shared***(list)*

**reduction***(reduction-identifier: list)*

target teams

Shortcut for specifying a **target** construct containing a **teams** construct.

#pragma omp target teams *[clause [, clause]…]*

*structured-block*

where *clause* is one of the following*:*

**num\_teams***(integer-expression)*

**thread\_limit***(integer-expression)*

**default(shared | none)**

**private***(list)*

**firstprivate***(list)*

**shared***(list)*

teams distribute

Shortcut for specifying a **teams** construct containing a **distribute** construct.

#pragma omp teams distribute *[clause [, clause]…]*

*for loops*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**dist\_schedule***(***static***[, chunk\_size])*

**num\_teams***(integer-expression)*

**thread\_limit***(integer-expression)*

**default(shared | none)**

**shared***(list)*

target teams distribute

Shortcut for specifying a **target** containing a **teams distribute** construct.

#pragma omp target teams distribute *[clause [, clause]…]*

*for loops*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**dist\_schedule***(***static***[, chunk\_size])*

**num\_teams***(integer-expression)*

**thread\_limit***(integer-expression)*

**default(shared | none)**

**shared***(list)*

teams distribute parallel

Shortcut for specifying a **teams** construct containing a **distribute parallel** construct.

#pragma omp teams distribute parallel *[clause [, clause]…]*

*for loops*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**dist\_schedule***(***static***[, chunk\_size])*

**num\_teams***(integer-expression)*

**thread\_limit***(integer-expression)*

**default(shared | none)**

**shared***(list)*

target teams distribute parallel for

Shortcut for specifying a **target** construct containing a **teams distribute parallel for** construct.

#pragma omp target teams distribute parallel for *[clause [, clause]…]*

*for loops*

where *clause* is one of the following*:*

**private***(list)*

**firstprivate***(list)*

**dist\_schedule***(***static***[, chunk\_size])*

**num\_teams***(integer-expression)*

**thread\_limit***(integer-expression)*

**default(shared | none)**

**shared***(list)*

master

Specifies a structured block that is executed by the master thread of the team.

#pragma omp master

*structured block*

critical

Restricts execution of the associated structured block to a single thread at a time.

#pragma omp critical *[(name)]*

*structured block*

barrier

Specifies an explicit barrier at the point at which the construct appears..

#pragma omp barrier

atomic

Ensures that a specific storage location is accesses atomically.

#pragma omp atomic

*expression-stmt*

where *expression-stmt* may be one of the following:

*x++;*

*x--;*

*++x*

*--x;*

*x binop= expr;*

*x=expr biniop x;*

#### Defaults

Default loop schedule is ‘schedule(static)’, so if no schedule clause is specified, a static schedule with unspecified chunk size is assumed. Similarly, the default distribute schedule is ‘dist\_schedule(static)’.

Nested parallelism is enabled by default.

## Runtime Library Routines

The HT OpenMP runtime library contains execution environment routines, which affect and monitor threads, processors and the parallel environment. The library routines are external functions with “C” linkage.

### Execution Environment Routines

omp\_get\_num\_threads

Returns the number of threads in the current team. The binding region for an **omp\_get\_num\_threads** region is the innermost enclosing **parallel** region.

int omp\_get\_num\_threads(void);

omp\_get\_max\_threads

Returns the upper bound on the number of threads that could be used to form a new team if a **parallel** construct without a **num\_threads** clause were encountered after execution returns from this routine.

int omp\_get\_max\_threads(void);

omp\_get\_thread\_num

Returns the thread number of the calling thread within the current team.

int omp\_get\_thread\_num(void);

omp\_get\_num\_devices

Returns the number of the target devices. ???

int omp\_get\_num\_teams(void);

omp\_get\_num\_teams

Returns the number of teams in the current **teams** region or 1 if called from outside of a **teams** region.

int omp\_get\_num\_teams(void);

omp\_get\_team\_num

Returns the team number of the calling thread. The team number is an integer between 0 and one less than the value returned by **omp\_get\_num\_teams** inclusive.

int omp\_get\_team\_num(void);

omp\_is\_initial\_device

Returns *true* if the current task is executing on the host device; otherwise it returns *false*.

int omp\_is\_initial\_device(void);

### Lock Routines

General purpose lock routines. Simple locks are supported. Simple locks cannot be set if it is already owned by the task trying to set it.

Initialize Lock

Initialize an HT OpenMP lock.

void omp\_init\_lock(omp\_lock\_t\**lock*);

Destroy Lock

Ensure that the HT OpenMP lock is uninitialized.

void omp\_destroy\_lock(omp\_lock\_t\**lock*);

Set Lock

Sets an HT OpenMP lock. The calling task region is suspended until the lock is set.

void omp\_set\_lock(omp\_lock\_t\**lock*);

Unset Lock

Unsets an HT OpenMP lock.

void omp\_set\_lock(omp\_lock\_t\**lock*);

Test Lock

Attempt to set an HT OpenMP lock but do not suspend execution of the task executing the routine.

void omp\_set\_lock(omp\_lock\_t\**lock*);

## Clauses

The set of clauses that is valid on a particular directive is described with the directive. Most clauses accept a comma-separated list of list items. All list items appearing in a clause must be visible, according to the scoping rules of the base language. Not all of the clauses listed in this section are valid on all directives. The set of clauses that is valid on a particular directive is described with the directive in Section 2.1.

### Data Sharing Attribute Clauses

Data sharing attribute clauses apply only to variables whose names are visible in the construct on which the clause appears.

default[shared|none]

Explicitly determines the default data-sharing attributes of variables that are reference in a **parallel** or **teams** construct, causing all variables referenced in the construct that have implicitly determined data-sharing attributes to be shared.

shared[*list*]

Declares one or more list items to be shared by tasks generated by a **parallel** or **teams** construct.

private[*list*]

Declares one or more list items to be private to a task. Each task that references a list item that appears in a private clause in any statement in the construct receives a new list item.

firstprivate[*list*]

Declares list items to be private to a task, and initializes each of them with the value that the corresponding original item has when the construct is encountered.

lastprivate[*list*]

Declares one or more list items to be private to an implicit task and causes the corresponding original list item to be updated after the end of the region.

reduction[*reduction-identifier:list*]

Specifies a *reduction-identifier* and one or more list items. The *reduction-identifier* must match a previously declared *reduction identifier* of the same name and type for each of the list items.

|  |  |
| --- | --- |
| **Operators for reduction (initialization values)** | |
| + (0) | | (0) |
| \* (1) | ^ (0) |
| . (0) | && (0) |
| & (~0) | || (0) |
| max (Least representable number in **reduction** list item type) | |
| min (Largest representable number in **reduction** list item type) | |

## C Language Limitations

The HTC compiler has several C-language limitations, which are listed below:

* Recursive functions and mutually recursive functions are not supported
* Floating point datatypes and computations are not supported
* Standard library support is not available when creating the FPGA
* Bitfields are not supported
* Calls through function pointers are not supported
* GNU “statement expressions” are not allowed
* Array and struct initializers are not supported
* The relocation of struct declarations from inner scopes to the outermost function scope is not allowed (declaration lifting). Structs should be declared at function scope.
* C++ features not present in C are not supported.

## HTC Extensions

### Intrinsics

HTC provides two “stencil” intrinsics which direct the generation of an optimized, streaming hardware implementation of a stencil function according to the given parameters.

A general version is available in the *rhomp\_stencil\_conv2d* intrinsic which specifies a non-square, non-centered stencil. The orgx and orgy parameters specify the stencil origin or "center.”

The *rhomp\_stencil\_conv2ds* intrinsic is specifies a square, centered, 2D convolution stencil (stencil dim is always odd, so there is a true center).

void rhomp\_stencil\_conv2d (void \*grid\_dst, void \*grid\_src,

int grid\_dimx, int grid\_dimy,

int stencil\_dimx, int stencil\_dimy,

int stencil\_orgx, int stencil\_orgy,

void \*kernel, int pipelen);

void rhomp\_stencil\_conv2ds (void \*grid\_dst, void \*grid\_src,

int grid\_dimx, int grid\_dimy,

int stencil\_dim, void \*kernel,

int pipelen);

where

grid\_dst: base address of the destination grid

grid\_src: base address of the source grid

grid\_dimx: number of grid columns (includes border elements if any)

grid\_dimy: number of grid rows (includes border elements if any)

stencil\_dimx: number of stencil columns

stencil\_dimy: number of stencil rows

stencil\_orgx: stencil x origin (0-based)

stencil\_orgy: stencil y origin (0-based)

kernel: base address of coefficient matrix

pipelen < 4096:Generate optimized coproc streaming stencil code

pipelen = 4096: Generate generic coproc stencil loop nest cod.

pipelen = 8192: Generate generic host stencil loop nest code

### HTC Specific Pragmas

The maximum number of threads supported by the FPGA hardware module is specified using a HTC specific pragma.

#pragma rhomp max\_phys\_threads (N)

where

N is a compile-time integer constant that indicates the number of threads (log2) supported by the FPGA.

If the pragma appears anywhere in the global scope, the default width of every module in the Unit is set to N (log2) threads. If the pragma occurs within a routine, the module width is only set for that routine. The routine level pragma overrides the default for that routine, so the user can specify one default for many routines and only override when necessary.

If the user does not use the pragma, the default thread width is 32 threads. Stencil modules and host entry modules are always single threaded.

## Coding and Optimization Strategies

### Thread Synchronization

HTC automatically synchronizes threads at the end of a parallel region, so a barrier is not needed after the last loop in a parallel region.

### Static Schedules

Only static schedules (for both ‘for’ and ‘distribute’) are supported. and explicit chunk size should be used to avoid a division operation on the FPGA, since a division operation in the FPGA may not make timing.

### Designating HTC Implementation Area

HTC predefines the macro

\_CNY\_HTC\_SOURCE

This macro is used to indicate code sections specific to the HTC implementation.