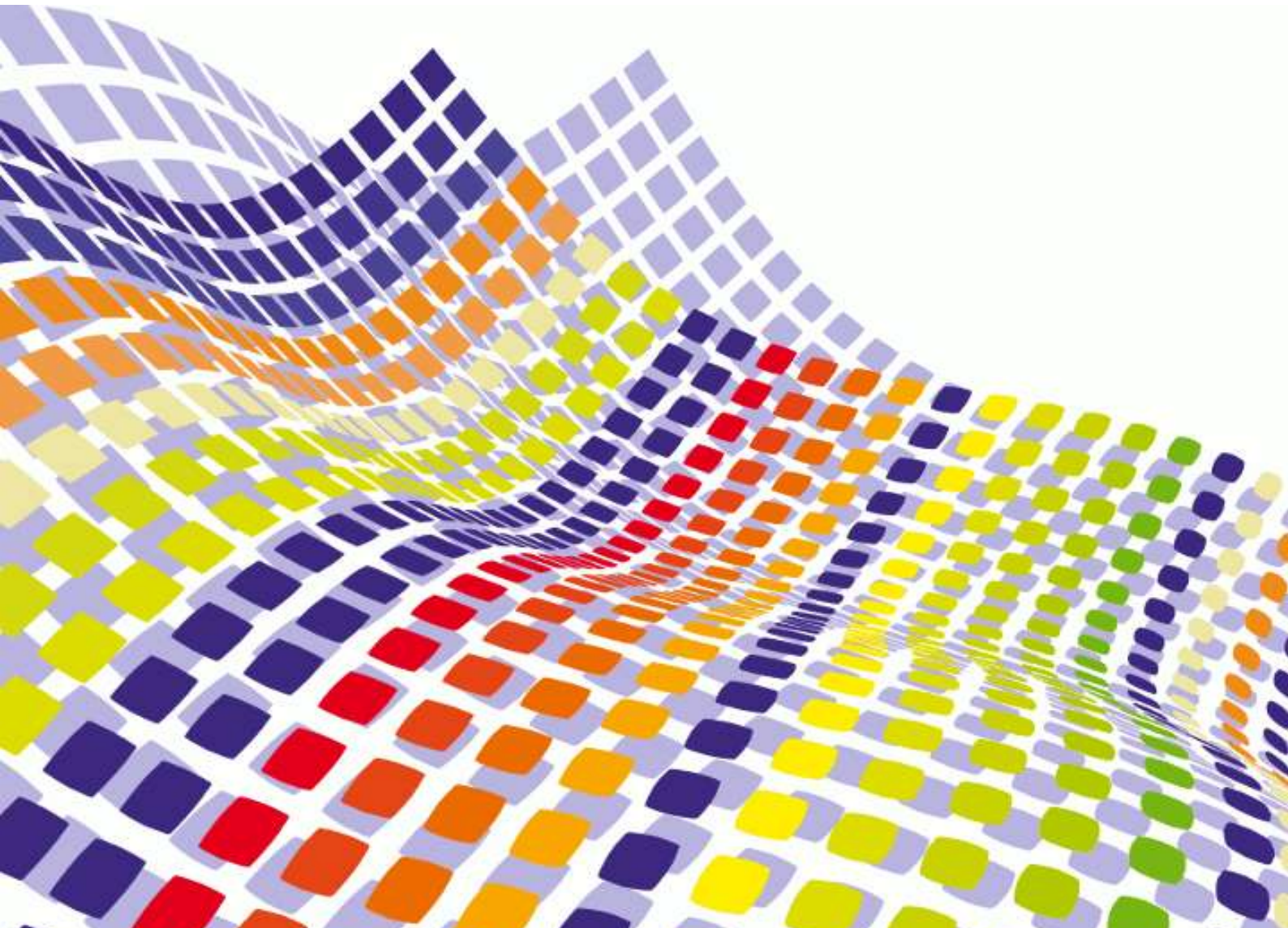
CAPS

hmpp

CAPS OpenACC Compiler

HMPP Workbench 3.2



IDDN.FR.001.490007.000.S.P.2008.000.10600

Headquarters – France
Immeuble CAP Nord
4A Allée Marie Berhaut
35000 Rennes
France

Tel.: +33 (0)2 22 51 16 00
Fax: +33 (0)2 23 20 16 43

info@caps-entreprise.com

N° d'agrément formation :
53 35 08397 35

CAPS – USA
4701 Patrick Drive Bldg 12
Santa Clara
CA 95054

Tel.: +1 408 550 2887 x70

usa@caps-entreprise.com

CAPS – CHINA
Suite E2, 30/F
JuneYao International Plaza
789, Zhaojiabang Road,
Shanghai 200032

Tel.: +86 21 3363 0057
Fax: +86 21 3363 0067

apac@caps-entreprise.com

Visit our website: http://www.caps-entreprise.com

SUMMARY

1. Introduction

1.1. Revisions history

Version	Date	Writer	Modified pages	Revision object
V3.1.0	05/22/2012	CAPS entreprise	All	Initial version
V3.1.1	04/06/2012	CAPS entreprise	All	Documentation update
V3.2.0	26/06/2012	CAPS entreprise	§4.2 §5.2.3	Environment setup for OpenCL program execution Addition of the `-openacc target` clause to order the code generation target
V3.2.1	17/07/2012	CAPS entreprise		Typography corrections
V3.2.3	26/09/2012	CAPS entreprise	§6.2	Document the environement variable ACC_DEVICE_TYPE

1.2. Introduction

In an effort to make it easier for programmers to take advantage of hardware computing accelerators, NVIDIA, Cray Inc., the Portland Group (PGI), and CAPS enterprise have announced during SuperComputing 2011 a new parallel-programming standard, known as OpenACC™.

OpenACC allows parallel programmers to provide simple hints, known as “directives,” to the compiler, identifying which areas of code to accelerate, without requiring programmers to modify or adapt the underlying code itself. By exposing parallel nature of the code to the compiler, directives allow to finely drive how the compiler will map the computation onto the accelerator.

It should be noted that thanks to the HMPP technology, OpenACC annotated applications can either be generated for NVIDIA or AMD GPU according that the generation language is CUDA (default mode) or OpenCL (see chapter 5.2.3, `Specification of the target language` for further details about the selection of the target language).

This document comes in addition of the `OpenACC Application Programming Interface Version 1.0` ([R1]) and it specifies details of the implementation of this standard in HMPP 3.2.x.

The remainder of this document is organized as follow:

- Chapter 2 provides the user with some information regarding the main OpenACC directives,
- Chapter 3 details the implementation defined behavior of the CAPS OpenACC compiler,
- Chapter 4 covers the installation of the CAPS OpenACC Compiler,
- Chapter 5 is dedicated to the compilation flow process,
- Chapter 6 covers HMPP program execution,
- Chapter 7 described the supported platforms and compilers.

A glossary can be found at the end of the document.

1.3. What is HMPP Workbench? What is the CAPS OpenACC Compiler?

In addition of the OpenACC keyword, you will find in this document many references to HMPP, HMPP Workbench or OpenHMPP. HMPP Workbench is the flagship product of CAPS Entreprise and has been available long before CAPS entreprise joined the OpenACC initiative. HMPP Workbench supports the OpenHMPP as well as the OpenACC directive set.

OpenHMPP is a very useful platform to explore topics not covered by OpenACC for many-core support such as data-flow extension as well as topics such as tracing interface, auto-tuning APIs or GPU-accelerated library integration. All these topics are considered by CAPS entreprise as crucial for many-core application deployment and software development tools.

While HMPP Workbench compiler supports both OpenACC and OpenHMPP directive sets, the CAPS OpenACC Compiler only supports OpenACC directives.

So HMPP Workbench and the CAPS OpenACC compilers share many properties as: common compilation commands, license server mechanism, runtime information debugging, etc… and that is why you will find in this document many references to both. To clarify, in this document, the following conventions are used:

- *"OpenACC directives"* designates OpenACC directives as defined in [R1];
- *"OpenHMPP directives"* designates OpenHMPP directives as defined in [R2];
- The *"CAPS OpenACC compiler"* designates CAPS compiler supporting only OpenACC directives;
- The "HMPP Workbench" designates the CAPS suite of tools supporting both OpenACC and OpenHMPP directives;
- *"HMPP applications"* designates applications containing either OpenACC directives, OpenHMPP directives or both;
- *"OpenACC application"* designates applications containing only OpenACC directives;
- *"OpenHMPP application"* designates applications containing only OpenHMPP directives;
- *"HMPP applications"* designates applications containing OpenACC directives or OpenHMPP directives;
- *"HMPP"* may be used as an abbreviation to reference the HMPP Workbench;
- *"hmpp"* may be used in command line to invoke CAPS compiler. Same keyword is used to compile either OpenACC applications either OpenHMPP applications.

For readers which would like to know more about HMPP you will be able to consult the following documentation:

- `HMPP Basics` ([R1]). This document introduces the main HMPP concepts.
- `HMPP Directives, Reference Manual ([R3])`. This manual introduces the main HMPP concepts and describes the HMPP directives. This document is the main document for people who want to discover HMPP.
- `HMPP Linux Manual ([R4])`. This manual describes how to compile and run your application on Linux platforms. It also introduces the compilers and Operating Systems supported;
- `HMPP License Installation Guide ([R5])`. This manual presents the procedure to set the HMPP license on your system.
- `HMPP Alternative directives Introduction ([R6])`. This document describes how to take advantage of existing GPU-accelerated libraries while minimizing the impact of their integration into the source code.

1.4. Execution Model

For the most part, an OpenACC program is a user application which executes on the host and where compute-intensive regions are offloaded to the accelerator device under control of the host (see [R1], chapter *Execution Model* for further details).

The device executes parallel code, which are code blocks containing work-sharing loops.

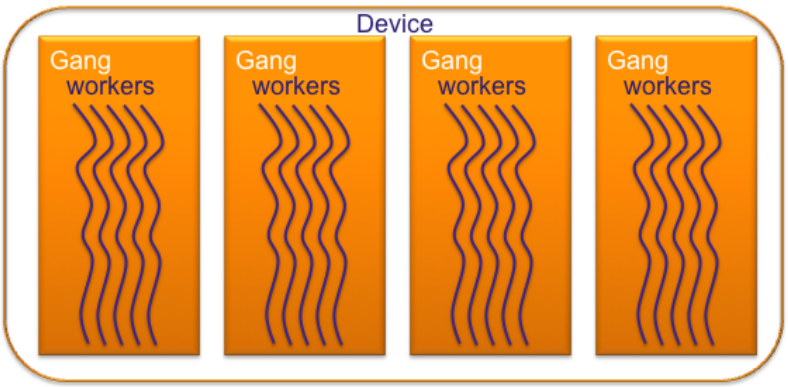
In OpenACC regions, the host stays responsible for the accounting required to execute code on the HWA, that is:

- Memory allocation on the HWA,
- Transferring data between the host and the HWA (and the opposite to get the results of the computation),
- Launching HWA code,
- De-allocating memory.

The parallel execution within the accelerator is divided into three levels of parallelism:

- Coarse grain called “`gang`”;
- Fine grain called “`workers`”, a worker is equivalent to a thread;
- Vectors called “`vector`”.

The gang and workers organization is illustrated below:



1.5. Memory Model

Contrary to e.g. OpenMP programs, which has a shared memory model, OpenACC has a distributed programing model. That is, most[1] HWA’s memory need explicit copies in order to have up-to-date data because HWA memory is not mapped into the host’s address space.

[1] Some HWAs, such as accelerated processing units (APU) share the same memory as the host. Others allow direct access to some of the host’s memory (e.g. Unified Virtual Addressing). In those cases, no explicit copy is required.

2. OpenACC Directives

All OpenACC directives are extensively documented in the specification document [R1]. The present document, "*OpenACC support in HMPP*" will not go into the same kind of details as the specification, but rather focus on the most important directives, and their behavior in HMPP if necessary.

OpenACC offers two different ways to offload computation on HWA:

- Kernel-based: allows the generation of work-sharing kernels from the loops enclosed in a `kernels` region (work sharing is the default),
- Parallel-based: will lead to execution in parallel, on all of a HWA executor, of a section of code until a work sharing section is entered. When a work sharing section is entered the work is shared between all executors (work sharing needs to be made explicit).

2.1. kernels

This directive allows specifying a region of the program that is to be compiled into a sequence of kernels for execution on HWA.

Syntax

In C, the syntax of the OpenACC `kernels` directive is

```
#pragma acc kernels [clause [[,] clause]...] new-line
  structured block
```

And in Fortran, the syntax is

```
!$acc kernels [clause [[,] clause]...]
  structured block
!$acc end kernels
```

Where clause is one of the following (please refer to [R1] for further explanations):

- `if( condition )`
- `async [( scalar-integer-expression )]`
- `copy( list )`
- `copyin( list )`
- `copyout( list )`
- `create( list )`
- `present( list )`
- `present_or_copy( list )`
- `present_or_copyin( list )`
- `present_or_copyout( list )`
- `present_or_create( list )`
- `deviceptr( list )`

An example of the usage of the directive `kernels` is in Listing 1 – An example of the OpenACC `kernel` directive on page 10.

```
#pragma acc kernels, copy(call_result[0:nb_opt], put_result[0:nb_opt], &
#pragma acc & option_strike[0:nb_opt], stock_price[0:nb_opt], option_years[0:nb_opt])
{
  int opt;
  for(opt = 0; opt < nb_opt; opt++) {
    float sqrtT, expRT, K, d1, d2, CNDD1, CNDD2, Riskfree=RISKFREE, Volatility=VOLATILITY;
    sqrtT = sqrtf(option_years[opt]);
    d1 = (logf(stock_price[opt] / option_strike[opt]) + (Riskfree +
            0.5f*Volatility*Volatility)*option_years[opt]) / (Volatility*sqrtT);
    d2 = d1 - Volatility * sqrtT;

    K = 1.0f / (1.0f + 0.2316419f * fabsf(d1));
    CNDD1 = RSQRT2PI * expf(- 0.5f*d1*d1) * (K * (A1 + K * (A2 + K * (A3 + K * (A4 + K * A5)))));
    K = 1.0f / (1.0f + 0.2316419f * fabsf(d2));
    CNDD2 = RSQRT2PI * expf(- 0.5f*d2*d2) * (K * (A1 + K * (A2 + K * (A3 + K * (A4 + K * A5)))));

    expRT = expf(- Riskfree * option_years[opt]);
    call_result[opt] = stock_price[opt] * CNDD1 - option_strike[opt] * expRT * CNDD2;
    put_result[opt]  = option_strike[opt]*expRT*(1.0f - CNDD2) - stock_price[opt]*(1.0f - CNDD1);
  }
}
```

Listing 1 – An example of the OpenACC `kernel` directive

As you can see, the `kernels` directive is not used “plain”, as the `copy` clause is used as well.

They are required to make the size of all arrays that are used in the `kernels` section explicit. In addition, they tell which arrays need to be copied to and/or from the HWA.

When the above code is compiled, the compilation messages would be similar to:

```
hmpp: [Info] Generated codelet filename is acc_region_cuda.hmf.cu".
hmppcg: [Message DPL3000] acc_region:17: Loop 'opt' was gridified (1D)
```

The message `Loop 'opt' was gridified (1D)` is very important: it means that HMPP was able to analyze the loop with loop counter `opt` and concluded that the loop count be parallelized on GPU, that is, gridified.

A counter-example would be compilation message such as:

```
hmppcg: [...] Loop 'opt' not gridified: Inter-iterations dependencies found
```

This message either means that HMPP found loop inter-iteration dependences that prevent automatic parallelization, or could not analyze the loop and marked it as sequential by default.

If you get this message, you should first:

- Check for scalar inter-iteration dependences. See if you could get rid of these dependences by some variable privatization, as you would do in an OpenMP program. For example, in Listing 1, if `sqrtT` was instead defined outside of the `acc` region, this would lead to write-after-write hazards which would prevent a safe parallel execution.
- Check for reductions. These can be dealt with the `reduction` clause of the `parallel` directive.
- Check for array inter-iteration dependences. Some array subscripts are too difficult to analyze by HMPP but aren’t by a programmer’s trained eye.

If you have done all what is needed to make the loop parallelizable, you can then use `independent` clause of the `loop` directive to force the gridification of the loop. Example:

```
#pragma loop independent
for (i=0; i<n; i++) {
  /* loop statements */
}
```

2.1.1. Avoiding some needless transfers: using copyout and copyin clauses

In Listing 1, we have put all arrays of the `acc` region in a `copy` clause. This means that their content is going to be uploaded and downloaded to/from the HWA before and after the `acc` region is executed. It should be noted that end-users need to be aware of the memory size available on the accelerator as well as the memory bandwidth in order to effectively accelerate a region of code.

The approach showed in Listing 1 is not optimal, because some arrays are only read, some are only written to. We can do better by using the `copyin` and `copyout` clauses, as in Listing 2

```
#pragma acc kernels, copyout(call_result[0:nb_opt], put_result[0:nb_opt]), &
#pragma acc & copyin(option_strike[0:nb_opt], stock_price[0:nb_opt], option_years[0:nb_opt])
{
/* statements */
```

Listing 2 – A version of Listing 1 where needless transfers are avoided

2.2. data

While the `kernels` pragma marks a region of code that should be executed on the HWA, the `data` pragma states which and when variables should be allocated and transferred on the HWA (when the data section is entered), and when they should be de-allocated, and transferred to the HWA (when the section is exited).

Quite logically, `kernels` pragma should be in a `data` pragma section, not the other way around.

In C, the syntax of the OpenACC kernels directive is

```
#pragma acc data [clause [[,] clause]...] new-line
{
  ...
  #pragma acc kernels [clause [[,] clause]...] new-line
    structured block
  ...
}
```

and in Fortran, the syntax is

```
!$acc data [clause [[,] clause]...]
  ...
  !$acc kernels [clause [[,] clause]...]
    structured block
  !$acc end kernels
  ...
!$acc end data
```

where clause is one of the following (see [R1] for further explanations):

- `if( condition )`
- `copy( list )`
- `copyin( list )`
- `copyout( list )`
- `create( list )`
- `present( list )`
- `present_or_copy( list )`
- `present_or_copyin( list )`
- `present_or_copyout( list )`
- `present_or_create( list )`
- `deviceptr( list )`

The main purpose of a data section is to avoid multiple re-allocations of GPU memory and transfers between the host and the GPU, which are slow.

Example:

```
// Allocation of data on GPU
#pragma acc data, present_or_copyout(call_result[0:nb_opt], put_result[0:nb_opt]), &
#pragma acc & present_or_copyin(option_strike[0:nb_opt], stock_price[0:nb_opt], &
#pragma acc & option_years[0:nb_opt],nb_opt, VOLATILITY,RISKFREE)
for(i=0; i<NB_RUN; i++) {
  double v0, v1;

  // Call of the kernels: the execution is done on the GPU
#pragma acc kernels, present(call_result[0:nb_opt], put_result[0:nb_opt]), &
#pragma acc & present(option_strike[0:nb_opt], stock_price[0:nb_opt], &
#pragma acc & option_years[0:nb_opt],nb_opt, VOLATILITY,RISKFREE)
  {
    int opt;
    float Riskfree=RISKFREE, Volatility=VOLATILITY;
    for(opt = 0; opt < nb_opt; opt++) {
      float sqrtT, expRT, K, d1, d2, CNDD1, CNDD2;
```

Listing 3 – An example of the OpenACC `data` directive

In Listing 3, a `kernels` section is located in a loop, which means that it will be called multiple times. If there was no `data` section, all the memory (de)allocations on the HWA, transfers will occur when the section is entered and exited, which incurs a non-negligible overhead.

Thanks to the `data` section, and HWA memory is allocated and initialized with the host's value once before the loop, and transferred back to the host and freed when the `data` section ends. Note that in this example, the `kernels` section now uses the `present` clause instead of the `copy*` clauses, because all allocation occur where the `data` section is.

2.3. parallel

This directive is very similar to OpenMP[2]'s parallel directive: that is, when a parallel region is entered, all "executors" of the program will execute all statements of the program, unless a work-sharing section is reached (e.g. an `omp loop`).

In OpenACC terms, this means that when an OpenACC `parallel` region is entered, gangs of workers are created[3] to execute the accelerator parallel region. One worker in each gang begins executing the code of the parallel section.

When a work-sharing `acc loop gang` loop is entered, the iterations of that loop will be distributed between the gangs, and one worker of each gang will be used to perform the computations.

Then, when a work-sharing `acc loop worker` loop is entered, the iterations of that loop will be distributed between the workers of each gang, so that all workers will contribute to perform the computations.

[2] The OpenMP Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures.

[3] Once gangs and workers are created, their number remains the same throughout the region.

In other words, unless there is an `acc loop worker` loop in an `acc parallel` region, no acceleration can be obtained because most of the computing resources of the HWA won't be utilized.

```
!$acc data, copy(C), copyin(A)
DO k=1, 3
   !! some code executed on CPU

   !! the execution is done on the GPU
   !$acc parallel, private(c11, c12, c13, c21, c22, c23, c31, c32, c33)

   c11 = 2.0
   c21 = 5.0
   c31 = 8.0
   c12 = 3.0
   c22 = 6.0
   c32 = 9.0
   c13 = 4.0
   c23 = 7.0
   c33 = 10.0

   !$acc loop gang
   DO i=2, M-1
      !$acc loop worker
      DO j=2, N-1
         C(j,i) = c11 * A(j-1,i-1) &
                + c12 * A(j-1,i) &
                + c13 * A(j-1,i+1) &
                + c21 * A(j,i-1) &
                + c22 * A(j,i) &
                + c23 * A(j,i+1) &
                + c31 * A(j+1,i-1) &
                + c32 * A(j+1,i) &
                + c33 * A(j+1,i+1)
      END DO
   END DO
   !$acc end parallel

   !! some more code executed on CPU
END DO ! k
!$acc end data
```

Listing 4 – An example of the OpenACC `parallel`, `gang`, and `worker` directives

3. Implementation-defined behaviors

In fair deal of cases, [R1] states that *the behavior is implementation defined*. This section collects these cases and mentions also the limitations of the current implementation.

3.1. Internal Control Variables

If the value of `acc-device-type-var` is not explicitly modified, its value depends on whether or not device acquisition has occurred. Before acquisition, the device type is `none`, and will be some kind of HWA once acquisition has succeeded.

If the value of `acc-device-num-var` is not explicitly modified, its value depends on whether or not device acquisition has occurred. Before acquisition, the device number is `1`, and will be the logical number of the device acquired once acquisition has succeeded.

3.2. Reduction clause

In an `acc parallel` region containing some `gang` and/or `worker` acc clauses, if a reduction operation is present, this one must be specified by using the `reduction` clause at the `acc parallel` clause level. This restriction will be lifter in a future release.

3.3. num_gangs clause

The `num_gangs` clause takes an immediate constant value in parameter. The default value is 32.

3.4. num_workers clause

The `num_workers` clause takes an immediate constant value in parameter. The default value is 256.

3.5. vector_length clause

This clause is not taken into account in the current release of HMPP.

3.6. collapse clause

A `collapse` clause cannot be used on any loop of a loop nest with a gang, worker or vector clause. This restriction will be lifter in a future release.

3.7. worker clause

A loop with the worker clause that contains a loop containing the gang clause will trigger a loop permutation to put the gang loop outside of the worker loop.

3.8. vector clause

A loop with the `vector` clause that contains a loop containing the `gang` or `worker` clause will trigger a loop permutation to put the gang loop outside of the worker loop, and the vector loop in the innermost position.

3.9. Declare directive

Currently, the `declare` directive cannot be used with global variables (variables defined in FORTRAN module or in global scope for C language).

3.10. acc_set_device_type

If the device type specified is not available or unsupported, the program will abort.

If the routine is called more than once without an intervening `acc_shutdown` call, with a different value for the device type argument, the behavior will depend on whether or not device acquisition has already occurred or not:

- If device acquisition has not occurred, the device type will be set to the new value,
- If device acquisition has already occurred, the new device type is ignored.

3.11. acc_set_device_num

If the value of `devicenum` is zero, the runtime will revert to its default behavior, which means that the runtime will try to acquire any HWA available and capable of executing the HWA code.

If the value of `devicenum` is greater than the value returned by `acc_get_num_devices` for that device type, the value is ignored and a warning is issued.

3.12. acc_init

If the device type specified is not available, the program will abort.

If this routine is called more than once without an intervening `acc_shutdown` call, with a different value for the device type argument, the behavior will depend on whether or not device acquisition has already occurred or not:

- If device acquisition has not occurred, the device type will be set to the new value,
- If device acquisition has already occurred, the new device type is ignored.

3.13. Environment Variables

If the values of the environment variables change after the program has started, even if the program itself modifies the values, the behavior of the program it not affected.

3.14. Scalar-integer-expression reference in OpenACC clause

HMPP Workbench 3.1 currently supports only constant argument in OpenACC clause directive. References to variables or expressions are not currently not supported. This restriction will be lifter in a future release.

3.15. OpenACC runtime routines

Current version of HMPP Workbench only supports `acc_device_not_host` arguments. This restriction will be lifter in a future release.

4. OpenACC Compiler Installation

The OpenACC Compiler is provided as a simple installer file which can be directly executed in the user's environment.

The package name is: `HMPPOpenACC-<version>_linux64.bin` with:

- `<version>` : current version of the CAPS OpenACC compiler.

4.1. CAPS OpenACC compiler Installation

To install the CAPS OpenACC compiler, launch the following command and follow the wizard instructions:

```
$ ./HMPPOpenACC-3.1.1[4]_linux64.bin
```

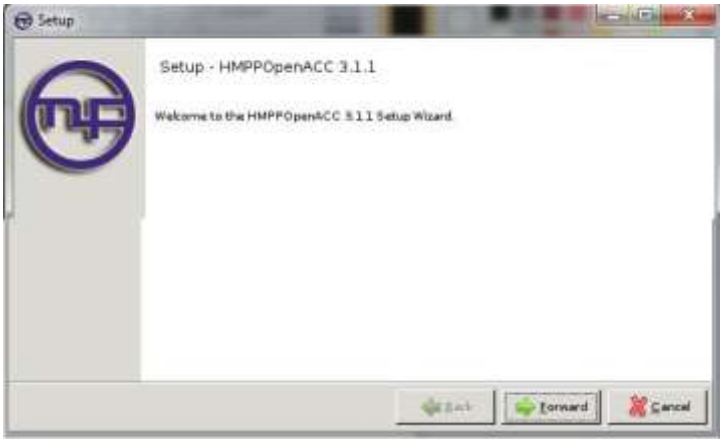


Figure 1 – CAPS OpenACC Compiler Wizard (example given for version 3.1.1)

A text mode is also available:

```
$ ./HMPPOpenACC-3.1.1_linux64.bin --mode text
```

4.2. CAPS OpenACC Compiler Environment Setup

Assuming that the CAPS OpenACC Compiler is installed in `/home/user/HMPPOpenACC-3.1.1`, the CAPS OpenACC environment is set running one of the following shell scripts:

```
$ source /home/user/HMPPOpenACC-3.1.1/bin/hmpp-env.sh
HMPP Workbench environment set
$
```

for sh-like shells, or

[4] Example is given for the CAPS OpenACC Compiler 3.1.1. Please, adapt the command line to the current version to install.

```
$ source /home/user/HMPPOpenACC-3.1.1/bin/hmpp-env.csh
HMPP Workbench environment set
$
```

for csh-like shells.

It should be noted that for OpenCL applications, two environment variables must be explicitly set:

- `OPENCL_INC_PATH`
- `OPENCL_LIB_PATH`

With CUDA SDK, these variables can be set as following

```
export OPENCL_HOME=${CUDA_HOME}/
export OPENCL_INC_PATH=${OPENCL_HOME}/include
export OPENCL_LIB_PATH=${OPENCL_HOME}/lib64
```

With AMD ATI Stream Computing:

```
export OPENCL_HOME=${AMDAPPSDKROOT}
export OPENCL_INC_PATH=${OPENCL_HOME}/include
export OPENCL_LIB_PATH=${OPENCL_HOME}/lib/x86_64
```

4.3. CAPS OpenACC Compiler License

Warning: For a complete description of the HMPP license policy, please refer to document [R5], *"HMPP License Installation Guide"*.

CAPS OpenACC Compiler and HMPP Workbench share the same license mechanism based upon the use of the HMPP License Server to deliver floating license.

To use the CAPS OpenACC Compiler, a valid license is needed. CAPS entreprise provides on request license. According to the type of the license that you ordered, you may have to run the command "`hmpp -licenses`":

- For user-based node-locked: on the system on which the compilation will be done. Please note that your login will be used for the generation of the license.
- For user-based license: on any system you want to use. Please note that your login will be used for the generation of the license.
- For floating-license: on the system on which the HMPP License server will be running.

In all cases, complete the requested information and send them to:

licenses@caps-entreprise.com

HMPP command license:

```
$ hmpp --licenses
```

```
If you need the generation of a new license, the following information will be required by CAPS
entreprise.
You can send them to licenses@caps-entreprise.com.

-------------------------------
First Name:   <TO BE COMPLETED>
Name:         <TO BE COMPLETED>
Company:      <TO BE COMPLETED>
E-Mail:       <TO BE COMPLETED>
Phone number: <TO BE COMPLETED>
Software: HMPP
Version:  xxxx <automaticcally completed>
Platform: xxxx <automaticcally completed>
Username: xxxx <automaticcally completed>
Hostname: xxxx <automaticcally completed>
Ethernet:
 - xxxxxxxxxx <automaticcally completed>
-------------------------------
```

Listing 5 - HMPP License Information example (to launch on the License Server host)

5. Compiling HMPP Applications[5]

The CAPS OpenACC compiler is available for C and FORTRAN languages and can be used to generate CUDA (default mode) or OPENCL code. It is used:

- to preprocess OpenACC annotated applications,
- to extract and to generate HWA code,
- and finally to compile and link the OpenACC application.


Note: All the commands described in this section can be encapsulated in a normal `Makefile` mechanism.

5.1. Overview

In terms of use, the CAPS OpenACC compiler workflow is really close to traditional compilers. However, as illustrated in Figure 2, we can distinguish two main paths:

- The left one (in Figure 2) is dedicated to the compilation of the *main* application which will be executed on the host processor (as in traditional compilers). In this case, we will designate the compiler used under the name *host compiler*,
- The right one (in Figure 2) is dedicated to the HWA code generation and compilation. The OpenACC regions are generated under the form of shared libraries in order to be loaded by the HMPP runtime during the execution of the application. In this case, we will designate the compilers under the name of "CAPS OpenACC *Compiler"* for the generation of the piece of code to be executed on the HWA and the "*hardware vendor compilers"* for the compilation of this code using the provided hardware vendor tools.

[5] In the following, "*HMPP applications*" may designate applications containing either OpenACC directives either OpenHMPP directives or both.

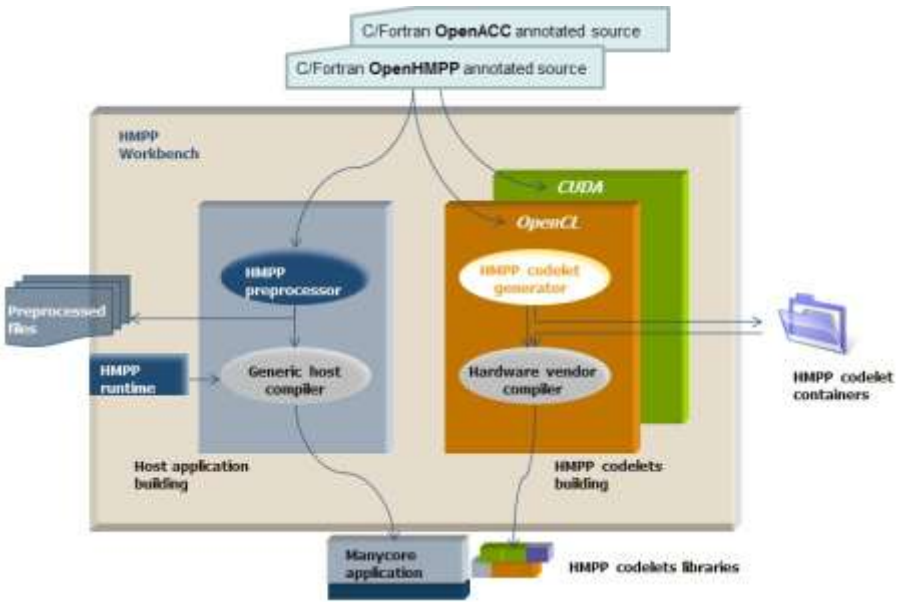


Figure 2 - HMPP Compiler Workflow (global view)[6]

Compiling such program is done by using the `hmpp` command followed by the appropriate compiler depending on the considered language (C or FORTRAN):

```
$ hmpp gcc program.c -o program.exe
```

Or:

```
$ hmpp ifort program.f90 -o program.exe
```

Like with usual compilers, the default output file name is `a.out`.

The `hmpp` commands successively runs the HMPP preprocessor to process the OpenACC directives by inserting calls to the HMPP runtime and then invoke the user's specified native compiler to produce the application executable.

`hmpp` extracts the OpenACC marked code from the application sources and generates their hardware accelerated implementation as shared libraries with the appropriate HMPP codelet generator.

5.2. Common Command Line Parameters

The HMPP compiler runs as follow:

```
$ hmpp [HMPP_OPTIONS]HOST_COMPILER [HOST_COMPILER_OPTIONS]
```

Here is the list of the available command line options for HMPP compilers.

[6] Supported platforms and languages may be dependent of the CAPS's product options subscribed.

5.2.1. General Options

Some noteworthy general options are:

- `-t, --temp DIRNAME`: sets the temporary directory (default is /tmp),
- `-k, --keep`: does not remove temporary files,
- `-d[n]?, --debug [n]?`: set HMPP verbosity. A numerical value can be specified to increase the level of the verbosity of the messages displayed.
- `-c` colorizes the output
- `-g` turns on flags for debugging/profiling during codelet generation/compilation
- The command line below illustrates the use of the `-d` option with a high value of verbosity (level 3).

```
$ hmpp -d3 icc myHMPPApplication.c
```

5.2.2. Host Compiler Options

Most of the standard compiler options are supported by HMPP. These options are directly given on the command line and follow the specification of the compiler.

The "`-d`" HMPP's option can be notified on the command line to increases the level of verbosity of HMPP during the compilation stage. Thus all the commands executed will be displayed allowing the user to check that the right options are given to the compiler.

```
$ hmpp -d ifort -O3 myHMPPApplication.f90
```

Note that "`-E`" option runs preprocessor only. With this option, only the preprocessing of the file is done, resulting in source files where the HMPP directives have been translated into calls to the HMPP runtime. The preprocessed files can then be compiled with the usual general purpose compiler

Compiler options that would change the semantic of the code should not be used. Typical example for FORTRAN compilers are the following:

- `-fall-intrinsics`
- `-fd-lines-as-code`, `-fd-lines-as-comments`
- `-fdefault-double-8, -fdefault-integer-8, -fdefault-real-8`
- `-fmodule-private`
- `-fbackslash`
- `-fcray-pointer`
- `-fdollar-ok`
- …

These options are mainly used to support FORTRAN dialects.

5.2.3. Specification of the target language

In the context of the OpenACC directives, the standard does not currently support the specification of a target language for code generation and by default HMPP generates CUDA code. So, to be able to handle different languages for code generation while remaining compliant with the OpenACC standard, HMPP Workbench command line has been extended in order to be able to specify a target language:

```
$ hmpp --openacc-target [CUDA|OPENCL]
```

Where:

- `--openacc-target`: launch the generation of the specified target code for OpenACC programs where:

- `CUDA`: orders CUDA code generation
- `OPENCL`: orders OPENCL code generation.

Listing 6 show an example of the HMPP compiler command line in the case of an OpenCL code generation from OpenACC directives.

```
$ hmpp --openacc-target OPENCL --force     icc  -std=c89 -I../common -w -O3
            ../common/hmpp_util.o ../common/getopt.o -lm -o BlackScholes_acc.exe
              BlackScholes_acc.c

hmpp: [Info] Generated codelet filename is "__hmpp_acc_region__8i9abfv3_opencl.hmf.cl".
hmppcg: [Message DPL3000] BlackScholes_acc.c:57: Loop 'opt' was gridified (1D)
```

Listing 6 - Example of OpenCL generation from OpenACC directives

5.2.4. Report option

This option provides users with some results of analysis done by HMPP. Currently, HMPP offers:

- `--io-report`: option to display the intents detected by HMPP.

5.2.5. HMPP Codelet Generation Options

These options can be used to modify the default behavior of the `hmpp` command. These options are the following:

- `-f, --force`: forces codelet file overwrite,
- `--codelet-required`: the compilation fails if the codelet(s) cannot be generated

5.2.6. HMPP codelet compilation: proprietary compiler options

In HMPP, the final codelet code is generated with the proprietary hardware accelerator compiler. In some context, it may be useful to forward some specific options to this compiler.

For NVIDIA architecture, options can be passed to the nvcc compiler (NVIDIA CUDA Compiler driver) by using the options `--nvcc-options`.

For example the following command line will forward the options `"ptxas=-v,-arch,sm_13"` to the `nvcc` compiler:

```
$ hmpp -d --nvcc-options -Xptxas=-v,-arch,sm_13 ifort main.f90 -o main.exe
…
hmpp: [Info] Running command: nvcc --cudafe-options --no_warning saxpy_cuda.cu -shared -Xptxas=-v
-arch sm_13 -o saxpy_cuda.so --compiler-options -fPIC
ptxas info    : Compiling entry function '_Z13hmppcg_loop0_ILj32ELj4EEvifPfS0_'
ptxas info    : Used 3 registers, 48+48 bytes smem, 2000 bytes cmem[0], 8 bytes cmem[1]
…
```

Another possible approach is to use an environment variable as for example the NVCCFLAGS.

```
NVCCFLAGS='-O3 -use_fast_math' hmpp gfortran -O3 sgemm1.f90 -o sgemm1.exe
```

5.2.7. HMPP miscellaneous options

Various others options can be used with HMPP:

- `--version`: displays HMPP version number,
- `--full-version`: displays HMPP full version message,
- `-h, --help`: displays an help message and exit,
- `--licenses`: displays information about HMPP licenses found in the system and exit.

5.2.8. __HMPP predefined macro

During compilation the `__HMPP` macro is set by default. Its value is equal to the current HMPP version. For instance `-D__HMPP=20500` for HMPP 2.5.0 or `-D__HMPP=30000 for HMPP 3.0.`

5.3. Environment Variables

5.3.1. HMPP Environment Variables

The following environment variables are taken into account by the HMPP compilers:

- `HMPP_CODELET_COMPILER_CC`: specifies which C compiler to use for the compilation of the codelet only. In some cases, the same compiler may be used for the compilation of the application and the codelets. However, it is possible to specify another compiler for the compilation of the codelets only. If this variable is set, then the specified compiler will be used for that.
- `HMPP_CODELET_COMPILER_CXX`: specifies which C++ compiler to use for compiling and linking generated codelets.
- `HMPP_CODELET_COMPILER_CFLAGS`: specifies some flags to be used when compiling a codelet source file. Useful if a handwritten codelet uses specific libraries.
- `HMPP_CODELET_COMPILERS_LDFLAGS`: specifies some flags to be used when linking a codelet.

5.3.2. OpenACC Environment Variables

Environment variables used in the context of the OpenACC directives are described in [R1].

The `_OPENACC` preprocessor macro is defined. The version used here has value `201111` as referenced in [R1].

The valid values of the environment variable `ACC_DEVICE_TYPE` are defined in the header `$(HMPP_HOME)/include/openacc/openacc.h`, in the enum `acc_device_t`.

For instance, to enable the execution on an OpenCL HWA, you should set it to `acc_device_opencl`. Example:

```
export ACC_DEVICE_TYPE=acc_device_opencl./lab.exe
INFO : Enter    data (queue=none, location=lab.c:170)
INFO : Acquire  (target=opencl)
```

Listing 7 - Example of execution on OpenCL HWA thanks to the `ACC_DEVICE_TYPE` environment variable

5.3.3. Hardware Vendor Environment Variables

The environment variables defined and used by the hardware vendors (NVIDIA, …) are normally not affected by HMPP.

6. Running HMPP Applications

The execution of the application is based on the HMPP runtime library that manages the correct execution of the HMPP application according to the user's environment.

6.1. Launching the Application

Launching a HMPP program is performed as follow on UNIX platforms, with an sh-like shell:

```
$ export HMPPRT_PATH=my_codelets_library_dir
$ ./program.exe
```

The `HMPPRT_PATH` environment variable sets the directory path where the codelets are (`my_codelets_library_dir` in previous example). The HMPP runtime follows a codelet naming convention to find and load the HWA implementation of codelets.

The command `./program.exe` launches the execution of the application.

6.2. Environment Variables

The runtime's behavior can be altered through the use of the following environment variables:

- `HMPPRT_LOG_LEVEL` controls the verbosity level of the runtime. Other environment variables can also be used to control the display of information. These one are described chapter 6.2.1.
- `HMPPRT_NO_FALLBACK` prevents the execution of the default native codelet (or other target codelets) when the execution of the first target codelet failed.
- `HMPPRT_PATH` allows to indicate the path to load HMPP generated dynamic libraries

6.2.1. Controlling Logging

In HMPP 3, some environment variables have been added to help user to analyze the behavior of their HMPP application. These variables are described below.

HMPPRT_LOG_LEVEL

`HMPPRT_LOG_LEVEL` controls the verbosity level of the runtime. The following values are available. Each level includes the previous one.

- `"fatal"`: display fatal messages emitted by the runtime.
- `"error"`: display error messages (includes also fatal ones)
- `"warn"`: display warnings (as well as previous levels)
- `"info"`: display information regarding the HMPP operations
- `"debug"`: display low-level operations, such as calls to the CUDA, OpenCL runtime. Mainly reserved to CAPS usage
- `"all"`: display all messages
- `"off"`: inhibit all messages

```
HMPPRT_LOG_LEVEL=all|debug|info|warn|error|fatal|off
```

HMPPRT_LOG_FILE

This variable allows redirecting the log messages to a file. By default, its value is set to "-" which correspond to the standard error

```
HMPPRT_LOG_FILE=Log_file
```

HMPPRT_LOG_FILE_RESET

If the value is not 0 (default value), the HMPPRT_LOG_FILE is reset when the runtime is initialized. This works only when the HMPPRT_LOG_FILE is not the standard output.

```
HMPPRT_LOG_FILE_RESET=integer
```

HMPPRT_LOG_FILE_HEADER

Replace the default header ("`Starting HMPPRT logging…`") by the new specified value (is not effective with standard output).

```
HMPPRT_LOG_FILE_HEADER=string
```

HMPPRT_NO_TIMESTAMP

If the value is not 0 (default value) the display of the timestamp is inhibited.

```
HMPPRT_NO_TIMESTAMP=integer
```

HMPPRT_NO_THREAD_ID

If the value is not 0 (default value) the display of the number of threads is inhibited.

```
HMPPRT_NO_THREAD_ID=integer
```

HMPPRT_NO_COLOR

If the value is not 0 (default value) the display of the color of the HMPP message is inhibited.

```
HMPPRT_NO_COLOR=integer
```

6.2.2. HMPPRT_NO_FALLBACK Environment Variable

When a codelet execution fails for a target, this variable prevents the execution of all other implementations of the codelet (when several targets are specified) or the native version.

Note that when this variable is set, if the execution of a codelet fails, the application exists with a returned value equal to the one set to `HMPPRT_NO_FALLBACK` variable.

To set this variable:

```
$ export HMPPRT_NO_FALLBACK=<ValueToBeReturned>
```

To unset this variable:

```
$ export HMPPRT_NO_FALLBACK=0
Or
$ unset HMPPRT_NO_FALLBACK
```

6.2.3. HMPPRT_PATH Environment variable

By default, the HWA code generated by HMPP (`.hmc`, `.hmg`, or `.hmf` files,) are searched in the current directory. To specify other directories, the environment variable `HMPPRT_PATH` can be used.

A list of directory can be specified separated by "`:`"

```
$ export HMPPRT_PATH=directory1:directory2
```

From the example above, "directory1" then "directory2" will be searched before taking into account the current directory to get the HMPP generated dynamic library.

6.3. Running OpenACC Applications

Once compiled, an OpenACC application behaves pretty much like a normal HMPP application. This means that, like an HMPP application, a codelet file (.hmf, .hmc, or .hmg) is generated, and that the same environment variables can be used to control execution. For example Listing 8 illustrates the execution of an OpenACC program compiled with HMPP 3.1.x (verbosity on).

Please refer to [R4] for detailed information.

```
HMPPRT_LOG_LEVEL=info HMPPRT_NO_TIMESTAMP=1 HMPPRT_NO_THREAD_ID=1 ./black.exe --dim=100000

INFO : Enter    data (queue=none, location=black.c:179)
INFO : Acquire  (target=CUDA)
INFO : Allocate call_result[0:100000] (element_size=4, host_address=0x7fb9da137010, memory_space=cudaglob,
queue=none, location=black.c:179)
INFO : Allocate put_result[0:100000] (element_size=4, host_address=0x7fb9da073010, memory_space=cudaglob,
queue=none, location=black.c:179)
INFO : Allocate option_strike[0:100000] (element_size=4, host_address=0x7fb9d744d010, memory_space=cudaglob,
queue=none, location=black.c:179)
INFO : Upload   option_strike[0:100000] (element_size=4, host_address=0x7fb9d744d010, queue=none,
location=black.c:179)
INFO : Allocate stock_price[0:100000] (element_size=4, host_address=0x7fb9d74af010, memory_space=cudaglob,
queue=none, location=black.c:179)
INFO : Upload   stock_price[0:100000] (element_size=4, host_address=0x7fb9d74af010, queue=none,
location=black.c:179)
INFO : Allocate option_years[0:100000] (element_size=4, host_address=0x7fb9d73eb010, memory_space=cudaglob,
queue=none, location=black.c:179)
INFO : Upload   option_years[0:100000] (element_size=4, host_address=0x7fb9d73eb010, queue=none,
location=black.c:179)
INFO : Allocate nb_opt[0:1] (element_size=4, host_address=0x7ffff076cf98, memory_space=cudaglob, queue=none,
location=black.c:179)
INFO : Upload   nb_opt[0:1] (element_size=4, host_address=0x7ffff076cf98, queue=none, location=black.c:179)
INFO : Allocate VOLATILITY[0:1] (element_size=4, host_address=0x40e598, memory_space=cudaglob, queue=none,
location=black.c:179)
INFO : Upload   VOLATILITY[0:1] (element_size=4, host_address=0x40e598, queue=none, location=black.c:179)
INFO : Allocate RISKFREE[0:1] (element_size=4, host_address=0x40e594, memory_space=cudaglob, queue=none,
location=black.c:179)
INFO : Upload   RISKFREE[0:1] (element_size=4, host_address=0x40e594, queue=none, location=black.c:179)
INFO : Enter    kernels (queue=none, location=black.c:190)
INFO : Call     __hmpp_acc_region__qlk0ufbb (queue=none, location=black.c:190)
INFO : Leave    kernels (queue=none, location=black.c:190)
INFO : Free     RISKFREE[0:1] (element_size=4, host_address=0x40e594, queue=none, location=black.c:179)
INFO : Free     VOLATILITY[0:1] (element_size=4, host_address=0x40e598, queue=none, location=black.c:179)
INFO : Free     nb_opt[0:1] (element_size=4, host_address=0x7ffff076cf98, queue=none, location=black.c:179)
INFO : Free     option_years[0:100000] (element_size=4, host_address=0x7fb9d73eb010, queue=none,
location=black.c:179)
INFO : Free     stock_price[0:100000] (element_size=4, host_address=0x7fb9d74af010, queue=none,
location=black.c:179)
INFO : Free     option_strike[0:100000] (element_size=4, host_address=0x7fb9d744d010, queue=none,
location=black.c:179)
INFO : Download put_result[0:100000] (element_size=4, host_address=0x7fb9da073010, queue=none,
location=black.c:179)
INFO : Free     put_result[0:100000] (element_size=4, host_address=0x7fb9da073010, queue=none,
location=black.c:179)
INFO : Download call_result[0:100000] (element_size=4, host_address=0x7fb9da137010, queue=none,
location=black.c:179)
INFO : Free     call_result[0:100000] (element_size=4, host_address=0x7fb9da137010, queue=none,
location=black.c:179)
INFO : Leave    data (queue=none, location=black.c:179)
```


Listing 8 - Execution of the OpenACC application with HMPP (verbosity on)

7. Supported Platforms and Compilers

7.1. Hardware Accelerators

7.1.1. HMPP CUDA generator

For CUDA generator, HMPP support starts from version CUDA Toolkit 4.0 (May 2011, [R7]).

7.2. Supported Operating Systems

HMPP works with any kernel 2.6 Linux distribution containing the `libc` library that comes with g++ 4.x and above,

Below is the list of validated operating systems. It should be noted that by default HMPP inherits the same portability features than those of the SDK used.

For NVIDIA architecture:

- Debian 5.0 and above;
- RedHat Entreprise Linux 5.3 and above;
- OpenSuse 11.1, 11.2, 11.3;
- Suse Linux Entreprise Server 11.0;
- Fedora 10;11, 12, 13
- Ubuntu 9.10, 10.04

For AMD ATI Stream architecture:

- OpenSuse 11.*
- Ubuntu 10.0*
- Red Hat® Enterprise Linux® 6.*

..Warning: To support dedicated hardware, proprietary hardware drivers should be correctly installed.

7.3. Compilers

The HMPP preprocessor takes ANSI C99, GNU C and FORTRAN source as input.

While most standard compilers should work with HMPP, below is the list of currently validated compilers:

- GNU gcc 4.1+,
- GNU gfortran 4.3+,
- Intel Compiler 9.1+,
- Open64 4.2.2+. It should be noted that Open64 offers a limited support of C99 codes.
- Absoft Pro Fortran Compiler version 11.1.

Please note that compiler versions mentioned in this chapter must be matched with versions of compilers supported by the underlying HWA constructor SDK.

8. Annexes

Annex 1. Glossary

HMPP	A short name for HMPP development workbench
HWA	Hardware Accelerator : e.g. a graphic card
OpenACC directives	Designates OpenACC directives as defined in [R1]
OpenHMPP directives	Designates OpenHMPP directives as defined in [R2];
CAPS OpenACC compiler	Designates CAPS compiler supporting only OpenACC directives
HMPP Workbench	Designates the CAPS suite of tools supporting both OpenACC and OpenHMPP directives
HMPP applications"	Designates applications containing either OpenACC directives either OpenHMPP directives either both
OpenACC application	Designates applications containing only OpenACC directives
OpenHMPP application"	Designates applications containing only OpenHMPP directives
HMPP applications	Designates applications containing OpenACC directives or OpenHMPP directives
hmpp	May be used in command line to invoke CAPS compiler. Same keyword is used to compile either OpenACC applications either OpenHMPP applications

Annex 2. Bibliography

[R1]	The OpenACC™ Application Programming Interface, Version 1.0 November, 2011. Available online on the website http://www.openacc-standard.org/
[R2]	HMPPWorkbench-3.2_Basics.pdf, CAPS entreprise
[R3]	HMPPWorkbench-3.2_HMPP_Directives_ReferenceManual.pdf, CAPS entreprise.
[R4]	HMPPWorkbench-3.2_Linux_Manual.pdf, CAPS entreprise
[R5]	HMPPWorkbench-3.2_LicenseInstallationGuide.pdf, CAPS entreprise
[R6]	HMPPWorkbench-3.2_HMPPALT_Directives_Introduction.pdf, CAPS entreprise
[R7]	NVIDIA developers site, http://developer.nvidia.com/object/cuda.html
[R8]	The AMD APP SDK v2.7 with OpenCL™ 1.2 support http://developer.amd.com/sdks/amdappsdk/downloads/pages/default.aspx