

Heterogeneous Habanero-C

The Heterogeneous Habanero-C (H2C) language under development in the [Habanero](#) project at Rice University provides an implementation of the [Habanero](#) execution model for modern heterogeneous (CPU + GPU) architectures.



- [Overview](#)
- [H2C Language Summary](#)
 - [Communication Constructs](#)
 - [Computation Constructs](#)
 - [Synchronization Constructs](#)
- [H2C Compiler and Runtime Framework](#)
 - [Two Phase Compilation](#)
 - [Runtime](#)
- [Example H2C program](#)

Overview

The Heterogeneous Habanero-C (H2C) *language, compiler and runtime* framework is specifically designed to achieve **portability**, **productivity** and **performance** on modern heterogeneous (CPU+GPU) architectures. The main goal is to take a machine independent program written in H2C and generate a machine specific executable.

Some highlights of H2C include:

1. Minimal, intuitive, language extensions makes it easier to write new programs and port existing programs.
2. Two stages compilation targets both domain experts and ninja parallel programmers.
3. **Shared Virtual Memory (SVM)** supports recursive pointer data structures on the GPU.
4. **Meta Data layout** framework generates target specific data layout.
5. Embedded DSLs for stencils and re-use patterns take advantage of local scratchpad buffers.

H2C requires the underlying platform to support [OpenCL](#). The H2C compiler relies on a *Machine Description* (MDes) file. It can be provided by the user or automatically generated by an auto-tuner. H2C uses an *offload* model wherein the CPU is the host and both CPUs and GPUs are devices.

A short summary of the H2C framework is included below. Details on the underlying implementation technologies can be found in the Habanero [publications](#) web page. The H2C implementation is still evolving. If you would like to try out H2C, please contact one of the following people: [Deepak Majeti](#), or [Vivek Sarkar](#).

H2C Language Summary

The language constructs are classified into communication, computation and synchronization constructs.

Communication Constructs

The *async* construct, is used to asynchronously transfer data among multiple devices. One can easily overlap computation with the asynchronous data transfers.

The *finish* statement, *finish <stmt>*, ensures all the data transfers within *<stmt>* have completed.

```
async [copyin (var1, var2, ...)] [copyout (var1, var2, ...)] [at (dev1, dev2, ...)] [partition (ratio)];
```

- 'copyin' clause is used to specify the data that needs to be copied to the device from the host
- 'copyout' clause is used to specify the data that needs to be copied to the host from the device
- 'at' clause is used to specify the targeted devices
- 'partition' clause is used to specify the ratio of partition

Computation Constructs

The *forasync* construct is a data/task parallel loop. It is the programmer's responsibility to ensure that loop iterations are independent.

```
forasync [in (var1, var2, ...)] [point (ind1, ind2, ...)] [range (siz1, siz2, ...)] [seq (se
q1, seq2, ...)] [scratchpad (var1, var2, ...)] [at (dev1, dev2, ...)] [partition (ratio)] {Body}
```

- 'point' clause is used to specify the loop indices in each dimension
- 'range' clause is used to specify the number of iterations in each dimension
- 'seq' clause is used to specify the tile size or the work-group size
- Body represents the loop iteration

Synchronization Constructs

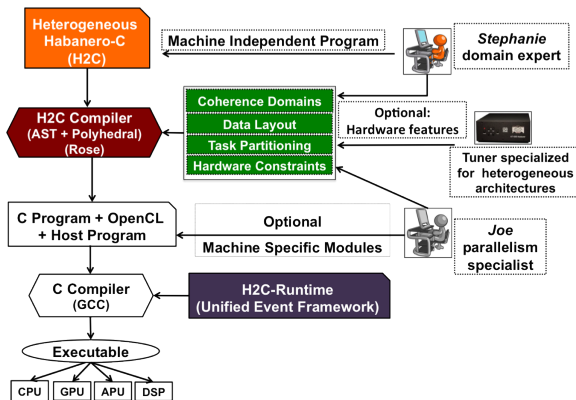
The *finish* construct ensures all the tasks spawned inside it are completed.

H2C Compiler and Runtime Framework

Two Phase Compilation

In the first phase, the H2C compiler translates a H2C program down to a C program, OpenCL kernel and the corresponding host program. Parallelism experts can optionally choose to add optimized OpenCL kernels. The compiler uses a Machine Description(MDes) file to generate a target specific OpenCL kernel and communication. The MDes file can either be specified by the programmer or automatically generated by an auto-tuner.

In the second phase, a standard C compiler is used to build an executable from the generated intermediate files along with the H2C runtime and OpenCL runtime.



Runtime

The H2C runtime includes a memory manger, scheduler and interfaces with the OpenCL runtime.

Example H2C program

Matrix Multiply in H2C

```
finish{
    async copyin(a,b) at(dev);
    foo(); //asynchronously copy data while executing foo
}
finish{
    forasync in(a,b,c,m,n,p) point(i,j) range(0:m,0:n) seq(4,128) shared(a,
b) at(dev){
        float temp =0;
        for(int k=0;k<p;k++){
            temp += a[i*p+k]*b[k*n+j];
        }
        c[i*n+j] =temp;
    }
}
finish{
    async copyout(c) at(dev);
    bar(); //asynchronously copy data while executing bar
}
```

Current H2C limitations

There are some limitations and pitfalls in the current implementation of the H2C programming model. These limitations are not inherent to the programming model, but rather are a result of incompleteness in the current compiler or runtime implementation.

- 1) The forasync construct cannot be nested in the current implementation.
- 2) There is no compiler check for correctness on the forasync body. Any code pattern not supported on the target device will result in runtime errors.
- 3) Pointer arithmetic on arrays which are communicated between CPU and GPU is not supported.

Installation

H2C Branch: <https://svn.rice.edu/r/parsoft/src/Habanero-C/branches/hcConcordBranch>

Dependencies:

Rose Version: ROSE 0.9.5a: orion.cs.rice.edu/~vc8/habanero-git-repo/hc/ROSE.git

EDG with H2C keywords: orion.cs.rice.edu/~vc8/habanero-git-repo/hc/ROSE-EDG.git

branch.dm14-hc-forasync.remote=origin

branch.dm14-hc-forasync.merge=refs/heads/dm14-hc-forasync

Boost Version: boost_1_38_0

Polyopt 0.2.1: <http://web.cs.ucla.edu/~pouchet/software/polyopt/>

Steps:

1. Build **boost_1_38_0**
2. Build **Rose 0.9.5a**
3. Build **Polyopt**
4. run **h2cpolyopt.sh** script located in H2C branch. This script copies the necessary files from the polyhedral installation
5. Build **H2C branch** using the makefile provided

Acknowledgement

Partial support for Heterogeneous Habanero-C was provided through the CDSC program of the [National Science Foundation](#) with an award in the 2009 Expedition in Computing Program.