Programming HPX-5 with Concurrent Collections

Buddhika Chamith, Andrew Lumsdaine, Ryan Newton, CREST, Indiana University

Abstract

Concurrent Collections (CnC) is a data flow centric programming model for expressing task parallel computation. Habanero unified CnC implements this model with multiple task execution runtimes like Open Community Runtime (OCR) and Intel TBB (based on Intel CnC) as back-ends. This project incorporates HPX-5 runtime as a back end runtime for Habanero unified CnC. This enables HPX programming via a higher level CNC abstraction. Preliminary execution time results of several CNC applications running on the new HPX-5 implementation is presented.

CnC Programming Model

The CnC programming model doesn't express parallelism explicitly. The programmer declaratively specifies the dependencies among the computational tasks through their inputs and outputs. Runtime scheduler need to determine how these dependencies are met while maximizing parallelism..

CnC programs are specified with two ordering constraints between tasks as given below.

1. producer/ consumer - Consumer waits for the producer until required input data is available
2. controller/ controlee - Controller determines if the controlee needs to be executed or not. This is specified with a task A 'prescribes' task B relationship in the graph specification.

"These constrains are enough to determine semantically correct parallel or sequential execution orders" [2].

CnC programming model is primarily based on the concept of collections which are key value data structures.

1. Item collection - Stores input and output data items needed/ generated by tasks. These data are immutable.
2. Step collection - Comprises of tasks to be performed

Each of these collections are keyed with tags for distinguishing between different instances of item and step entities corresponding to different task instances.

HPX-5 Integration

HPX-5 is a distributed task runtime library based on ParallelX runtime model. HPX-5 provides a global name and address space which serves as execution contexts for processes with multiple parallel tasks
which may span multiple nodes. Local Control Objects (LCO) such as
futures, semaphores, gates etc. are provided for task coordination and
synchronization.

Internally CNC constructs are mapped to following HPX-5 entities.

* Step collections – HPX tasks
* Item collections – Implemented using future LCO's to enforce producer/ consumer relationship between data dependent tasks

CnC Graph Specification

Habanero CnC uses a graph DSL to declare the CnC specification. All CnC graph specifications have the following structure:

- Begins with an optional ***«context»*** declaration. This is the global context of the application for storing application parameters and is available to all the tasks.

```
$context {
 int n, k;
};
```

- Followed by zero or more ***«item-collection»*** declarations with following format.

[«data-type» «collection-name» : «key-id0», «key-id1», ...] ;

The tag is made up of a tuple of key-ids each of which is an integer value. Single value collections are represented with a zero-ary tuple. (Note: What follows '//' are comments).

```
[ double *cells: i, j ]; // Tag is (i,j)
[ struct timeval startTime: () ]; // Zero-ary tuple
```

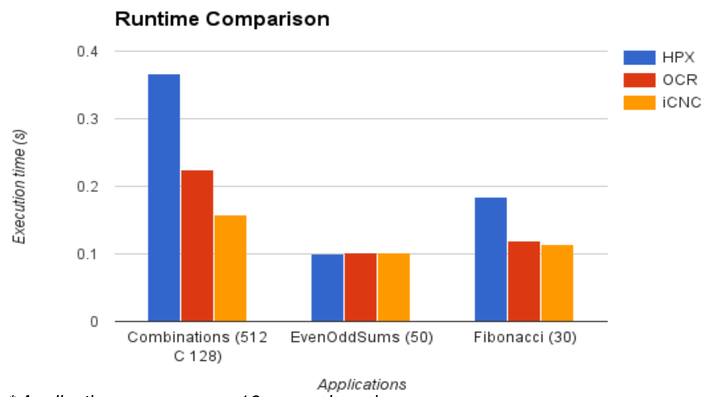
- One or more ***«function-io»*** declarations. These are step collection specifications with data dependencies and prescribe relationships.

```
( compute: i, j) // Step task with tag (i,j)
  <- [ cells: i-1, j ] $when(i > 0), // Conditional input
        // when i>0 from cells item collection at tag (i-1, j)
  -> [ cells: i, j  ]; // Ouptut to cells at tag i,j
  (compute i+1, j+1) // Prescribe compute with next
                     // tag (i+1, j+1)
```

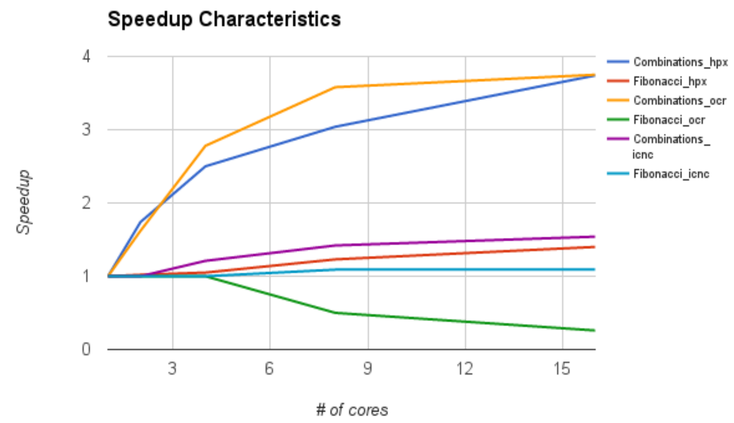
Example - Fibonacci

```
$context {
  int n;  // Fibonacci number needed to be computed
};

[long* fib: i]; // Item array to hold Fibonacci 1..n.

( $initialize: () )
 -> (compute_fib: $rangeTo(0, #n) ); // Prescribes tasks to
        // compute Fibonacci 0 to n. rangeTo specifies the
        // range. Tag range is also overloaded to 0 to n

( compute_fib: i)
 <- [ x @ fib: i-2 ] $when(i>1),
    [ y @ fib: i-1 ] $when(i>1)
 -> [ z @ fib: i   ]; // compute_fib logic contains the computation
    // to calculate ith Fibonacci number which is then output to fib.

( $finalize: () ) <- [ fib: #n ]; // #<val> used to refer parameters
    // from the context. Gets nth Fibonacci number from the fib
    // collection as the input.
```

Backend Comparison



** Applications were run on 16 cores shared memory*



** Speedup of CNC applications with different back-ends*

Summary

CNC programming model allows task parallel computations to be described as data flow graphs. It makes the task of writing safe parallel programs easier since parallelism need not be specified explicitly and the immutability of shared data is guaranteed. The backend runtime is responsible for scheduling parallel tasks with optimal parallel efficiency while satisfying the data flow dependencies given in the specification. With the HPX runtime integration in Habanero unified CNC now it is possible to run CNC programs on HPX. Source is available at [6].

References

1] Budimlić, Z.; Burke, M.; Cavé1, V.; Knobe, K.; Lowney, G.; Newton, R.; Palsberg, J.; Peixotto1, D.; Sarkar, V.; Schlimbach, F.; Taşırlar, S. (2010). "Concurrent Collections" Scientific Programming (IOS Press)
[2] https://software.intel.com/sites/landingpage/icc/api/tutorial.html
[3] https://github.com/habanero-rice/cnc-framework/wiki/Graph%20Spec%20DSL
[4] https://wiki.rice.edu/confluence/download/attachments/17735877/CnC%20Overview.pdf
[5] http://hpx.crest.iu.edu/
[6] https://github.com/chamibuddhika/cnc-framework