CnC-Python: Multicore Programming with High Productivity

Shams Imam, Vivek Sarkar.
Rice University.

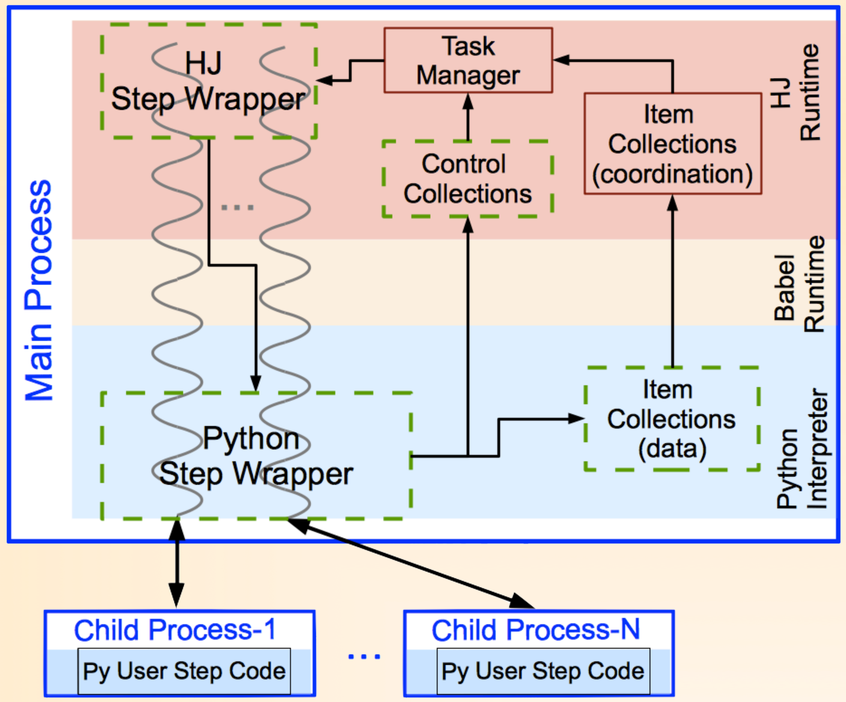
http://cnc-python.rice.edu
http://habanero.rice.edu

1. Background

- Python provides high productivity
- Standard Python's Global Interpreter Lock serializes multithreaded computations
- Concurrent Collections (CnC) programming model enables domain experts to specify dependences among sequential computational units
- CnC-Python runtime implicitly maps available parallelism onto given hardware

2. Implementation

- Leverage multithreading capabilities of Habanero-Java using Data-Driven Tasks
- Java-Python interoperability supported by Babel runtime
- Deliver parallelism in Python by transparently managing multiple instances of the Python interpreter



3. Preliminary Results

- SumPrimes benchmark: Speedup of CnC-Python w.r.t. ParallelPython

# of Workers	2	4	6	8
Parallel Python	125.63 s	63.61 s	42.24 s	32.19 s
CnC-Python	33.86 s	11.54 s	6.88 s	5.41 s
Speedup Factor	3.71	5.51	6.14	5.95

- Scalability of Cholesky Decomposition in CnC-Python (1000x1000 array)

Tile Size	# of Workers			
	2	4	6	8
25	230.76 s	88.36 s	54.74 s	39.97 s
50	244.71 s	82.66 s	50.47 s	36.51 s
100	238.44 s	82.08 s	51.61 s	39.53 s
125	237.17 s	83.28 s	54.02 s	42.88 s

Timings include serial initialization cost along with parallel kernel.

4. Future Work

- Minimize data serialization costs while scheduling computational units
- Add support for Matlab, C, C++ to enable multi-language implementation of computational units (in addition to Python & Java)