# Porting CoMD to HPX

Bibrak Qamar Chandio
CSCI-Y 790 Spring 2016
Submitted to Prof. Dr. Thomas Sterling

This report talks about the efforts and contributions made to port CoMD—a molecular dynamics code [1]—to the experimental High Performance ParalleX (HPX) runtime system. It also presents early performance results for the HPX version of CoMD, hereafter referred to as CoMD-hpx. A direct result of this effort is the contribution that CoMD-hpx is being released as part of **HPX-5 3.0.0**. Currently, the aim of the release is to have feature complete port of CoMD to HPX, as such it is not geared towards performance. Nevertheless, CoMD-hpx achieves 20% the performance of reference CoMD-mpi for 128 cores on Big Red II. More on the performance is provided later in the report.

This report is divided into 3 sections. The first section talks about CoMD and in general the molecular dynamics application and its internals. In the second section we dive into design choices and implementation of CoMD's port to HPX. Finally in the third section we present some early performance results and conclude the report with future work.

## 1) Molecular Dynamics

CoMD [2] is a reference implementation of typical classical molecular dynamics algorithms and workloads. CoMD-hpx uses the code base of CoMD which is created and maintained by ExMatEx: Exascale Co-Design Center for Materials in Extreme Environments (exmatex.org).

In molecular dynamics simulations a material is represented by atoms and molecules. Throughout the simulation the method involves evaluating force that acts on each atom due to all other atoms in the modelled system and the numerical integration of Newtonian equations of motion for each of those atoms. The energy of the modelled system is expressed in terms of potential and kinetic energy of the atoms.

**Validation Criteria:** The validation criteria for correctness of the code and the model is that the total energy should conserve and no atoms be lost. That is the total energy and the number of atoms before and after should remain the same.

**Domain Decomposition:** CoMD decomposes atoms spatially across Cartesian coordinate system. From a computer science perspective this Cartesian coordinate system is mapped across computing node. Since it is practically inefficient to compute all atom interaction, CoMD uses what is called a cutoff distance for each atom and only computes the interactions for atoms in its cutoff distance, see Figure 1. Each time an atoms moves in Cartesian coordinate system it is sent to the computing node that maps the new points on the Cartesian coordinate system. To make things easy the cutoff distance is much smaller than domain of the computing node.

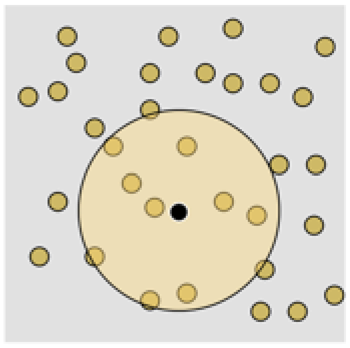

*Figure 1: Atoms and Cutoff distance*

This along with the design that atoms be packed in cubic cells, which have size larger than that of cutoff distance, helps make identification of atoms pairs which are within cutoff distance efficient. Figure 2 shows the system of Figure 1 decomposed into cubic cells.

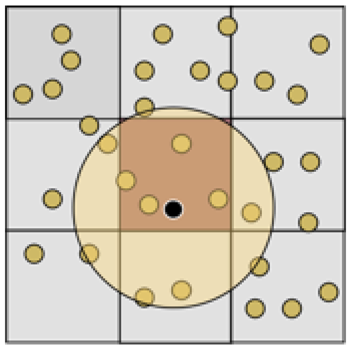

*Figure 2: Further division into cubic cells*

**Communication:** When the system is modeled in three dimensions there are in total 26 such neighbouring cubic cells for each cubic cell. For an atom only its neighbouring cells are needed to be tested and for that only 26 cubic cells are communicated. The actual communication of 26 neighbouring cells is accomplished by only 6 messages sent in 3-steps. First x-faces, followed by y-faces and then z-faces are sent.

### 2) CoMD-hpx Implementation

CoMD-hpx is written from the source code of CoMD-mpi, as a consequence current port of CoMD to HPX mimics MPI-kind of Bulk Synchronous Model (BSM) style. It is worth noting here that not much programming effort was needed (once the programmer gets well acquainted to HPX) to convert such an MPI code to HPX. This demonstrates the potential of HPX API for its productivity and ease when it comes to transitioning legacy MPI code to new adaptive runtime systems.

Following are the three main changes (commandments) that were needed to transition from CoMD-mpi to CoMD-hpx.

I. All data to be allocated in HPX global memory.
II. MPI collective communication to replace HPX collectives.
III. MPI send_recv to replace HPX asynchronous remote action invocation.

Allocating data in HPX global memory essentially makes the transition from MPI ranks to HPX localities. Under the MPI model each node has a rank and allocates data on its local storage. When we allocate memory in the global address space of HPX we essentially allocate data that will be acted upon by different localities (this can or cannot be on the same node). When we invoke action on this HPX global address it gets executed on the locality where the actual data resides.

The MPI collectives are implemented using HPX futures and *hpx_process_collective_allreduce_**. As for the author's knowledge these collectives are not very optimized and work on performance centric algorithms is underway at CREST.

Finally, for communicatimg cubic link cells, MPI_Sendrecv is implemented in CoMD-hpx using HPX's parcels that encapsulates data and sends it to remote locality where it is invoked as a function. The synchronization mechanism involves two LCOs. The first one is an AND gate that ensures that it has received all 6 messages from its neighbour localities and it can now proceed with the simulation with the newly received data. The second LCO is a future that is used to check at the receive side whether this message is designated for the current generation/epoch of Sendrecv messages. It could be a case that the message that has arrived might be from a neighbouring locality that is ahead of this locality and this message is actually from another epoch.

### 3) Performance Results

As already stated, the aim of the release is to have feature complete port of CoMD to HPX, as such it is not geared towards performance. Nevertheless, this section shows some promising performance results for the CoMD-hpx as compared to the original CoMD-mpi. The simulation was run for 100 time steps modeling a total of 4,000,000 atoms. Performance numbers were collected on both the platforms (MPI and HPX) on a total of 128 cores. The computer used was Indiana University's Big Red II. These 128 cores represent 4 nodes of 32 cores each.

Figure 3 shows the running time in seconds and Figure 3 shows the corresponding speedup gained. CoMD-hpx starts off slow on 1 locality but as the number of nodes/cores increases it quickly catches up CoMD-mpi.

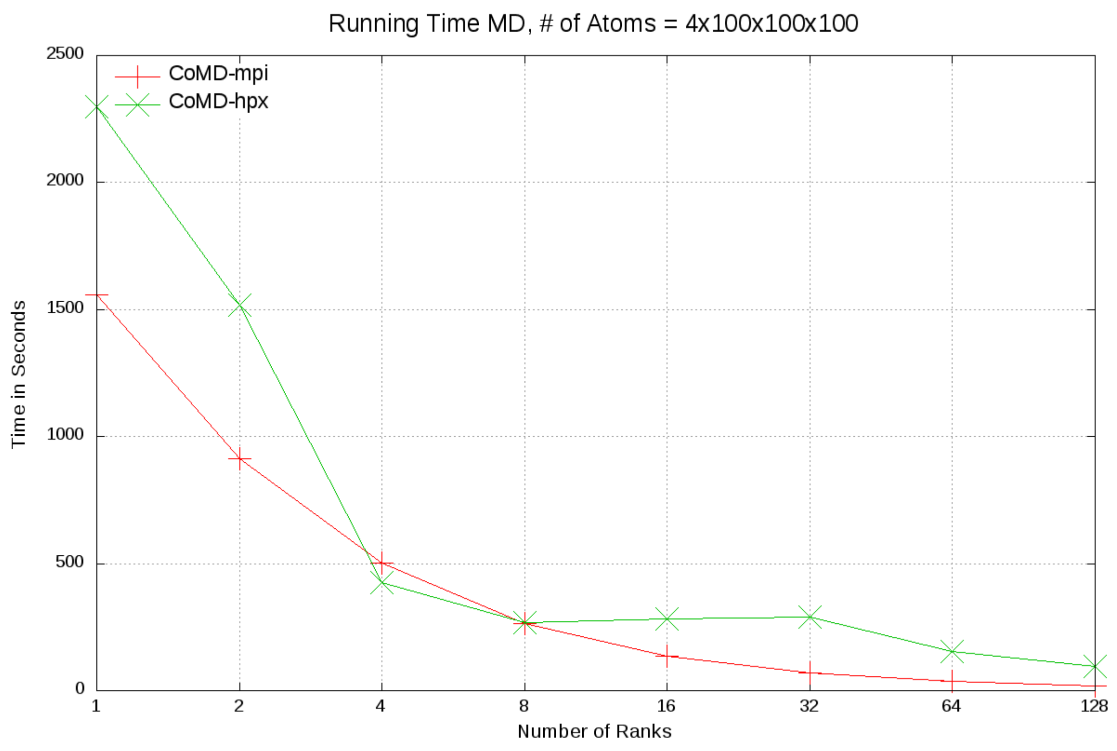

*Figure 3: Running time*

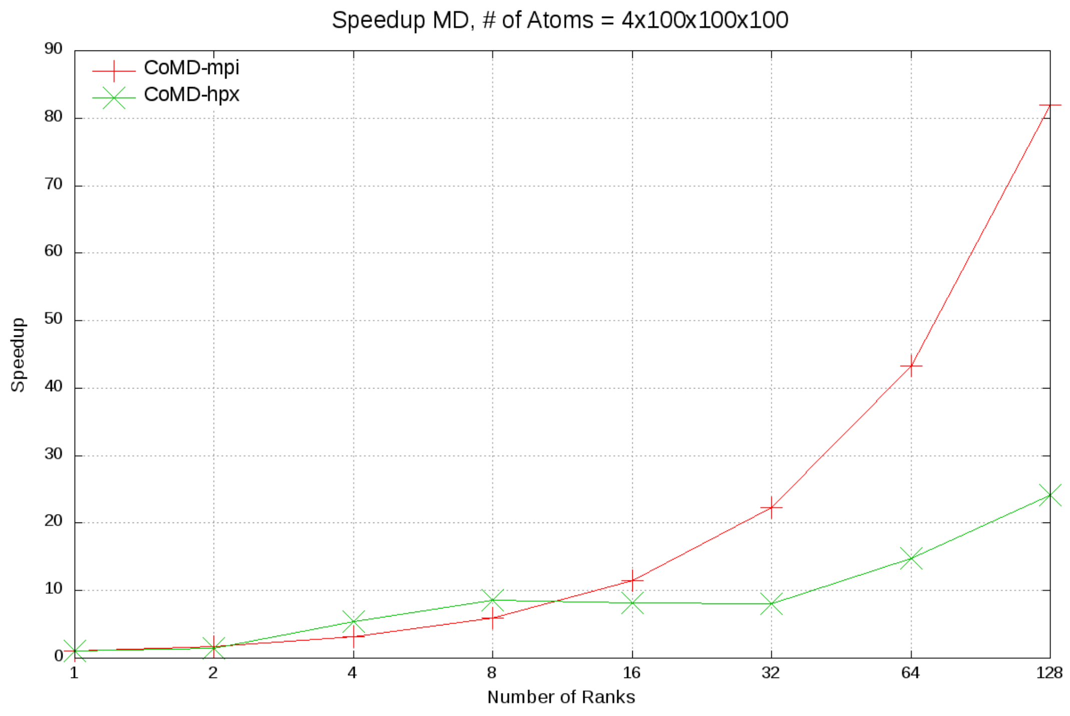

*Figure 4: Speedup*

## 4) Future Work

Now that a working HPX port for CoMD is in place the next step is to work on performance tuning. The near sighted path is to enable over-decomposition. Over-decomposition has the promise to hide latency and increase the overall system throughput. But in the long run it will be interesting to investigate how the core of CoMD-hpx be changed so as to deviate as much possible from Bulk Synchronous Model. This will involve introducing and exploiting asynchrony and parallelism.

**Reference:**

[1] http://www.exmatex.org/comd.html

[2] https://github.com/exmatex/CoMD