Enhanced Replica Exchange in NAMD Partition Framework in Charm++

Final report for NAMD NEIS-P2

Abstract

Replica exchange molecular dynamics is a useful method to study molecular systems for large energy ranges. Earlier, in an effort by J. Phillips, additions to Charm++'s MPI layer were made to enable replica exchange in NAMD. While that effort was successful, it resulted in suboptimal performance due to use of MPI layer instead of native layer. In this project, we have developed a generic partition framework in Charm++ that not only provides significantly better performance for replica exchange in NAMD, but is also useful in other research, e.g. replica based fault tolerance. In the process, we have added hooks for runtime system supported topology aware partitioning, and user-driven custom partitioning. The partitioning framework is part of Charm++'s stable release, and is being used in NAMD's development version. It has been tested on multiple supercomputers including Cray's XE6/XK7 and IBM's Blue Gene/Q.

1 Introduction

Replica exchange molecular dynamics is a method to effectively sample high-dimensional rough energy landscapes. In this method, a molecular system is simultaneously simulated with a range of initial temperature. Based on certain properties and derived probabilities, information is exchanged among the replicas. A simple method to simultaneously simulate these replicas is to submit different jobs for each simulation, and communicate via the file system. This approach is suboptimal given the slow communication via file system and dependence on job execution.

As part of the NAMD NEIS-P2 project, a partition framework in Charm++ has been developed. The framework enables simultaneous execution of multiple simulations within one user job. NAMD makes use of the partition framework to generate replicas of a given system, perform individual simulations, and exchange

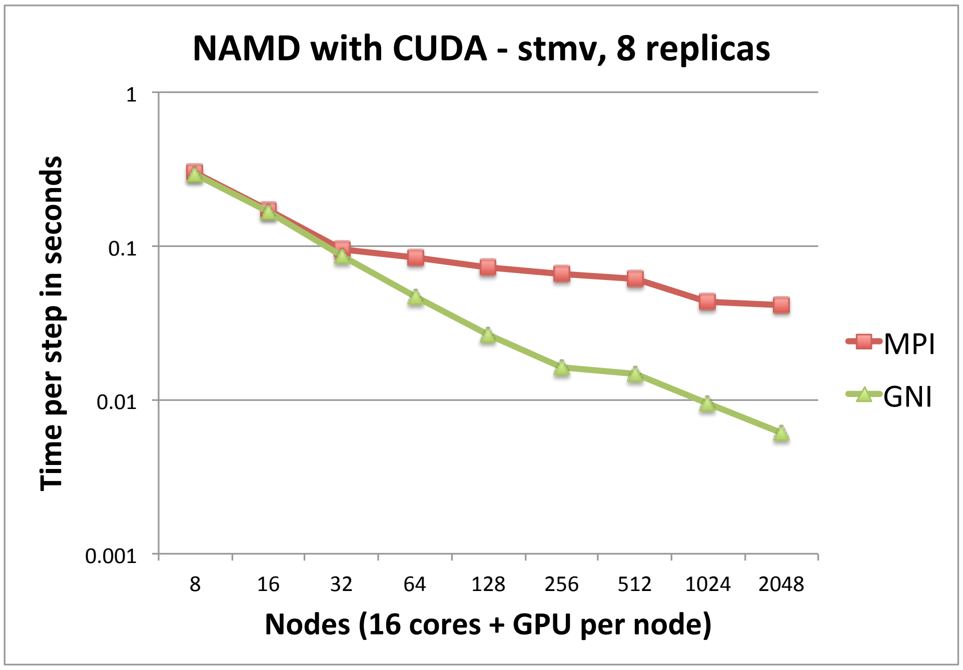

Figure 1: Performance comparison of replica in NAMD. Execution time is significantly better for GNI based replica runs enabled by the partitioning framework described in this report.

information among replicas. The following tasks were accomplished as part of this project:

- A generic framework that allows for running multiple Charm++ instances within one user job was developed.
- A new API was added for enabling inter-partition (across Charm++ instances) communication within one user job.
- NAMD was modified to make use of this framework for implementing replica exchange MD.
- Performance tuning was done to improve execution time for replica exchange MD.
- Support for topology aware partitions, variable sized partitions and user-driven partitioning was added to Charm++.

- The framework is available for use on any LRTS compliant layer of Charm++ including Cray's GNI, IBM's DCMF and PAMI, MPI and NET.

Figure 1 presents a comparison of simulation time for NAMD with CUDA using MPI layer and GNI layer. As the number of nodes in the system increase, we observe that performance gap between the two implementation increases. Currently, such performance gaps are typical of NAMD (and Charm++ in general) using MPI layer vis-a-vis a native machine layer [1, 2]. We are currently working on analyzing the impact of topology awareness on partitions creation.

The work done in this project is part of production version of NAMD and Charm++. The replica exchange method can be used in NAMD by downloading the nightly development version. More details on Charm++ partitioning framework can be found at `http://charm.cs.illinois.edu/manuals/html/charm++/26.html`.

2 Future Work

In the existing replica exchange method in NAMD, replicas independently perform a predefined number of simulation steps, and thereafter synchronize to exchange information. In future, we plan to add fine grained communication among replicas, which occur every step without the need for explicit synchronization.

From the framework's perspective, as mentioned earlier, work is in progress on analyzing impact of topology aware partitioning. Cray machines do not provide disjoint allocations, and hence finding a good partitioning is more difficult. We have found schemes based on space filling curves to be useful, and we continue to explore them. We also plan to work on adding further API to enable inter-partition communication at Charm++ level (instead of Converse, the portability layer under Charm++). Such an addition will increase programmer's productivity and simply code development.

Detailed report, with breakdown per quarter, follows in the following sections.

3 First Quarter

3.1 Problem Statement

First step was in requirements gathering, which produced the following:

- Implement support for distinct replica partitions within Charm++.
- Charm++ semantics for operations within a partition must be unchanged, which implies that existing code should work without any modifications.

- The implementation must be portable across machine layers, especially to the uGNI layer, unlike the MPI prototype.
- Communication across partitions must be supported.
- Cross partition communication semantics should be clearly differentiated from within partition communications.
- Partition size should be selectable at runtime.
- Typically, partition sizes will be homogeneous, but a use case exists where a single master partition, probably containing only a single node, would have a different size from the standard partitions.
- Support for heterogeneous sized partitions (other than single master), or dynamic sizing of partitions, is not supported by any application use case and is therefore not part of the design space for the solution.

3.2 Proposed Solution

Various implementation options were considered: within machine layers, at the Charm++ level, or within the Converse layer. The first option, modeled after the original MPI prototype, was rejected as being impractically unportable. The second was rejected for being difficult to implement without requiring code changes to existing codes. The most effective implementation was judged to be one which confines most (if not all) changes to be done at the Converse layer. See Figure 2 for overview of runtime system layers.

The primary impact of the implementation is that the processor set will be partitioned and a distinct Charm++ instance run within each partition.

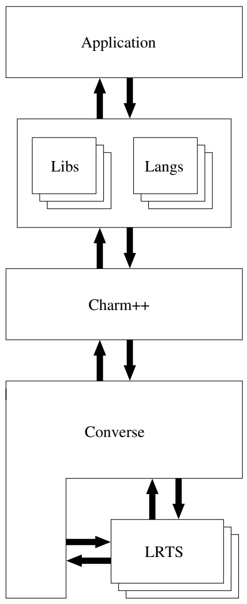

Figure 2: The CHARM++ system architecture.

3.2.1 Partitioning and Rank Translation

The primary effect of partitioning the process set is that it creates a new rank mapping scheme for each level of partitioning.

1. *Partition Creation* is called immediately after LrtsInit. This call (based on command line arguments or using some complex user defined scheme) partitions the processor space into Charm++ instancess. As a result, there are

two type of ranks for each processor - (a) local rank (out of CmiNumPes) which preserve their current meaning, and are local to a Charm++ instance (b) global rank which is the rank of PE as seen from the job's perspective.

2. *Rank Translator* is set of functions to translate to/from local rank from/to global rank

3. *Modifications for intra-instances traffic*: any send/recv to/from machine layer from/to Converse will use the rank translator to make sure that the corresponding layer gets the right ranks (local/global). The changes are local to RTS, and user are unaware of them.

4. *New Converse level send/recv functions for global communication*: communication based on partition id and local ranks.

In essence, all the Charm++ level routines and data structures (such as node groups, load balancing etc) work using the local ranks, and hence are limited to the local Charm++ instances. Any translation is handled by Converse layer, which also provides functionality for inter-instance communication. This format will lead to minimal changes without significant performance drop.

4 Second Quarter

4.1 Tasks Completed

The following changes were made to Charm++ to enable basic support for replica:

- Startup was modified to enable replicas based on command line arguments
- Local and Global ranking, and API associated with it were added
- Changes to Converse were made to correctly handle communication within a replica
- Converse level API added to enable inter-replica communication

4.2 Added API

- Local Rank - Existing function calls such as CkMyPe, CkMyNode, CmiMyPe, CmiMyNode now refer to local entities.
- Global Rank - Suffixed *Global* results in global ranks - CmiNumNodesGlobal, CmiMyNodeGlobal, CmiMyPeGlobal.

- Rank translation - Functions have been added to convert local ranks to global ranks - CmiGetNodeGlobal, CmiGetPeGlobal. These functions should ideally be used only by the runtime system. Functions for converting global ranks to local ranks are omitted because they should not be ever used, and may have severe performance impact.
- Partition - Calls to get information about partitions - CmiMyPartition, CmiPartitionSize, CmiNumPartition.
- Inter-replica Communication - Functions corresponding to basic Converse send functions were added - CmiRemoteSyncSend(local_rank, partition, size, message), CmiRemoteSyncSendAndFree(local_rank, partition, size, message).

.

5 Third Quarter

5.1 Tasks Completed

The following tasks were completed during this quarter:

- Output to standard output (stdout) from various partitions was redirected to separate files by passing the target path as a command line option. The run time parameter `+stdout <path>` is to be used for this purpose.
- Support for replication was made part of the production code of Charm++, and was included in Charm++ 6.5 release.
- Modifications were made to NAMD to use the partitioning scheme implemented in Charm++.
- NAMD with replica was tested on all LRTS compliant platforms MPI (all machines), DCMF (BG/P), UGNI (XE6) and PAMI (BG/Q).

5.2 Changes to NAMD

Major changes were made to NAMD in two places to make use of the new partitioning implementation in Charm++:

- DataExchanger: A new chare group called DataExchanger was added to handle inter-partition communication. This group contains functions and entry methods required to perform all the communication initiated by ScriptTcl.

- ScriptTcl: Implementation of TCL commands have been rewritten to use DataExchanger for communication. Owing to changes in the implementation of TCL commands only, any TCL script that used to work earlier (with the MPI patch) will continue to work without changes.

6 Final Quarter

Having integrated replica into NAMD and Charm++, bug fix and optimizations were the principal focus for the last quarter. The following options were added:

- Runtime parameter: `+partitions <part_number>` or `+replicas <replica_number>` : number of partitions to be created. If no further options are provided, allocated cores/nodes are divided equally among partitions. Only this option is supported from the 6.5.0 release of Charm++; remaining options described below are supported starting 6.6.0 release of Charm++.

- Runtime parameter: `+master_partition` : assign one core/node as the master partition (partition 0), and divide the remaining cores/nodes equally among remaining partitions.

- Runtime parameter: `+partition_sizes L[-U[:S[.R]]]#W[,...]` : defines the size of partitions. A single number identifies a particular partition. Two numbers separated by a dash identify an inclusive range (*lower bound* and *upper bound*). If they are followed by a colon and another number (a *stride*), that range will be stepped through in increments of the additional number. Within each stride, a dot followed by a *run* will indicate how many partitions to use from that starting point. Finally, a compulsory number sign (#) followed by a *width* defines the size of each of the partitions identified so far. For example, the sequence `0-4:2#10,1#5,3#15` states that partitions 0, 2, 4 should be of size 10, partition 1 of size 5 and partition 3 of size 15. In SMP mode of Charm++ which allows for separate communication thread shared by worker threads, these sizes are in terms of nodes. All workers threads associated with a node are assigned to the partition of the node. This option conflicts with `+master_partition`.

- Runtime parameter: `+partition_topology` : use a default topology aware scheme to partition the allocated nodes.

- Runtime parameter: `+partition_topology_scheme <scheme>` : use the given scheme to partition the allocated nodes. Currently, two generalized schemes are supported that should be useful on torus networks. If scheme is set to 1, allocated nodes are traversed plane by plane during partitioning. A hilbert curve based traversal is used with scheme 2.

- Compilation parameter: `-custom-part`, runtime parameter: `+use_custom_partition` : enables use of user defined partitioning. In order to implement a new partitioning scheme, a user must link an object exporting a C function with following prototype:

  extern “C” void createCustomPartitions(int numparts, int *partitionSize, int *nodeMap);

  - `numparts` (input) - number of partitions to be created.
  - `partitionSize` (input) - an array that contains size of each partition.
  - `nodeMap` (output, preallocated) - a preallocated array of length `CmiNumNodesGlobal()`. Entry *i* in this array specifies the new global node rank of a node with default node rank *i*. The entries in this array are block wise divided to create partitions, i.e entries 0 to partitionSize[0]-1 belong to partition 1, partitionSize[0] to partitionSize[0] + partitionSize[1] - 1 to partition 2 and so on.

  When this function is invoked to create partitions, TopoManager is configured to view all the allocated node as one partition. Partition based API for TopoManager and Charm++ is yet to be initialized, and should not be used. A link time parameter `-custom-part` is required to be passed to `charmc` for successful compilation.

References

[1] Yanhua Sun, Gengbin Zheng, Chao Mei Eric J. Bohm, Terry Jones, Laxmikant V. Kalé, and James C.Phillips. Optimizing fine-grained communication in a biomolecular simulation application on cray xk6. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, Salt Lake City, Utah, November 2012.

[2] Yanhua Sun, Gengbin Zheng, L. V. Kale, Terry R. Jones, and Ryan Olson. A uGNI-based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect. In *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Shanghai, China, May 2012.