



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Project Report: CREST (Center for Research and Extreme Scale Technologies)

U. Wickramasinghe

May 7, 2015

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.



Project Report

by Udayanga Wickramasinghe,

CREST (Center for Research and Extreme Scale Technologies)

Table of Contents

HYBRID MPI (HMPI)	3
OPENMP (THREAD MODEL) BASED SYNERGISTIC TRANSFER	3
AFFINITY CONTROL	3
RESULTS	3
SIGHT	4
DISTRIBUTED ON THE FLY LOG MERGING	5
NEW API'S AND OPTIMIZATIONS	5
BUG FIXES AND IMPROVEMENTS TO BUILD SYSTEM	6
FLOW	6
DATA SERIALIZATION/DE-SERIALIZATION	6
MRNET INTEGRATION	7
CBTF	7
STREAMING QUERIES	7

Hybrid MPI (HMPI)

Many core architectures like the Xeon Phi are becoming popular because of their extremely high computing capacity for a lower power consumption, support for the x86 instruction set and the ability to work as an accelerator for conventional processor. Previous results on Xeon Phi Hardware showed that HybridMPI is an efficient MPI model to leverage performance benefits of such many core shared memory architectures. XeonPhi is perfectly placed as an HMPI use case because of its high computational capacity and more importantly of the availability of hardware intrinsic like high bandwidth memory bus and a highly efficient cache memory and coherence protocol to out perform per-core bandwidth/throughput of any of other platforms available today. In order to better utilize this hardware , 2 additional prototype features were implemented for Hybrid MPI and initial results were observed .

OpenMP (Thread model) Based Synergistic Transfer

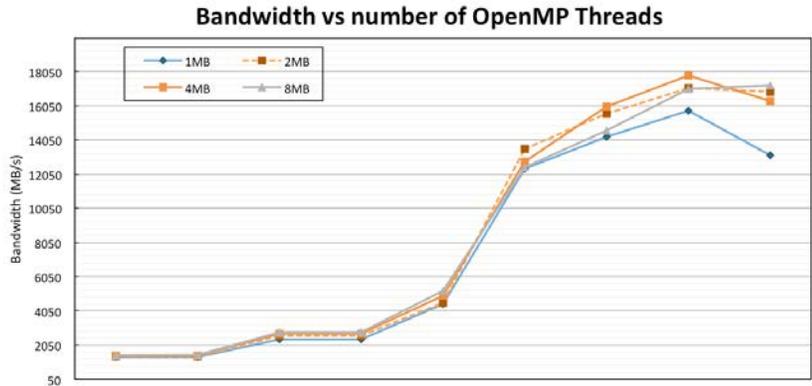
Synergistic transfer makes use of the fact that message bandwidth can be increased by involving both sender and receiver in the communication. Since the Xeon Phi has 60 cores, communication bandwidth can be further 'synergized' by contributing as many cores possible to either/both sender/receiver engaged in transferring data. We have used OpenMP as a technology to drive the message pipeline using multiple cores on Xeon Phi. Furthermore native techniques such as non temporal store instructions (VMOVNRNGO/VMOVNRN) were utilized to further increase the bandwidth.

Affinity Control

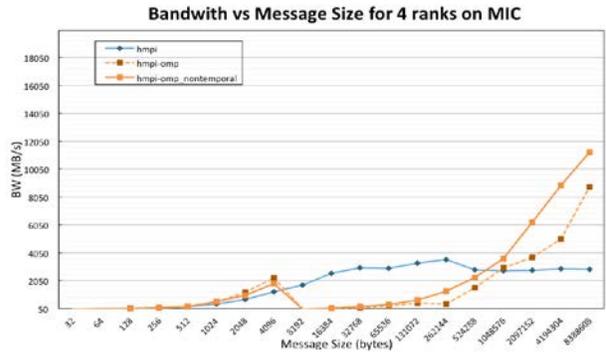
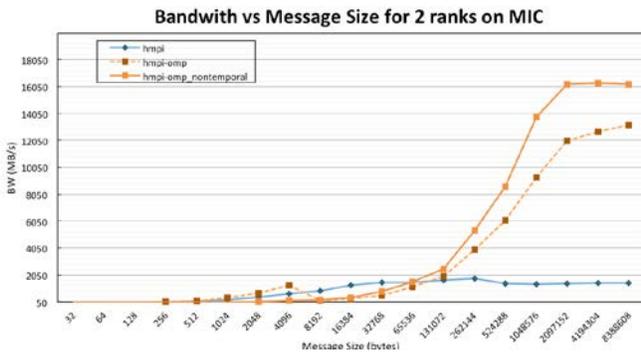
Initial results with OpenMP synergistic transfer suggested that overhead due to thread and processor contention can be a major factor towards performance of the system. Therefore the communication model was coupled with light weight profiler that takes into account thread assignment into cores. Couple of different techniques were experimented, including thread affinity control based on thread pinning to cores , and adaptive techniques based on available resources.

Results

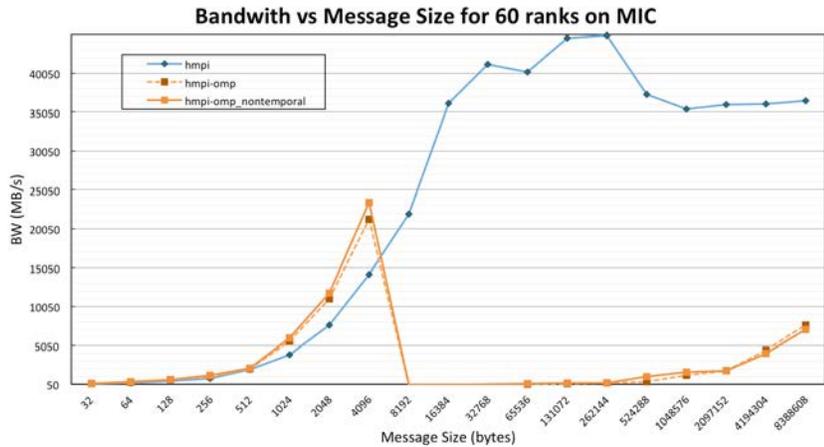
Several benchmark experiments (LLNL Presta benchmark) were conducted on TACC STAMPEDE super computing cluster to test prototype performance. Results were very encouraging for lesser number of ranks. More specifically OpenMP based message transfer allowed 16GB/s peak bandwidth with just 2 ranks and speedup in the range of 2X to 15X with lower number of ranks running. Higer percentage of improvement was visible on large message sizes. However Performance degradation was visible when we increased the ranks and hence bottleneck in resource contention for memory bus and microprocessing units.



¹ Figure 1 : Strong Scaling with prototype by increasing number of Threads/Cores



Fig



ber of ranks

Figure 3 : Performance degrades rapidly for HMPI-OMP prototype when number of ranks >= 30

Sight

'Sight' is a log analysis tool developed by Greg Bronevetsky. 'Sight' helps analyze scientific and domain specific applications by instrumenting them to produce logs that can infer useful information to end user in terms of tracing, formatting/structuring, graphs, performance data (ie:- papi,etc), etc.

Distributed on the Fly Log Merging

'Sight' was initially designed to work standalone and perform log analysis offline. However we were able to extend it to a scalable and on the fly log analysis framework for 'Sight' using MRNet framework. 'Sight' uses MRNet infrastructure which is a communication overlay based on a hierarchical reduction network. The newly introduced framework works based on the following principles.

1. Nodes working as Applications for 'Sight' will produce logs on the file system. Each application will need to be instrumented with 'Sight' API either manually or automated way.
2. A MRNet front end process will need be initiated at some physical node and will be listening to any 'Sight' specific node's that will be attached once the application processes are initiated at the respective nodes.
3. MRNet Communication layer on 'Sight' will intercept logs produced by application and these will be sent upstream as MRNet packets.
4. MRNet filter nodes sitting in between front node and application nodes will intercept streaming log data, parse and finally merge (greedy optimistic merge) them appropriately as and when useful information is available on them.
5. Merged logs are finally gathered on MRNet front end process.

Current working prototype has a single Merge level tree implemented and multiple hierarchical merging will be implemented soon. We have created separate branch on 'Sight' repository (<https://github.com/bronevet/sight/tree/mrnetDevelop>) for this and examples and documentation were included.

New API's and Optimizations

'Sight' has been used in many applications and one of the requirements was to use 'Sight' in a communication runtime called HPX-5 developed by IU. HPX is a implementation of ParallelX which is based on PGAS model of communication. Since HPX runtime is based on C , we have created an C based API extension that can be used by HPX like systems. Further objectives include optimize 'Sight' to further support a light weight kernel that can be directly plugged into high throughput runtime systems such as HPX.

Bug Fixes and Improvements to Build System

Various kinds of improvements were made to support threads and graceful cleanup methods into 'Sight'. Current MAKE based build system was improved to support an efficient build platform for 'Sight'. Future goals include migrating 'Sight' to CMAKE build system.

Flow

'Flow' is an implementation of a continuous or streaming query engine for key/value based data. Main purpose is to perform highly efficient distributed querying and processing of large graphs, etc data structures for HPC applications and hence achieve scalable performance over a significant set of nodes. We use MRNet as the communication overlay for streaming the binary structures/data on the workflow engine. Basic component in 'Flow' is called an 'operator' which multiple of them can be pipelined to perform some meaningful task. Each operator is fed with some input and process the data according to a certain schema and produce some output, this is further elaborated in the diagram below.

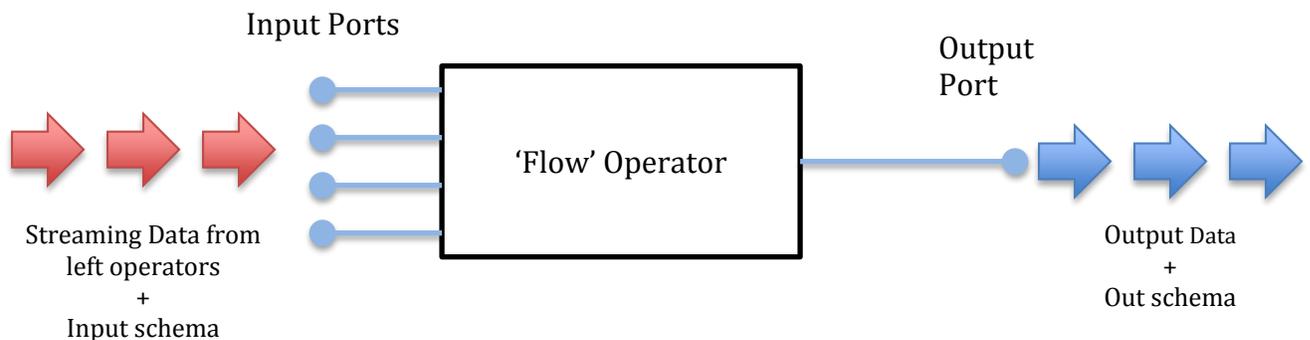


Figure 4 Basic Architecture of Flow Operators

Data Serialization/de-Serialization

In order to stream data efficiently, a binary packing format has been introduced by 'Flow'. Therefore each operator has serialization and deserialization layer (platform specific) built into it, which will perform packing and unpacking of data according to a certain schema. Currently a file based and a memory buffer based serialization mechanisms are built into 'Flow' engine.

MRNet Integration

MRNet interconnects operators on heterogeneous and/or homogeneous set of nodes in a 'Flow' network. There are few different MRNet specific streaming operators which is listed below.

MRNet Frontend Source Operator → responsible for creating MRNet Frontend node and initializing communication. It will receive the aggregated stream packets from child nodes.

MRNet Backend Sink Operator → responsible for sending data as stream packets from different data sources ie:- files/database systems.

MRNet Filter Source → responsible for connecting and processing upstream data. Filter Source will take packets from downstream, de-serialize them and perform some aggregation operation. 'Flow' configuration will typically be a Filter source → scatter/gather operator/s → Filter sink

MRNet Filter Sink → responsible for serializing the aggregated binary data to upstream components.

CBTF

Some discussion took place to integrate Component Based Tool Framework (CBTF) to 'Flow' engine. CBTF is a tool development framework, which has the ability to combine compatible reusable components into existing tools or frameworks by providing interface API's to connect different components seamlessly. Furthermore MRNet based communication model is built into CBTF which is an added advantage. Integration of CBTF with 'Flow' will be a future milestone.

Streaming Queries

This feature haven't yet been built into 'Flow', but would be an immediate milestone. Our idea is to build SQL like declarative syntax based query processor and plan generator for streaming data on top of 'Flow'.