Methodology for Adaptive Active Message Coalescing in Task Based Runtime Systems

Bibek Wagle, Samuel Keller, Adrian Serio and Hartmut Kaiser
Louisiana State University

May 25, 2018

iWAPT 2018

LSU | Center for Computation & Technology

STE||AR GROUP

Background

- Fine grained tasks results in fine grained communication pattern
- Efficient communication
    - Latency
    - Bandwidth
    - Overheads associated with creating and sending messages
- What can be done ?
    - Reduction of overheads

Message Coalescing to Reduce Overheads

- Message coalescing is a technique that is useful for reducing overheads
- Combine small messages into large ones
- Effectively send the same amount of data while reducing per message overheads

Approaches to Coalescing

- Manual coalescing
    - High effort
    - Impractical for larger projects
- Runtime system provided coalescing
    - HPX, AM++ / Active Pebbles, Charm++
- Issues:
    - How many messages to coalesce?
    - Do different applications need different parameters?
    - How do you determine the coalescing parameters?
- Solution: Intelligent Adaptive coalescing approach that dynamically varies its parameters depending upon application behavior.

Existing Solutions

- Charm++ exhibits basic adaptive approach for message coalescing
- Application is run automatically with different coalescing parameters each iteration
- Possible improvements:
  - Allow for varying coalescing parameters mid iteration based on the phase of the application
  - General adaptive framework that does not require iterative steps or predictable pattern of communication.

Towards Advanced Message Coalescing

- For Advanced general adaptive coalescing framework :
    - Implement message coalescing in HPX
    - Identification of metrics and runtime characteristics pertaining to fine grained communication overheads
    - Utilization of identified metrics and runtime characteristics for adaptive tuning of coalescing parameters.

The HPX Runtime System

- Asynchronous Task based distributed runtime system written mostly in C++
- HPX application can run on both a single machine as well as a cluster with thousands of nodes
- Exposes a concurrency and parallelism API consistent with the ISO C++ standard
- Real time performance measurement capabilities
- Runtime adaptive capabilities

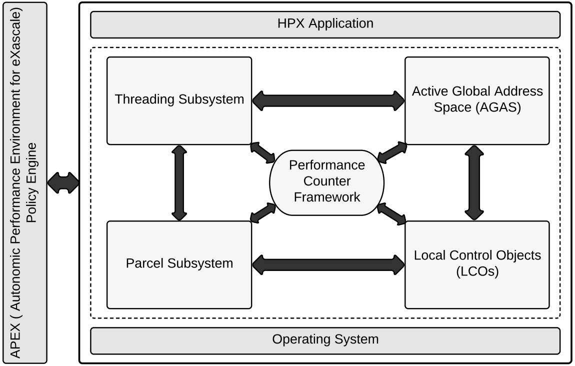

Figure 1: Architecture of HPX

The HPX architecture consisting of AGAS for addressing any HPX object globally , LCOs for synchronization of tasks , Threading Subsystem for employing lightweight tasks on OS threads , Parcel Subsystem for executing tasks remotely, Performance counter framework for instrumentation and debugging purpose and APEX for runtime adaptive capabilities.

The HPX Parcel

- A form of Active message.
- Created when a method, called action in HPX terminology is called remotely.
- Goes though serialization process which converts it into stream of bytes and is sent over the wire
- HPX presently supports : TCP/IP, MPI and IB-Verbs protocols for remote sends
- Reconstructed at the receiving end and placed in scheduler queue for execution.

Destination Address	Action	Arguments	Continuations

Figure 2: Structure of a HPX Parcel. A parcel has four components: the destination address; the action, which is the method/function to execute at the destination; the arguments for the function; and optional continuations.

Parcel Coalescing

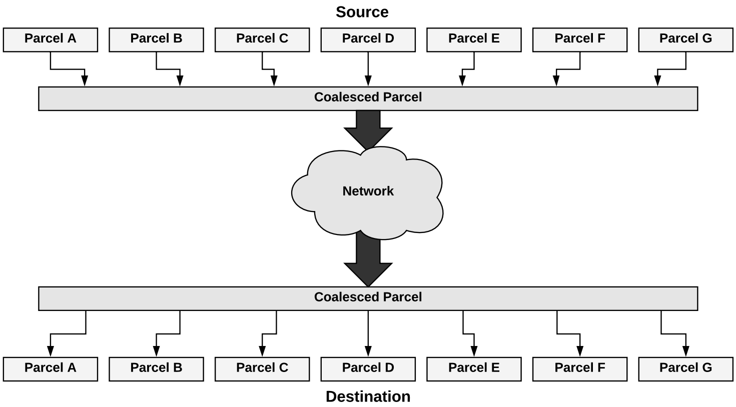

Figure 3: A diagrammatic representation of message coalescing. Individual active messages are grouped together to form a large message at the sending end which is reconstructed into the original individual entities at the receiving end.

Parcel Coalescing in HPX

- Designed around two parameters
    - Queue length : the number of parcels to coalesce in a single send
    - Wait time : time to wait in microseconds for the queue to be full before flushing the queue
- Wait time helps avoid deadlocks

Network Performance Metrics

- Metrics for measuring network overhead are necessary for achieving our goal of adaptive message coalescing.
- We define overhead as the time spent processing information to be communicated across the network.

Task Duration

We looked at the overall time spent on executing each HPX-thread or tasks including the overhead. We define task duration using the following equation:

$$t_d = \sum t_{func} \tag{1}$$

where $\sum t_{func}$ is the total time spent by the HPX scheduler executing each HPX thread.

Task Overhead

We then looked at the average time spent on thread management for each HPX-thread or tasks. We calculate task overhead using the following equation:

$$t_o = \frac{\sum t_{func} - \sum t_{exec}}{n_t} \tag{2}$$

where $\sum t_{exec}$ is the time spend by the HPX scheduler doing useful work and $\sum t_{func}$ is the task duration as defined in Equation 1 and n_t is the number of executed HPX threads.

Observations

- We observed a positive correlation between task overhead and overall execution time of our test applications for various coalescing parameters.
- After establishing that task overhead has a positive correlation with the overall execution time, we separated the network related overhead from other overheads.
- HPX performs network related tasks such as packaging a parcel into a message, serialization, handshaking and locality resolution in the form of background work.

Background Work Duration

We define total time spent doing background work as the background work duration and it is obtained using the following equation:

$$t_{bd} = \sum t_{background-work} \tag{3}$$

Network Overhead

The network overhead count is the ratio of thread background work duration to task duration. Network Overhead is shown in Eq. 4.

$$n_{oh} = \frac{\sum t_{background-work}}{\sum t_{func}} \tag{4}$$

Here, $\sum t_{background-work}$ is the total time spent performing network related work and $\sum t_{func}$ is the total time to reach the completion of each HPX thread.

Hence, Network Overhead gives us the fraction of overall time spent on performing network related work.

Testing the Network Overhead metric

- We test for correlation between our Network Overhead metric and overall execution time of our test applications.
- Experimental Testbed
    - Marvin Thin Compute Nodes of ROSTAM Cluster
    - 2x Intel Xeon E5-2450 CPU 16 Cores total
    - 48GB 1333 MHZ DDR3 Memory
    - HPX v 1.0 , GCC 6.3 , IMPI 2017.2.174

Testing the Network Overhead metric

- We use two test applications:
    - A Toy Example
    - The Parquet Application

Toy Application Example

```
//Create Action
complex<double> get_cplx()
{
return complex<double>(13.3,-23.8);
}

HPX_PLAIN_ACTION(get_cplx,actn);
HPX_ACTION_USES_MESSAGE_COALESCING(actn);  ❶

//Create instance of the actions
actn act;

vector<hpx::future<complex<double>>> vec;
vec.reserve(numparcels);

//Find the other locality
auto localities=hpx::find_remote_localities();
auto other=localities[0];

int num_repeats=4;
//Repeat num_repeats times
for (int j = 0; j < num_repeats; j++)
{
for (int i = 0; i < numparcels; ++i)
{
vec.push_back(hpx::async(act, other));      ❷
}
//Wait for all the tasks to complete
hpx::wait_all(vec);
}
```

Figure 4: Toy Application

Definition

Experiments on the Toy application was performed on two Nodes where each node sent a million messages to each other.
We define the process of sending a million message as a phase as shown in annotation 2 in Figure 4.

Toy Application Execution Time per Phase

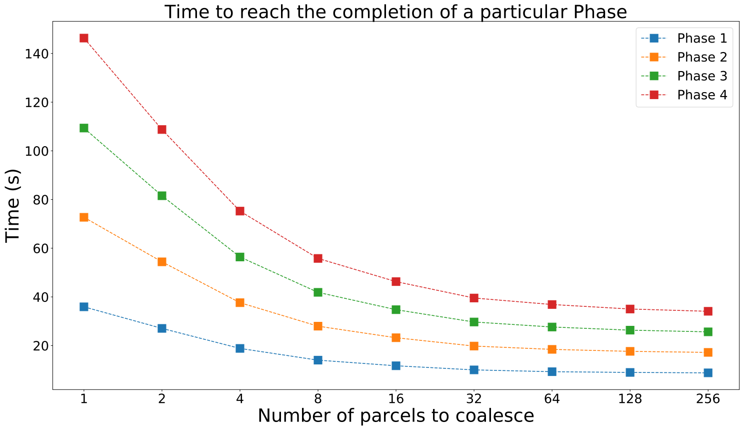

Figure 5: Time to reach the completion of a particular phase in the toy application for various values of number of parcels to coalesce in a single message with a wait time of 4000μs.

Toy Application Network Overhead vs Time Per Phase

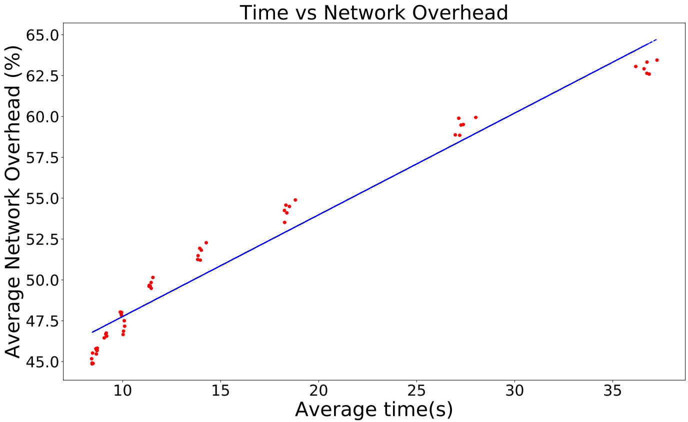

Figure 6: Scatter Plot of the average network overhead per phase vs average execution time per phase for the toy application. Each dot represents a set of parcel coalescing parameters. Average overhead is the average for four phases. A Pearson's correlation coefficient of 0.97 indicates a strong positive correlation between network overhead and runtime.

Observations

- Fastest time per iteration seen with largest value of number of parcels to coalesce.
  - Lack of dependency with any other communication or computation
- Does not reflect the behavior of a real application.

The Parquet Application

- A complex physics simulation
- Requires use of many rank-three tensors
- The linear dimension (N_c) of the simulation controls the tensor size
- Throughout the simulation, large number of messages are sent between nodes
- The rotation phase sends $8 * N_c^2$ parcels containing N_c elements

Parquet Time to reach a particular iteration

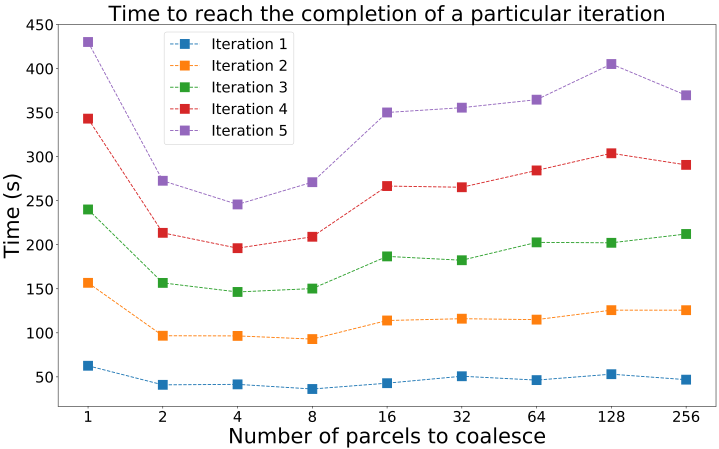

Figure 7: Time to reach the completion of different iterations in the parquet application for various numbers of parcels coalesced in a single message with a wait time of $4000\mu s$. Each color indicates a different iteration.

Parquet Execution Time vs Network Overhead

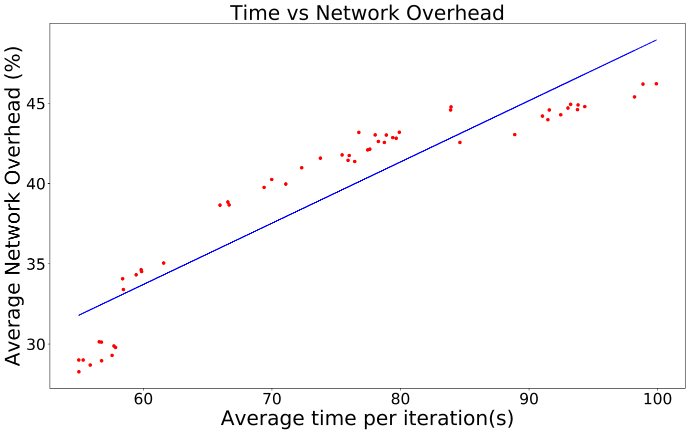

Figure 8: Scatter Plot of Average Network Overhead Vs Average time per iteration for the Parquet Application. Each dot represents a set of parcel coalescing parameters. A Pearson's correlation coefficient of 0.92 was calculated indicating a strong positive correlation.

Parquet Execution Time

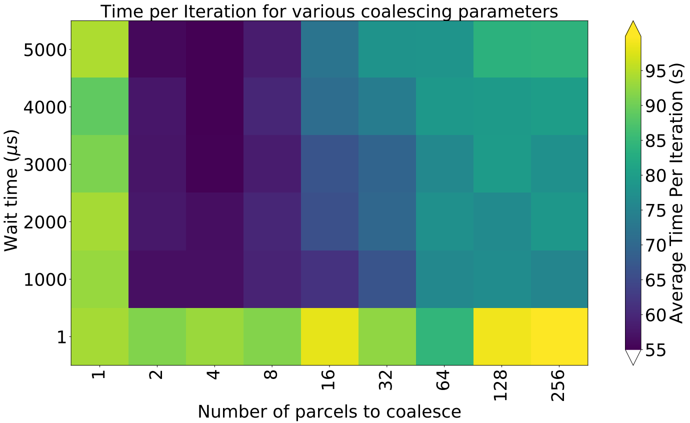

Figure 9: Average time per iteration for various coalescing parameters.

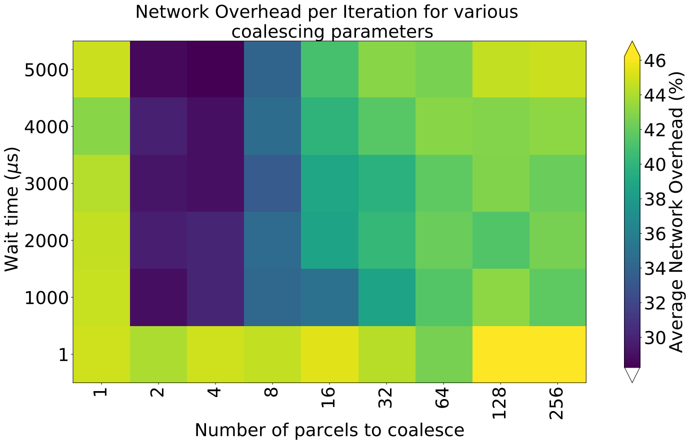

Figure 10: Average Network Overhead per iteration for various coalescing parameters.

Side by Side for Visual Correlation

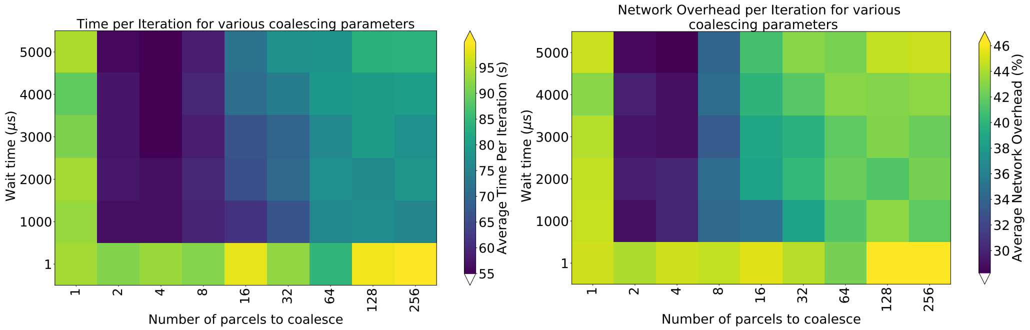

Figure 11: Average time per iteration and average Network Overhead per iteration for various coalescing parameters.

Change Coalescing Parameters each Phase

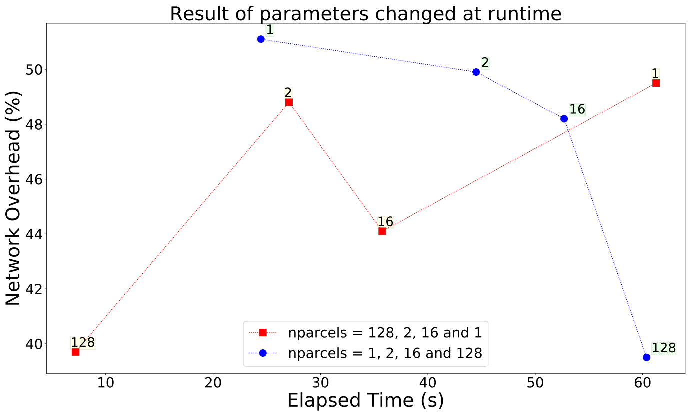

Figure 12: Network overhead for various values of number of parcels to coalesce in a single message each phase with a wait time of 2000μs for two different runs of the toy application.

Conclusions

- Sub-optimal parameter selection results in drastic performance loss.
- Non-availability of methods other than brute force to “guess” coalescing parameters signals need for adaptive methods.
- Metrics identified in this research showed strong positive correlation with execution time for two different applications.
- Initial results hints towards the possibility of being able to use our metrics for adaptive tuning of parcel coalescing.

Support

This work was partly funded by the NSF EPSCoR LA-SiGMA project under award #EPS-1003897 ,the NSF STORM project under the award #ACI-1339782 and NSF Phylanx project award #1737785. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Questions?