Adaptive auto-tuning in HPX using APEX

Mohammad Alaul Haque Monil[1], Bibek Wagle[2], Kevin Huck[3], Hartmut Kaiser[2]

[1]Department of Computer and Information Science , [3]Performance Research Laboratory, University of Oregon, Eugene, Oregon.

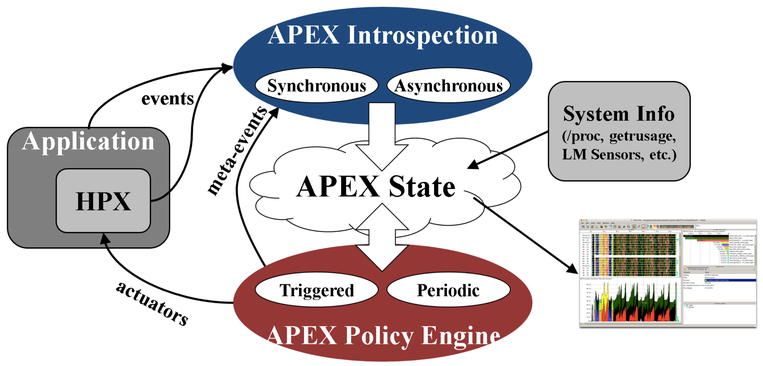
[2]School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, Louisiana.

Abstract

Finding an optimal value for a parameter that impacts application performance is a hard problem and often requires repetitive execution and hence incurs wastage of resources. In this research, we provide a preliminary study which demonstrates parameter value searching at runtime for better performance. We use APEX performance measurement library to implement adaptive auto-tuning policy to tune parcel coalescing parameters based on a sampled counter of HPX runtime system.

APEX

- APEX is a performance measurement library for distributed, asynchronous tasking models/runtimes. i.e. HPX, but there are others. [1]
- It provides lightweight measurement (tasks <1ms), high concurrency and distinction between OS and runtime (HPX) thread context.
- It works based on task dependency chain and has infrastructure for dynamic feedback and control of both the runtime and the application



APEX Introspection and Event Listener:

- APEX collects data through "inspectors": Synchronous uses an event API and event "listeners". Asynchronous do not rely on events, but occur periodically based on Sampled values (counters from HPX).
- Profiling listener: Capture parent task relationship, Start, Stop event, etc.
- TAU and OTF2 Listener (postmortem analysis): Synchronously passes all measurement events to TAU and libotf2 to build an offline profile/trace analysis.

APEX Policy Engine :

- Policies are rules that decide on outcomes based on observed state.
- Triggered policies are invoked by introspection API events and periodic policies are run periodically on asynchronous thread.
- All Policies are registered with the Policy Engine with a callback function. Callback functions define the policy rules.
- Enables runtime adaptation using introspection data through feedback and control mechanism and engages actuators across stack layers.
- It integrates Active Harmony [4] for searching and auto-tuning.

Parcel Coalescing in HPX

HPX is a C++ runtime system based on the ParalleX model[3]. The HPX threading system employs lightweight tasks, known as HPX threads, that are scheduled on top of operating system threads. The Active Global Address Space (AGAS) system in HPX provides a mechanism for addressing any HPX object globally.

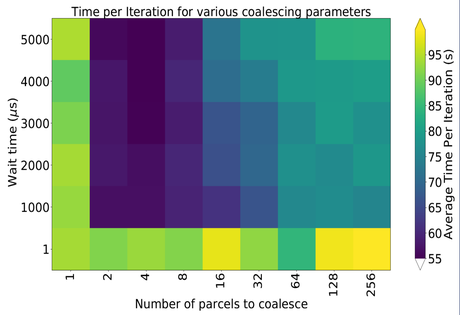
- Lightweight tasks in HPX produce fine grained communication. Parcel coalescing technique (Algorithms 1.) is used in HPX to reduce overhead.
- The number of parcel and coalescing interval have huge impact on performance.

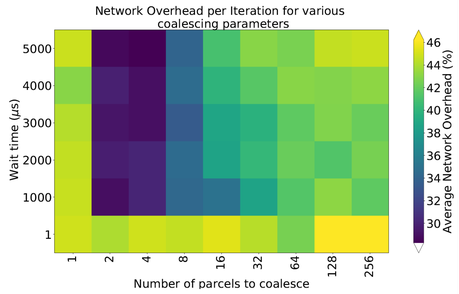
Algorithm 1 Parcel Coalescing

```
procedure COALESCING MESSAGE HANDLER
    nparcels ← number of parcels to coalesce in a message
    interval ← wait time in microseconds
    s ← state of arriving parcel
    tslp ← time since last parcel
    if tslp > interval then
        send parcel
    switch s do
        case First :
            Start Flush timer
            Queue Parcel
        case !First||Last :
            Queue Parcel
        case Last(QueueFull) :
            Stop Flush timer
            Flush queued parcels
```

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

- Recent research [2] demonstrated a positive correlation between task overhead (Network Overhead n_{oh}) and overall execution time.
- Parquet, a complex physics simulation is tested for different wait time and number of messages to coalesce.
- The first graph represents heat map of execution time and the second one represents average network overhead.
- Two graphs show similar heat maps which show the correlation between execution time and network overhead.
- The application was run many times to find out this result.

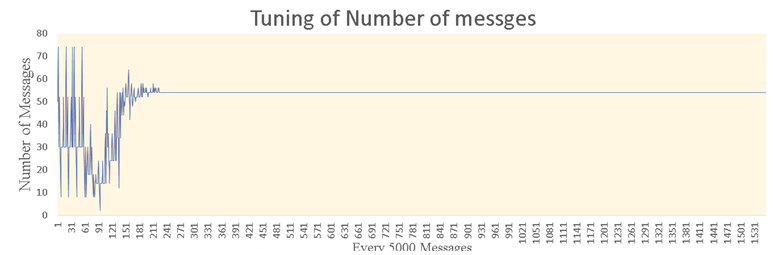


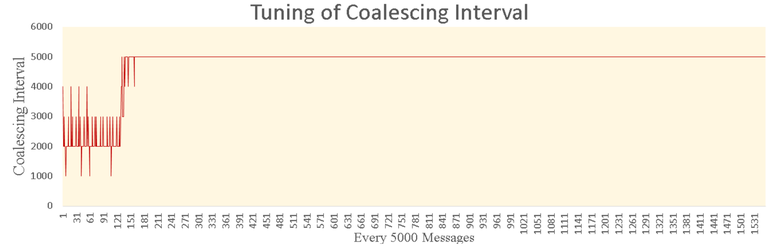


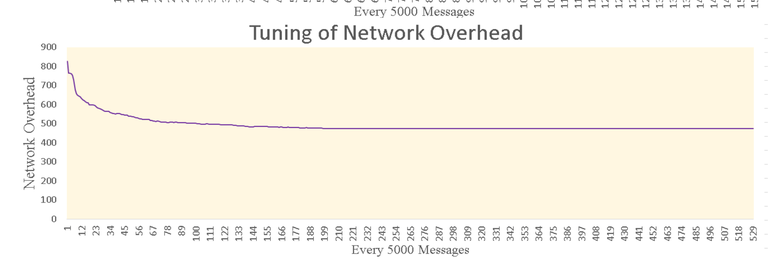
- This finding brings the opportunity for adaptive APEX policy where APEX policy will find the suitable values for wait time and number message to coalesce during the application runtime.

Adaptive Parcel Coalescing Policy in APEX

- To avoid repetitive execution to search for the best parameter values we defined a Parcel Coalescing Policy with the option to trigger the policy periodically or based on an event, for example: every 5000 messages. It can start with a default/random/user_provided starting values for the interval and the number of messages to coalesce.
- The callback function for the policy is a call to Active harmony with the APEX sampled counter value of network overhead of HPX and the current value of interval and number of messages to coalesce. Active harmony observes the counter value to change the value of the two parameters.
- Below figures represent the impact of the policy on a toy application [2] where policy is triggered every 5000 message send events between two nodes.
- It shows that It convergence of the two parameters while reducing the network overhead.







Future Work

We plan to test this policy for a couple of real application in large scale. Moreover, we wan to find out an adaptive approach to trigger this policy based on application characteristics.

Reference

1. K. A. Huck, A. Porterfield, N. Chaimov, H. Kaiser, A. D. Malony, T. Sterling, and R. Fowler. An autonomic performance environment for exascale. *Supercomputing frontiers and innovations*, *2*(3), pp.49-66, 2015
2. B. Wagle, S. Kellar, A. Serio and H. Kaiser. Active Message Coalescing in Task Based Runtime Systems. International Workshop on Automatic Performance Tuning. IWAPT-2018. (accepted)
3. H. Kaiser, M. Brodowicz, and T. Sterling, "Parallex an advanced parallel execution model for scaling-impaired applications," in Proceedings of the 2009 Intl. Conference on Parallel Processing Workshops, ICPPW '09. Washington, DC, USA, 2009, pp. 394–401.
4. C. Tapus, I. Chung, and J. K. Hollingsworth. Active harmony: Towards automated performance tuning. In 2002 ACM/IEEE Conference on Supercomputing, SC '02, pages 1–11, Los Alamitos, CA, USA, 2002.