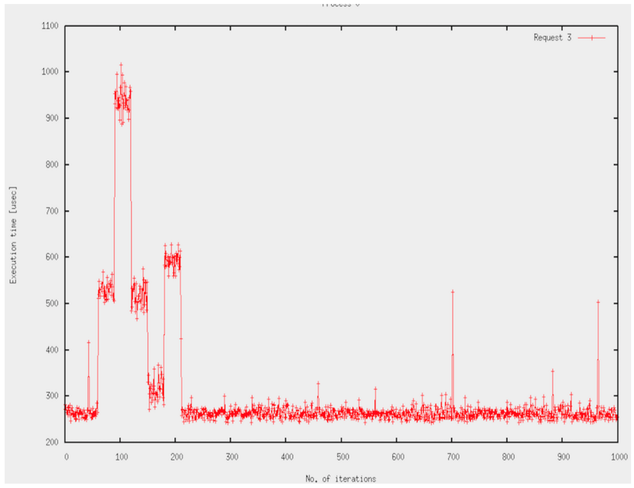
Optimizing Non-blocking Collective Communication Operations

Youcef Barigou, Vishwanath Venkatesan, and Edgar Gabriel
Parallel Software Technologies Laboratory, Department of Computer Science,
University of Houston, Houston, TX

Abstract

The Abstract Data and Communication Library (ADCL) is a parallel application level communication library aiming at

- Provide abstract interfaces for often occurring communication patterns
- Provide a wide variety of implementations for each communication pattern
- Provide algorithms to choose the fastest available implementation at runtime



In order to minimize the overhead introduced by the selection logic, the library has the capability of learning from the previous explored problems.

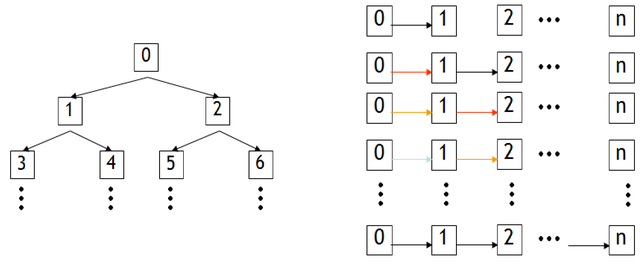
High Level API

```
ADCL_Request req;
ADCL_Timer timer;

// Initialize blocking persistent collective operation
ADCL_Alltoall_init ( sbuf, scount, sdat, rbuf, rcount, rdat, comm, &req);

// Associate handle with a timer object
ADCL_Timer_create ( req, &timer);

//Main application loop
for (i=0; i<MAXIT; i++ ) {
   ADCL_Timer_start (timer);
   ...
   // Start communication operation
   ADCL_Request_start (req);
   ...
   ADCL_Timer_end (timer);
}
```

- Supported for most collective operations
- Extends the concept of persistent MPI point-to-point operations
- Can be referenced by an ADCL_Request

Collective Operations

- Binary Tree (left) vs. Chain (right) broadcast algorithms

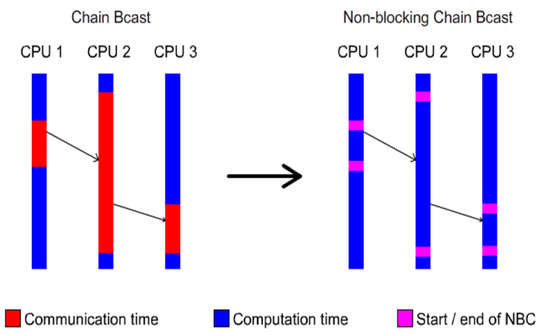


Network saturated starting from a certain number of messages

Performs very well for very large messages

- Offer a higher level abstraction for often occurring communication patterns
- Separate desired data movement from actual implementation
- Allow for numerous optimizations internally

Non-blocking Collective Operations

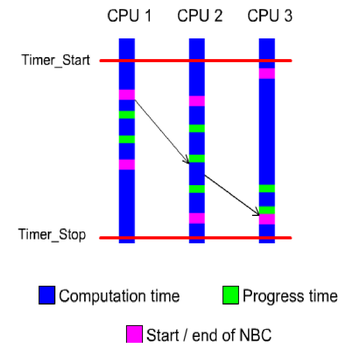


- Maintain advantages of blocking collective operations
- Overlap communication and computation

Auto-tuning Non-blocking Collective Communication Operations

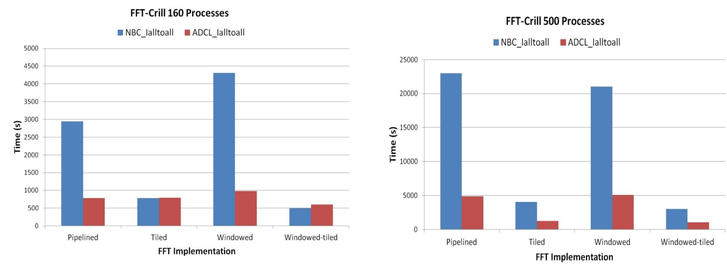
- Auto-tunes different non-blocking LibNBC-based implementations of collective operations
- Allows the overlap of computation and communication
- Supports default Ibcast, Ialltoall, Ireduce and Iallgatherv collective operations
- The number of progress calls can be tuned as an attribute
- Uses Timer object in order to provide an accurate performance evaluation for each implementation (measures the time of both computation and communication)

```
ADCL_Request req;
ADCL_Timer timer;

// Initialize non-blocking persistent collective operation
ADCL_Ialltoall_init ( sbuf, scount, sdat, rbuf, rcount, rdat, comm, &req);
// Associate request with a timer object
ADCL_Timer_create ( req, &timer);

//Main application loop
for (i=0; i<MAXIT; i++ ) {
   //Start timer
   ADCL_Timer_start (timer);
   ...
   // Start communication operation
   ADCL_Request_init (req);
   ...
   // Progress communication operation
   ADCL_Request_progress (req);
   ...
   // Wait for completion
   ADCL_Request_wait (req);
   ...
   // Stop timer
   ADCL_Timer_end (timer);
}
```



Experimental Results

- ADCL_Ialltoall vs. LibNBC_Ialltoall
- FFTW kernel
- 160 and 500 processes
- Four implementation, 350 iterations each



References

[1] Auto-tuning Non-blocking Collective Communication Operations Youcef Barigou, Vishwanath Venkatesan and Edgar Gabriel. *Accepted for publication at The International Workshop on Automatic Performance Tuning (iWAPT), held in conjunction with the 29th IEEE International Parallel & Distributed Processing Symposium (IPDPS). Hyderabad, INDIA, 2015*

[2] Edgar Gabriel, Saber Feki, Katharina Benkert, and Michael M. Resch. Towards Performance Portability through Runtime Adaption for High Performance Computing Applications. *Concurrency and Computation | Practice and Experience, 22(16), 2010*