

# PROCESS ARRIVAL PATTERN FOR MPI COLLECTIVE OPERATIONS

*Based on Paper : A Study of Process Arrival Patterns for MPI Collective Operations*

Presenter: Zheng Gu

# Objective of this study



- What are the process arrival patterns in practical MPI programs
- Whether we can control the process arrival patterns in our MPI program
- What is the impact of imbalanced process arrival patterns on collective communication algorithms
- How can we deal with imbalanced process arrival patterns

# Definition

MPI collective communication

Process arrival pattern

Balance & imbalance process arrival pattern

# MPI Collective Communication

- In MPI, collective communication are implemented on top of point-to-point operations(send/recv) and require more than two parties, e.g broadcast, reduce.
- Three classes of collective operations:
  - Synchronization: **MPI\_Barrier**
  - data movement: broadcast, scatter, gather
  - collective computation: reduce, scan

# Process Arrival Pattern

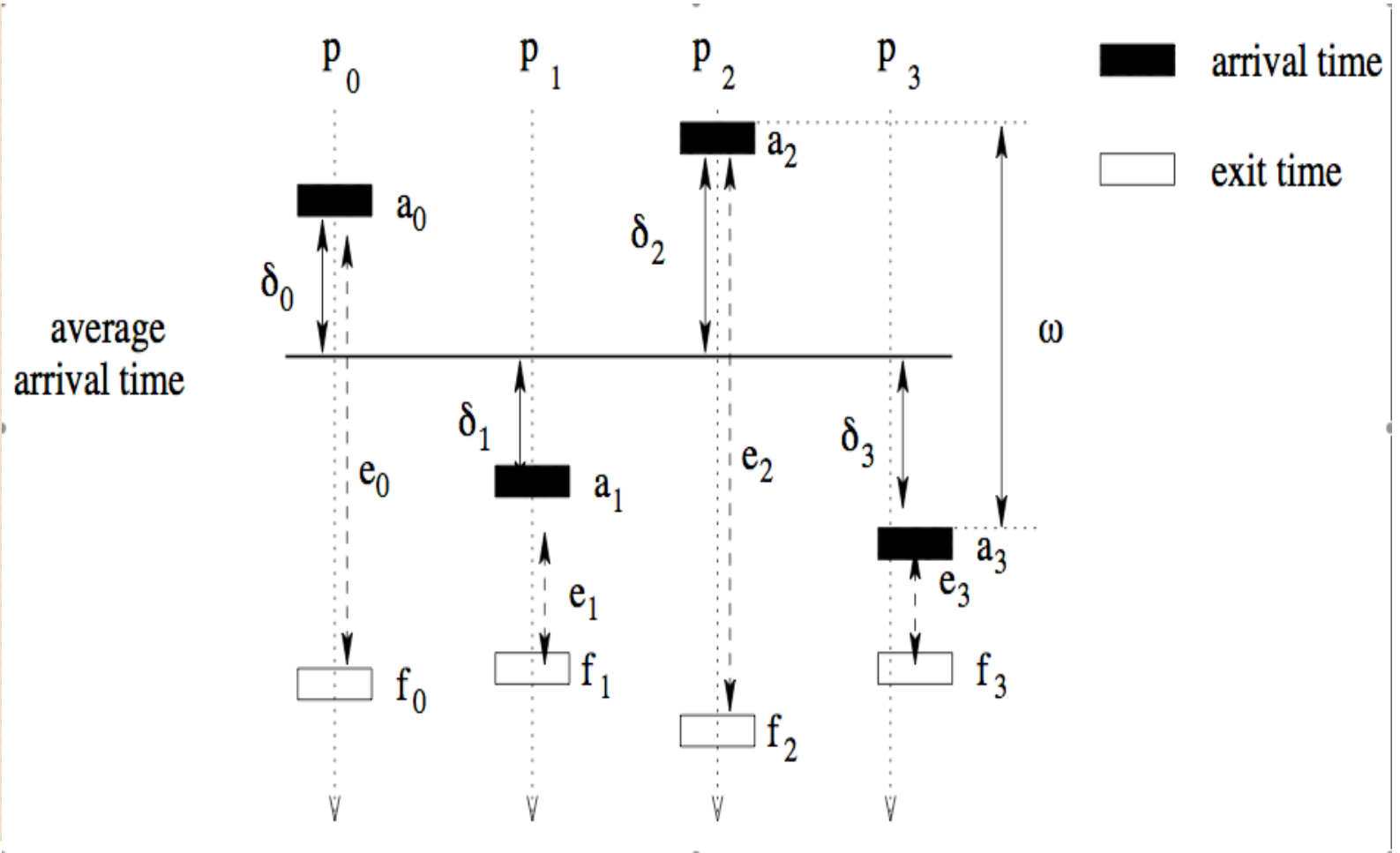


- The timing when different processes arrive at an MPI collective operation
- Caused by early completion or late start of some messages
- Process arrival pattern V.S. application load balancing problem
  - ALB is orders of magnitude larger
  - It is possible to balance ALB, but not PAP, why?

# Why hard to balance PAP



- The process arrival pattern is affected by:
  - Application
  - Operating System
  - System hardware
  - Communication library



## Process Arrival Pattern(Cont.)

# Define balance & imbalance

- Let  $T$  be the time to communicate one message in the operation
- Represent process arrival pattern:  $a_i$
- Worst case imbalance time:  $\max(a_i) - \min(a_i)$
- Worst case imbalance factor is:  $\omega / T$ 
  - T is time to communicate one message in the operation
- If worst case imbalance factor  $< 1$ 
  - balanced
- Else
  - imbalanced

# How balanced is the process arrival pattern of existing MPI operations?

- Existing collective communication algorithms are designed, analyzed, and evaluated with the assumption of the balanced process arrival pattern. Is this good enough?

# Benchmarks on 2 typical clusters

Lemieux

Beowulf

# Lemieux



- 750 compaq alphaserver ES45 nodes with 1 GHz SMP processors & 4GB memory
- Connection: Quadrics
- OS: Tru64 Unix Operating System

# Beowulf



- 16 Dell Dimension 2400 nodes with 2.8 GHz P4 Processor & 128MB memory
- Connection: Dell Powerconnect 2624 1Gbps Ethernet switch
- OS: Linux(Fedora)

# Data Collection



- The data is collected by the MPI wrapper library.
- Wrapper records an event at each MPI process for each entrance and exit of each MPI collective communication routine.
- Information includes timing, the operation, the message size
- Times are measured using the `MPI_Wtime` routine

benchmark	description
FT	solves PDE with forward and inverse FFTs
IS	sorts integer keys in parallel
LAMMPS	simulates dynamics of molecules in different states
PARADYN	sim. dynamics of metals and metal alloys molecules
NBODY	simulates effects of gravitational forces on $N$ bodies
NTUBE 1	performs molecular dynamics calculations of diamond
NTUBE 2	performs molecular dynamics calculations of diamond

## Summary of Application Benchmark

benchmark	routine	msg size (byte)	dyn. count
FT	alltoall	131076	22
	reduce	16	20
IS	alltoallv	33193*	11
	allreduce	4166	11
	alltoall	4	11
LAMMPS	allreduce	42392	2012
	bcast	4-704	48779
	barrier		4055
PARADYN	allgatherv	6-1290*	16188
	allreduce	4-48	13405
NBODY	allgather	5000	300
NTUBE 1	allgatherv	16000*	1000
NTUBE 2	allreduce	8	1000

## Summery of Benchmark(cont.)

benchmark	imbalance factor			
	Lemieux (n = 128)		Beowulf	
	average	worst	average	worst
FT	91.0	652	278	1.2K
IS	61.0	358	1.4K	11K
LAMMPS	4.00	19.0	273	630
PARADYN	9.10	46.0	12.0	79.0
NBODY	13.0	132	12.0	50.0
NTUBE 1	4.8K	38K	4.30	17.0
NTUBE 2	85K	347K	9.00	39.0

## Statistics of Process Arrival Pattern

- average of the worst case imbalance factors for all programs on both clusters are quite large, which excludes the factor of load balance
- PAP depends on system architecture. NTUBE1 & NTUBE2 has smaller IF on Beowulf

# Micro benchmark

```
(1) MPI_Barrier(...);
(2) for (i=0; i<1000; i++) {
(3)     /* compute for roughly X milliseconds */
(4)     for (m=0; m< XTIME; m++)
(5)         for (k=1, k<1000; k++)
(6)             a[k] = b[k+1] - a[k-1] * 2;
(7)     arrive[i] = MPI_Wtime();
(8)     MPI_Alltoall(...);
(9)     leave[i] = MPI_Wtime()
(10)}
```

Figure 4: Code segment for a micro-benchmark

comp. time	worst case imbalance factor( $\frac{\bar{w}}{T}$ )		
	exit	computation	arrival
50ms	15.2 ± 0.7	23.4 ± 0.3	32.5 ± 0.8
100ms	15.2 ± 0.6	46.8 ± 1.6	54.5 ± 1.9
200ms	15.0 ± 0.3	87.4 ± 1.8	92.7 ± 1.9
400ms	15.1 ± 0.8	160 ± 1.9	164 ± 2.0
800ms	15.0 ± 0.3	320 ± 3.6	322 ± 3.6

Statistics of micro-benchmark on Lemieus

comp. time	worst case imbalance factor ( $\frac{\bar{\omega}}{T}$ )		
	exit	computation	arrival
50ms	$5.07 \pm 1.29$	$3.16 \pm 0.02$	$7.02 \pm 1.29$
100ms	$4.32 \pm 1.00$	$7.52 \pm 0.02$	$9.53 \pm 0.99$
200ms	$3.71 \pm 0.11$	$14.18 \pm 0.02$	$15.17 \pm 0.06$
400ms	$6.22 \pm 0.23$	$31.41 \pm 0.30$	$33.17 \pm 0.35$
800ms	$11.62 \pm 0.41$	$56.24 \pm 0.05$	$56.29 \pm 0.20$

Statistics of micro-benchmark on Beowulf

# Conclusion from statistics



- Different processors take different time to run the same computation
- Different processors take different time to perform the communication
- The imbalance in both communication and computation are contributing to the imbalance in the process arrival patterns

# Impacts of imbalanced PAP

MPI\_Bcast

MPI\_Alltoall

# Attribute of operations



- MPI\_Alltoall
  - Inherently synchronized
- MPI\_Bcast
  - Not inherently synchronized

# Summary of evaluation



- Collective communication algorithms respond differently to different process arrival patterns
- The algorithm that performs better with a balanced process arrival pattern tends to perform worse when the process arrival pattern becomes more imbalanced
- The impact of an imbalanced process arrival pattern can be large.



# Solution

MPI\_Alltoall

# Pair/Ring algorithm

## □ Pair

- Suppose there are  $n$  processors. Of total  $n-1$  steps, in step  $i$ , process of  $j$  exchanges a message with process of  $j \text{ xor } i$

## □ Ring

- Suppose there are  $n$  processors. Of total  $n-1$  steps, in step  $i$ , process of  $j$  sends a message to process of  $(j+i) \bmod n$  and receives a message from process of  $(j-i) \bmod n$

# Pair/Ring algorithm(cont.)



- Perform well when the process arrival pattern is balanced
- Can cause system contention when imbalance factor is large

# Ring/Pair + one MPI barrier



- Eliminate system contention
- Good performance when the worst case imbalance factor is small

# Ring/Pair + light barrier

- Ring/pair + one MPI barrier hurts the performance when a large number of processes arrive at the operation significantly earlier than others
- Idea is: whenever there is a possibility that two messages in different phases can be sent to the same processes at the same time and cause contention, a light barrier that performs pair-wise synchronization is added to sequentialize the two messages.

# Simple algorithm



- Posts all receives and all sends, starts the communications, and waits for all communications to finish
- Eliminates the coordination among processes.
- Performs well for sufficiently imbalanced process arrival patterns.

# Summary of result



- Denote improve routine as ROBUST and the native routine as NATIVE
- Test against MICRO benchmarks and application benchmarks
- On different platforms, observe speed-up up to 55.7% of all benchmarks by using ROBUST over NATIVE
- Mainly attributed to taking process arrival pattern into consideration.



Q&A

# Discussion



- Do you think process arrival pattern aware algorithms could be applied to other collective operations? Why?



Thank You!