# Parallel Programming Environment for a V-Bus based PC Cluster

Sang Seok Lim, Yun Heung Paek, Kyu Ho Park and Jay Hoeflinger

October 10, 2001

**Speaker : Sang Seok Lim**

CORE Lab. EECS KAIST Korea

# Contents

- Introduction
- V-Bus based PC cluster
  - V-Bus network card
- Programming Environment
  - MPI
    - One-sided communication : Mpi_put/Mpi_get
  - CORE-polaris
    - LMAD
    - Data scattering/collecting
    - Communication Optimization
- Experimental Results
- Conclusion

# Introduction

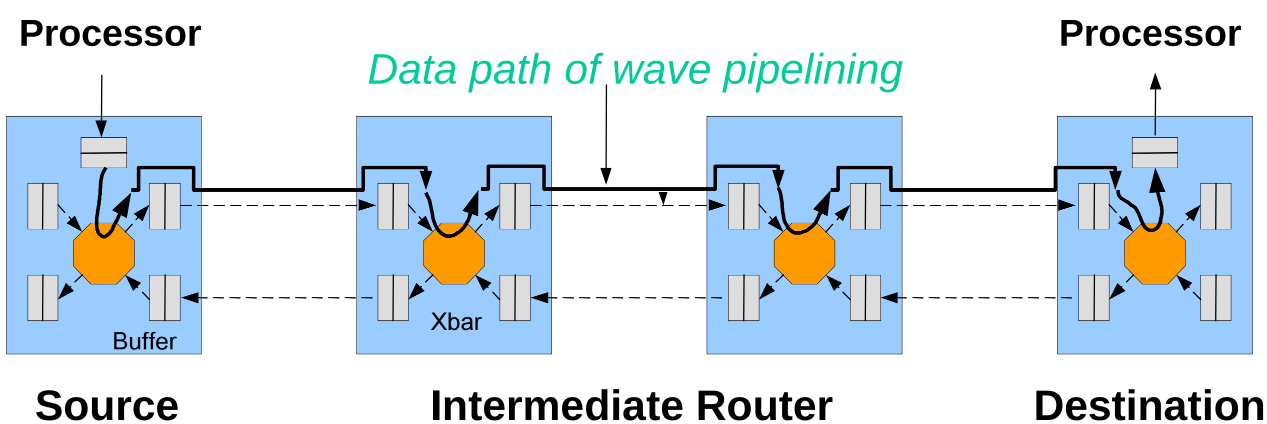
- V-Bus based PC cluster
  - PCs are interconnected with V-Bus network cards
  - V-Bus network card implemented on FPGA technology
- Programming environment for a V-Bus based PC cluster
  - MPI : MPI and one sided communication
  - Parallelizing compiler : CORE-polaris

CORE

Core Polaris

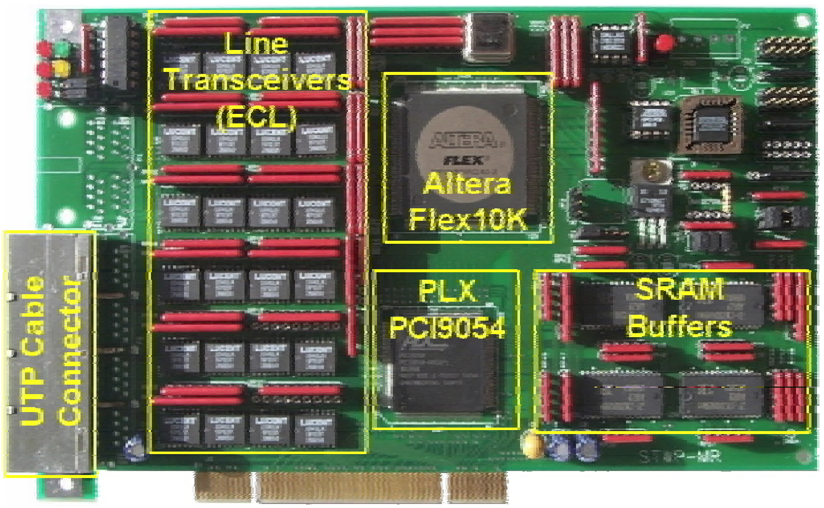
# V-Bus based PC Cluster(2)

- V-Bus
  - Network establish a bus virtually only when a bus is required
  - *Efficient broadcasting* without extra physical buses
- Skew tolerant wave pipelining
  - Automatic skew sampling circuit



# V-Bus based PC cluster(3)

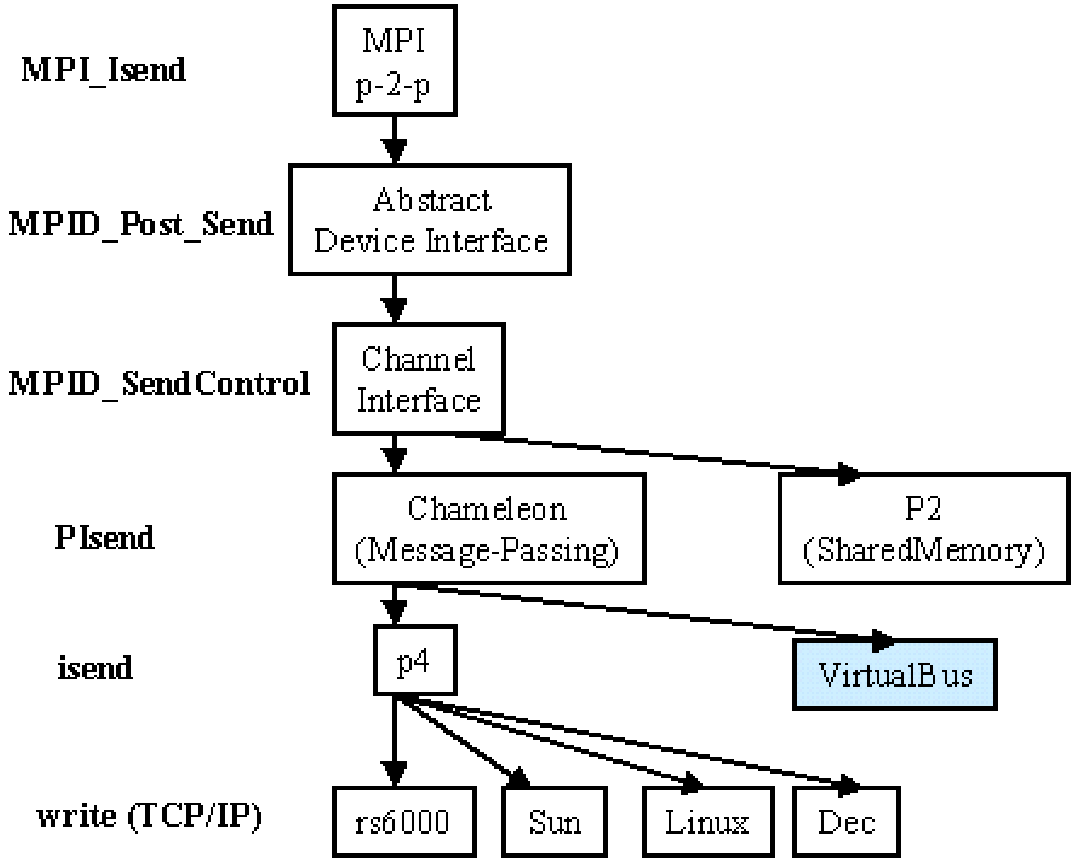
- The interconnection network architecture
  - Mesh interconnection network



# MPI (1)

- Extensions to the Message Passing Interface
  - Develop a new ADI in MPICH for a V-Bus network card
  - Support all functions of MPI-1
  - A part of MPI-2
    - One sided communication
    - Mpi_put/Mpi_get : strided and contiguous region transfering
    - Mpi_Win_lock, Mpi_Win_unlock, Mpi_fence etc.
- Feature of our MPI-2
  - Optimization of collective communication primitives using *V-Bus Broadcasting operation*
  - User-level communication

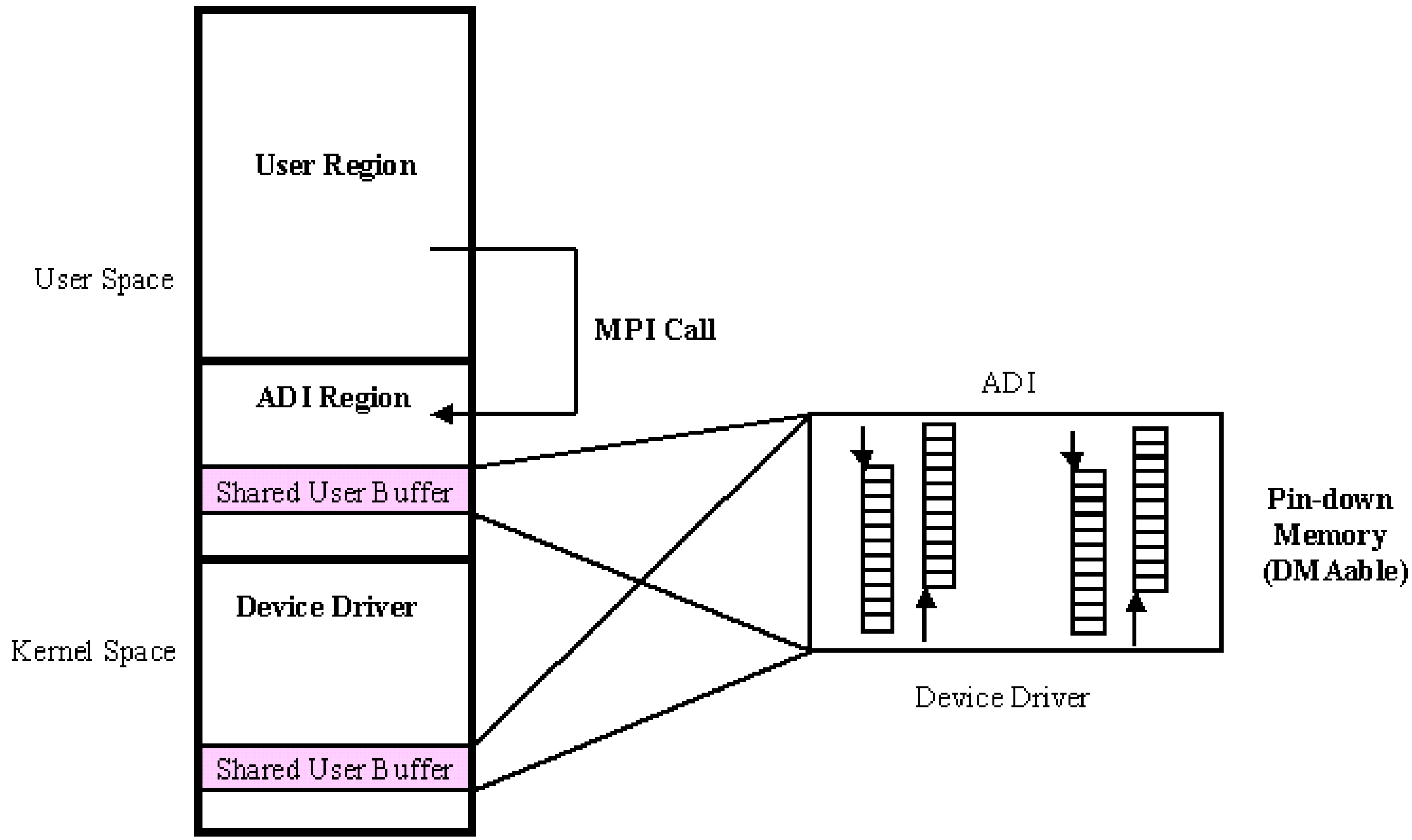
# MPI (2) : MPICH


MPI_Isend
MPI p-2-p
MPID_Post_Send
Abstract Device Interface
MPID_SendControl
Channel Interface
PIsend
Chameleon (Message-Passing)
P2 (SharedMemory)
isend
p4
VirtualBus
write (TCP/IP)
rs6000
Sun
Linux
Dec

CORE

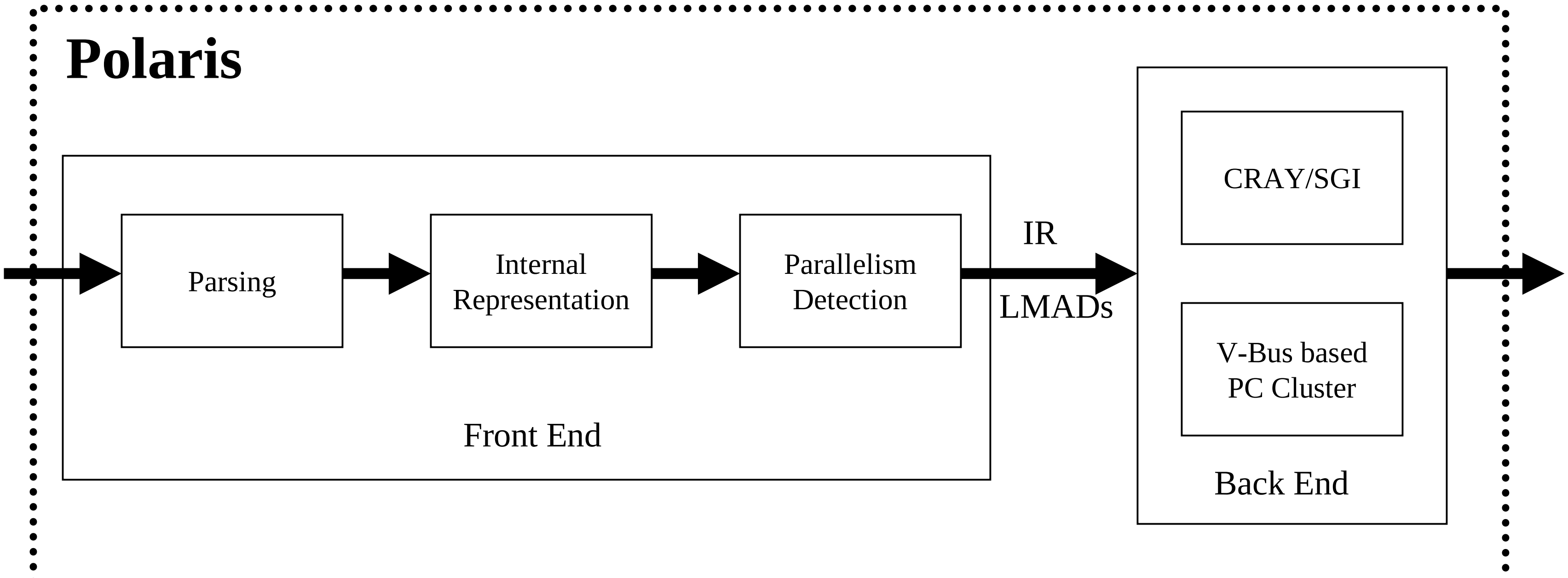
# MPI(3) : Communication Mechanism

MPI



# Parallelizing Compiler: polaris

- Generate a parallel program targeted for shared memory machines



# LMAD(1)

- To detect a parallel loop, the compiler need to analyz e memory access patterns of all variables
- Accuracy of array access analysis is important
- Linear Memory Access Descriptor : LMAD
  - Array access representation
  - Dependency test : Access Region Test
  - Communication generation
    - Data scattering and collecting

# LMAD(2)

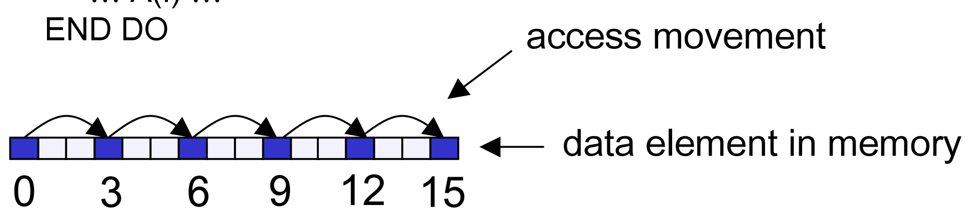
- Describes access movement through memory in terms of a series of dimensions
- Three elements
  - Stride, span, base offset
  - notation

$$A^{stride1=\alpha_1, stride2=\alpha_2, \ldots, stride\,\mathrm{n}=\alpha_n}_{span1=\delta_1,\ span2=\delta_2,\ \ldots, span\,\mathrm{n}=\delta_n} + B$$

# LMAD(3)

- Stride
  - Movement through memory

```
DO I=1,11,3
    … A(I) …
END DO
```



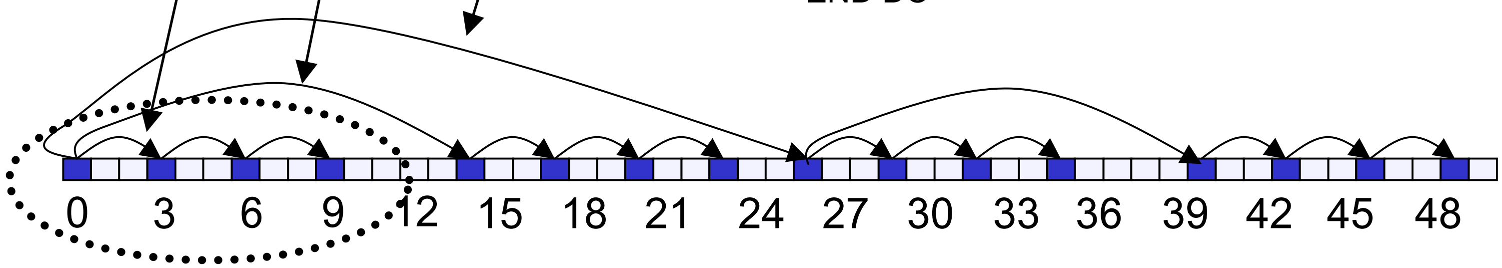

- Base offset
  - The data element access movement starts from…

# LMAD(4)

- Span
  - The total element length that the access traverses when the index iterate its entire ranges
- Example

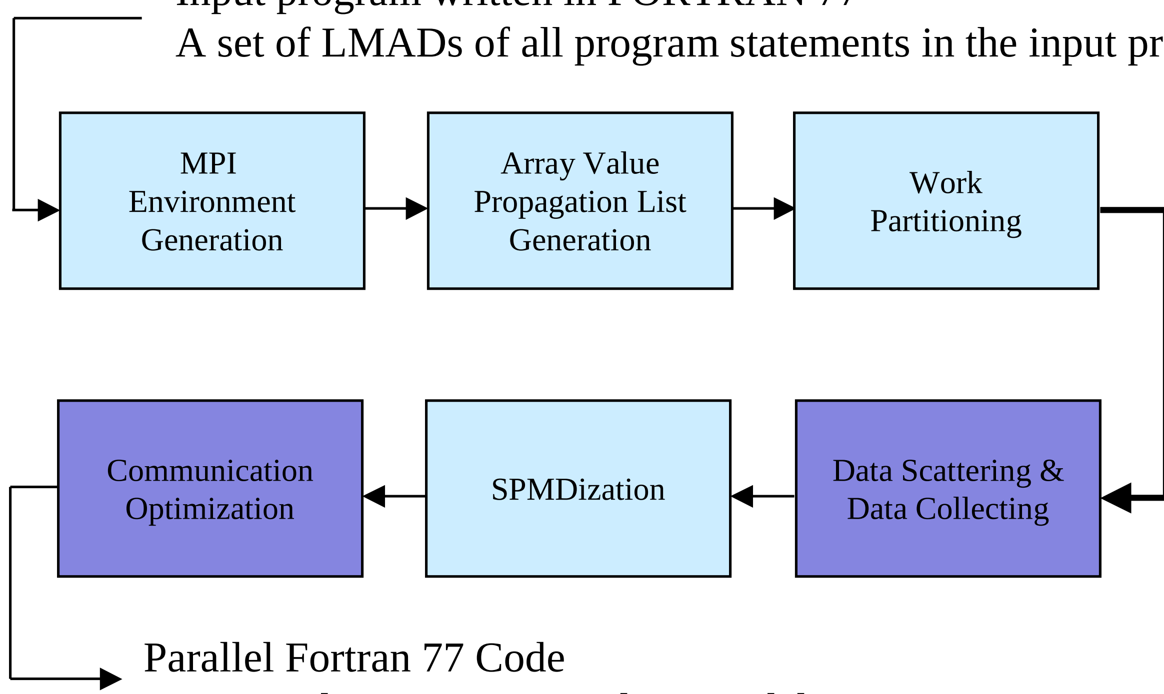
$$A^{stride1=3,stride2=14,stride3=26}_{span1=9,span2=14,span3=26}+0$$

```
REAL        A(14,*)
DO I=1,2
            DO J=1,2
                        DO K=1,10,3
                        … A(K+26*(I-1),J)
                        END DO
            END DO
END DO
```



# CORE-polaris

Input program written in FORTRAN 77
A set of LMADs of all program statements in the input program




Parallel Fortran 77 Code
SPMD style using Master/Slave model

# Data scattering and collecting

- Data scattering
  - At the entrance of a parallel region the master identifies th e objects that each slave *may access* within the regions
  - Copies the objects to the memory of each slave
- Data collecting
  - Identifying any *modified data* from slaves at the end of the parallel region
  - Updating the copies in *master's memory* with the modified data

# Data scattering using LMAD(1)

- Original LAMD is splitted into two sub-LMADs
  - **LMAD$_{offset}$** : calculate a set of data offsets from base address in a sequence of communication primitive generation
  - **LMAD$_{mapping}$** : map data access into communication primitives provided by the MPI-2 library

$$A^{stride1=\alpha_1,\ stride2=\alpha_2,\dots,stride\,n=\alpha_n}_{span1=\delta_1,\ span2=\delta_2,\ \dots,span\,n=\delta_n}+B$$

**LMAD$_{mapping}$**

$$A^{stride1=\alpha_1}_{span1=\delta_1}$$

**LMAD$_{offset}$**

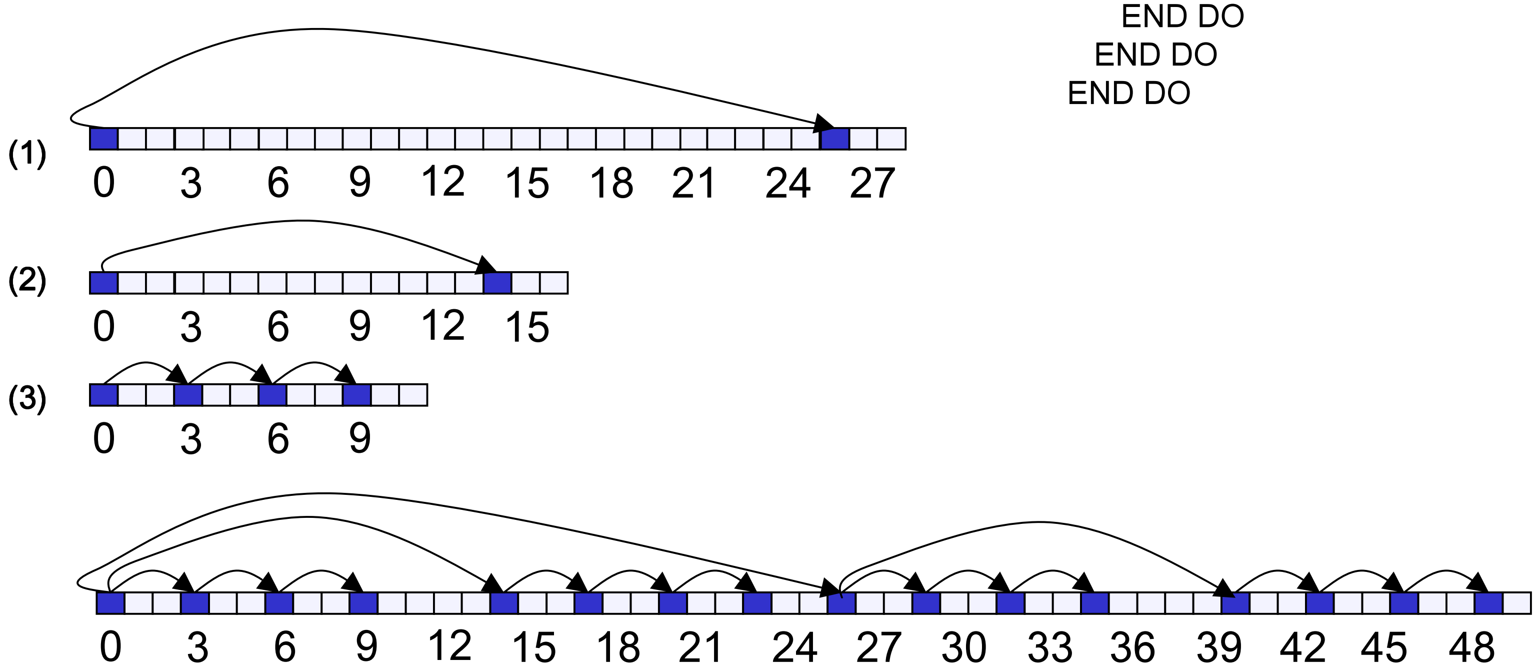
$$A^{stride2=\alpha_2,\dots,stride\,n=\alpha_n}_{span2=\delta_2,\ \dots,span\,n=\delta_n}$$

# Data scattering using LMAD(2)

(3) (2) (1)

$$A^{3,14,26}_{9,14,26}+0$$

```
     REAL         A(14,*)
(1)  DO I=1,2
(2)    DO J=1,2
(3)      DO K=1,10,3
                  … A(K,J+26*(I-1))
         END DO
       END DO
     END DO
```



# Data scattering using LMAD(3)

- The set of offsets that are calculated from **LMAD**$_{\textbf{offset}}$

$$\{x_1 \times \alpha_2 + x_2 \times \alpha_3 + \ldots + x_n \times \alpha_n \mid 0 \le x_i \le \delta_i / \alpha_i,$$
$$\text{where } x_i \text{ is non-negative integer}\}$$

- **LMAD**$_{\textbf{mapping}}$ is mapped into Mpi_put/Mpi_get as follows
  - If the stride of **LMAD**$_{\textbf{mapping}}$ is constant and
    - stride of **LMAD**$_{\textbf{mapping}}$ = 1 : *contiguous Mpi_put/Mpi_get*
    - stride of **LMAD**$_{\textbf{mapping}}$ > 1 : *stride Mpi_put/Mpi_get*
  - If the stride of **LMAD**$_{\textbf{mapping}}$ is not constant
    - one memory element by one memory element

# Communication optimization(1)

- Network latency is higher than conventional supercomputers
- Small *exact* regions -> *approximate* regions
- Three granularity levels based on LMAD
  - Fine, Middle, Coarse granularity

# Communication optimization(2)

- *Fine* granularity
  - Exact regions are always transferred
  - The number of communication

    $$(\delta_2 / \alpha_2) \times (\delta_3 / \alpha_3) \times \ldots \times (\delta_n / \alpha_n) + 1$$
- *Middle* granularity
  - **$\mathbf{LMAD}_{\mathbf{mapping}}$** is always mapped into contiguous Mpi_put/ge t, even the stride is greater than 1.
  - Contiguous Mpi_put/get has high throughput and low la tency than those of stride Mpi_put/get
  - The number of communication

    $$(\delta_2 / \alpha_2) \times (\delta_3 / \alpha_3) \times \ldots \times (\delta_n / \alpha_n) + 1$$

# Communication optimization(3)

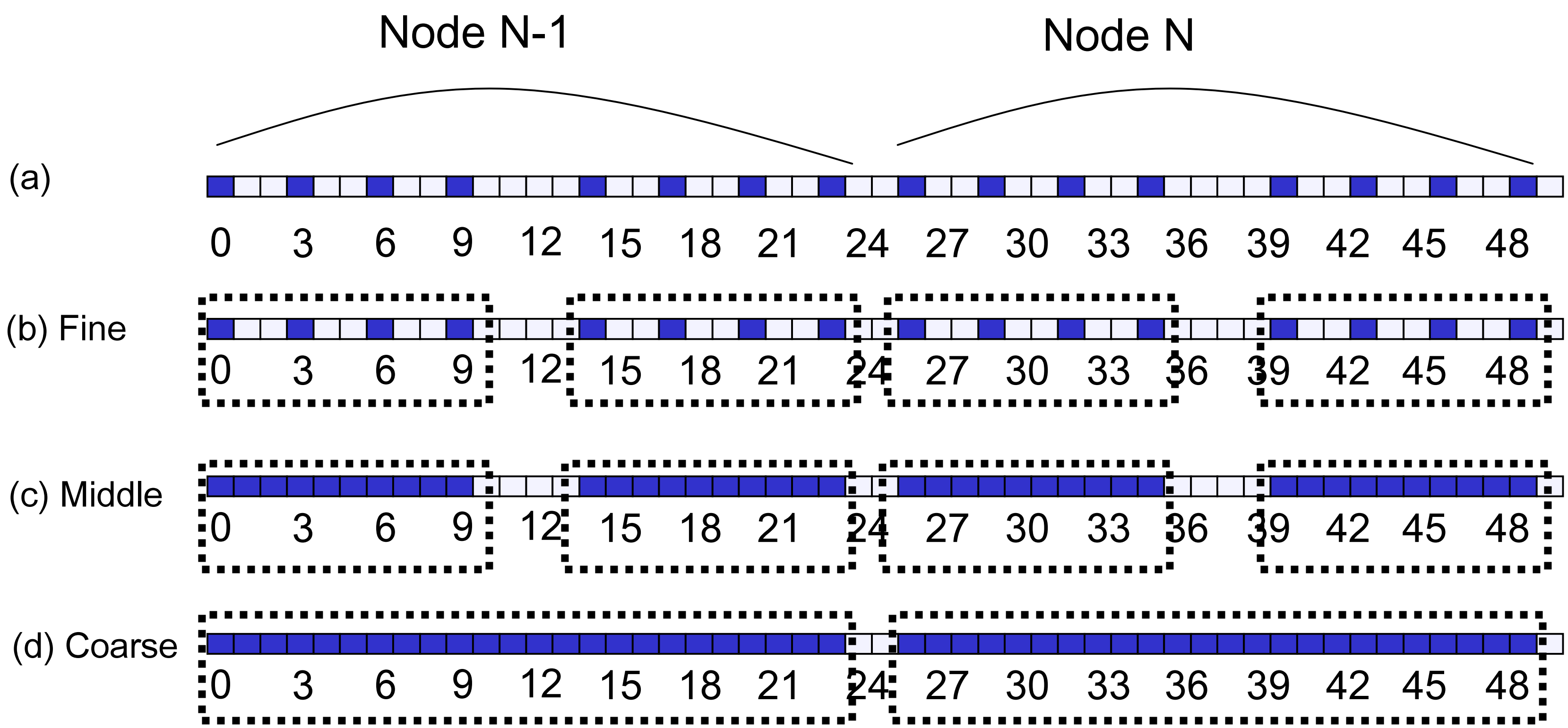
- *Coarse* granularity
  - Merging scattered regions into a big approximate regions

$$\textbf{LMAD}_{\textbf{offset}} \quad A^{stride\ n\text{-}1=\alpha_{n-1}}_{span\ n\text{-}1=\delta_{n-1}} \qquad \textbf{LMAD}_{\textbf{mapping}} \quad A^{stride1=\alpha_n}_{span1=\delta_n}$$

  - The number of communication

$$(\delta_n / \alpha_n) + 1$$

# Communication optimization(4)

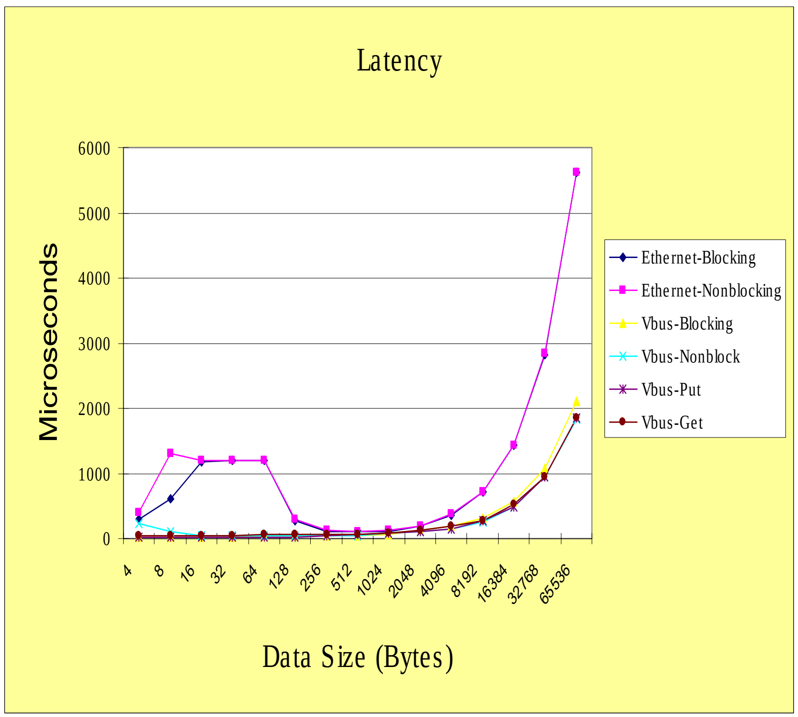


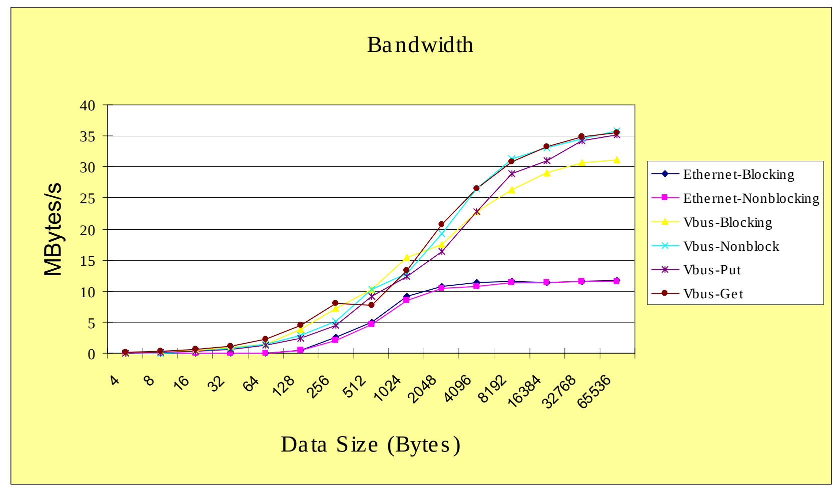
# Communication optimization(5)

- If *overlapped data objects* in approximate regions
  - Race condition happens at data collecting phase
  - Check the upper and lower bound of approximate regions
  - If overlapped data objects exist
    - *Fine granularity* communication
- For now, it is up to the user that select the optimal gr anularity with the help of profiling routines

# Experiment : MPI-2

- 100Mbps Ethernet VS V-Bus network



# Experiment : CORE-polaris

- 300MHz , 64MB memory

**Table 1. Total execution time of the MM code**

| Speedups | | Array Size | | |
|---|---|---|---|---|
| | | 256*256 | 512*512 | 1024*1024 |
| # of Nodes | 1 | 0.96 | 0.96 | 0.96 |
| | 2 | 1.086 | 1.53 | 1.60 |
| | 4 | 1.75 | 2.74 | 3.033 |

**Table 2. Communication time for matrix multiplication, swim and CFFZINIT of TFFT**

| Total Communication Time (sec) | Granularity | | |
|---|---|---|---|
| | fine | middle | coarse |
| MM(1024*1024) | 0.72 | 0.89 | 0.01128 |
| Swim(ITMAX=1) | 0.20590 | * | 0.072166 |
| CFFZINIT(M=11) | 0.3584 | 0.0768 | 0.0068 |

# Conclusion/Further work

- Two programming interfaces for a V-Bus based PC cluster
  - MPI and MPI-2
    - One sided communication
    - User level communication
    - Optimized for V-Bus network
  - CORE-polaris
    - Data scattering/collecting using LMAD
    - Communication optimization for V-Bus based network
- Further work
  - More large scale PC-cluster(4 by 4)
  - Cross loop/subroutine communication optimization