Non-blocking Java Communications Support on Clusters

Guillermo L. Taboada*, Juan Touriño, Ramón Doallo

computer architecture group

UNIVERSIDADE DA CORUÑA
SPAIN

{taboada,juan,doallo}@udc.es

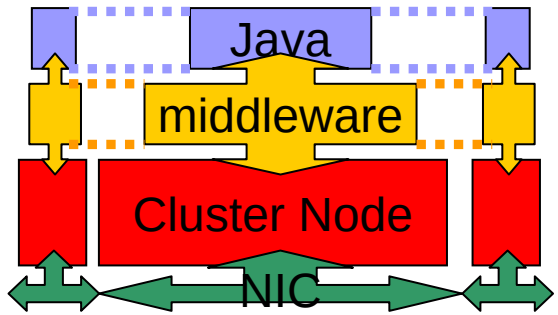
13th European PVM/MPI Users's Meeting (EuroPVM/MPI'06), Bonn

Outline

- Introduction
- Designing Java Communication Libraries on Clusters
- Implementing Efficient Java Communication Libraries on Clusters
- Performance Evaluation
- Conclusions

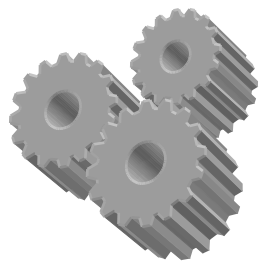
Introduction

- ↑ interest on clusters (↑ computation ↓ cost)
- Growing solution:
  - Java (and HPC Java) on clusters
- Challenge: scalable peformance cluster+Java
  - Network (physical level) performance scales
  - Java middleware less efficient than native code
    - → Java is not going to scale performance
    - High Performance Networks not supported or supported with poor performance
      - Ways of support:
        - IP Emulations
        - High Performance Sockets



Introduction

- **Previous work:**
  - **Non-blocking communication support**
    - **Java NIO (New I/O)**
      - **Improves scalability, basic in client/server applications**
    - **Message-Passing Java**
      - **mpiJava, wrapper to native MPI implementation that supports non-blocking comms.**
      - **MPJ Express, Java message-passing library (NIO communication device)**
      - **MPJ/Ibis, Java message-passing system with non-blocking support through multi-threading**
  - **High Performance network support**
    - **Almost centered on Myrinet**
      - **Solutions based on protocols designed *ad hoc*, poorly maintained and with numerous layers**
        - **Numerous libraries →**
          ↑ communication overhead

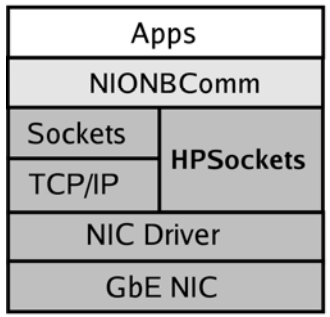


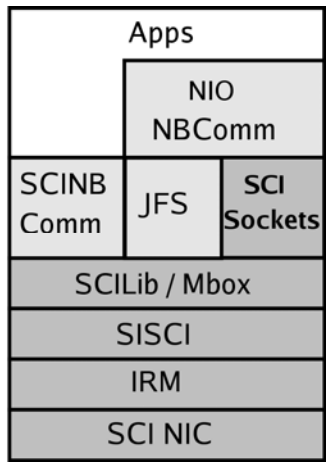
Introduction

- Our solution, 2 Java communication libraries:
  - NBComm
    - Non-blocking communications support
      - Focused on latency reduction
      - High Performance Network communication libraries support
  - Java Fast Sockets (JFS)
    - 1st High Performance Java Sockets implementation
    - Implements an API widely spread (Java Sockets) with ↑ performance compared to RMI
  - Both libraries support High Performance Networks
    - SCI, using native code
      - NBComm implementation (SCINBComm)
      - JFS implementation on SCI
    - GbE, “pure” Java solutions
      - NBComm implementation (NIONBComm)
      - JFS general implementation
    - Later: Myrinet…

Introduction

- **Overview of SCI & GbE communication libraries:**
  - **GbE**
    - **NIONBComm over Java NIO**
      - **High *overhead* due to TCP/IP stack**
      - ***Overhead* reduction using High Performance Sockets (Fast Sockets, SOVIA, GAMMAsockets)**
  - **SCI**
    - **IRM: low level driver**
    - **SISCI: communication library**
    - **SCILib: unidirectional message queues**
    - **Mbox: notification library**
    - **SCI Sockets: H.P. Sockets on SCI**
    - **SCINBComm: NBComm implementation**
    - **JFS over native libraries (SCILib/Mbox)**
    - **NIONBComm over Java Sockets**
      - **JFS**
      - **Java Sockets over SCI Sockets**

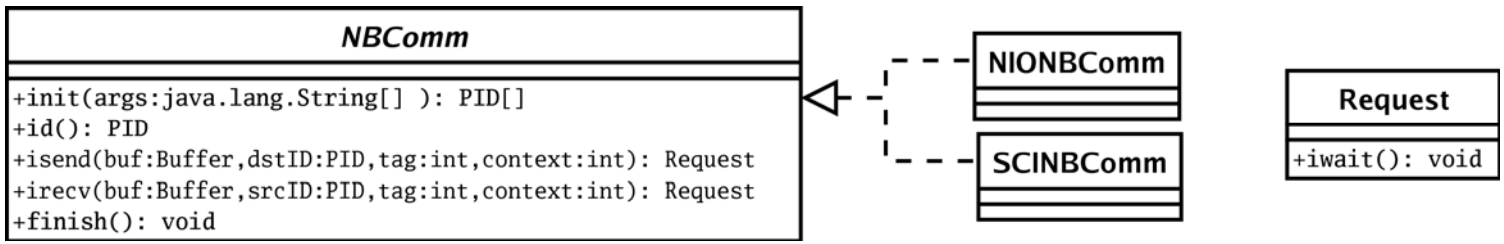




Designing Java Communication Libraries on Clusters

- **NBComm**
  - **Abstracts the communication level giving support to different interconnection networks → network level transparency**
  - **Provides Java with efficient non-blocking communication**
    - **Reduces data copying**
    - **Accesses to efficient native libraries**
    - **Allows overlapping Computation/Communication**
  - **Implementations:**
    - **NIONBComm: "pure" Java using Java NIO**
    - **SCINBComm: accessing SCI native libraries through JNI**

Designing Java Communication Libraries on Clusters



- **NBComm:**
  - **Abstract class *NBComm***
  - **Subclasses for supporting different interconnection networks**
    - ***SCINBComm* supports SCI**
      - **Native implementation over SCILib**
    - ***NIONBComm* on *-Ethernet**
      - **“pure” Java implementation using Java NIO**
  - ***Request* for handling *non-blocking* requests**

Designing Java Communication Libraries on Clusters

- **API de NBComm:**

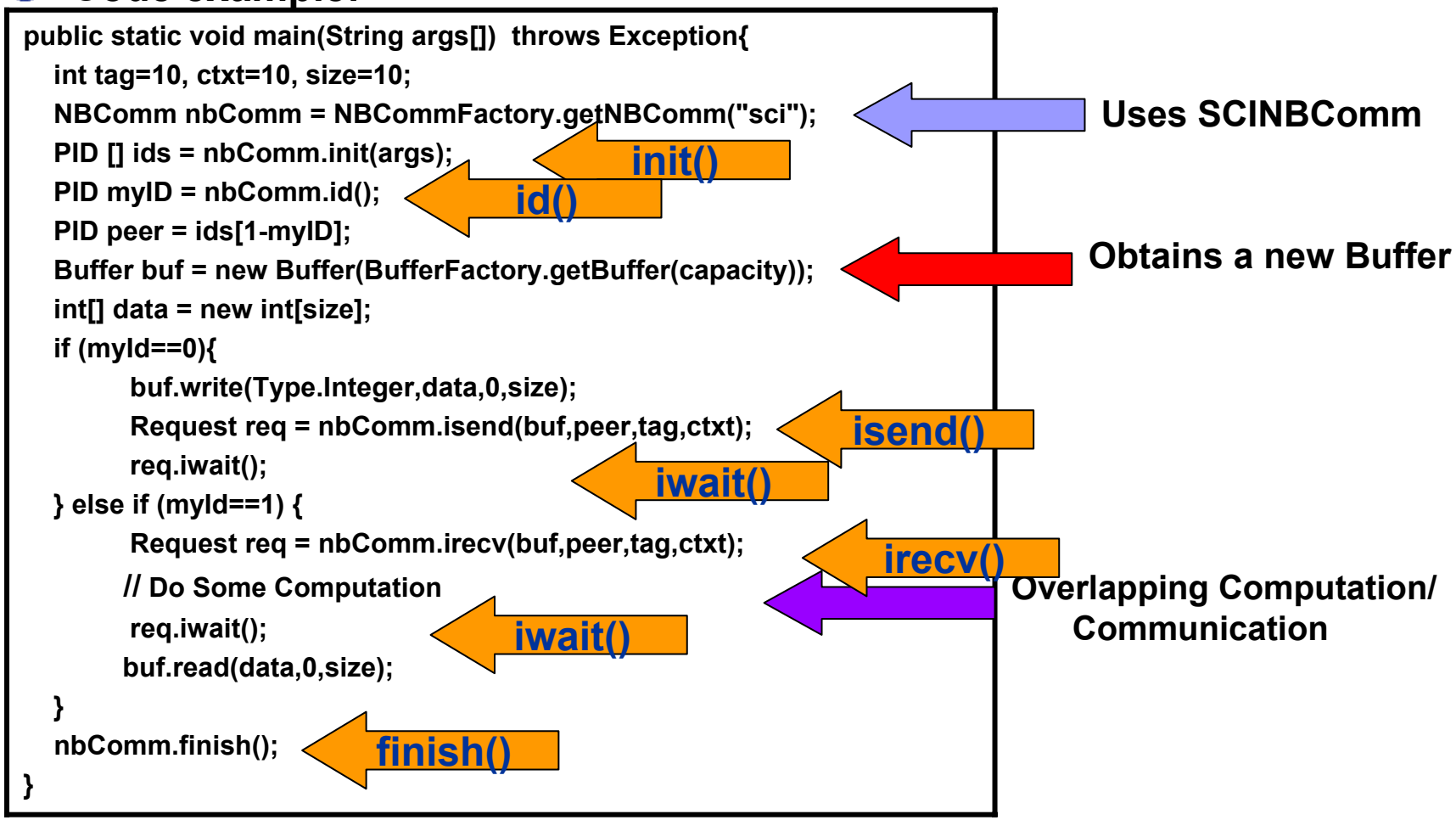
NBComm
+init(args:java.lang.String[]): PID[] +id(): PID +isend(buf:Buffer,dstID:PID,tag:int,context:int): Request +irecv(buf:Buffer,srcID:PID,tag:int,context:int): Request +finish(): void

  - **init() inits communication (args: configuration parameters)**
  - **finish() (barrier for all processes)**
  - **id() obtains the process identification (PID)**
  - **iwait() waits for the resolution of a request**
  - **isend() & irecv() perform communication using a *direct* ByteBuffer**

Designing Java Communication Libraries on Clusters

- Code example:

```
public static void main(String args[])  throws Exception{
   int tag=10, ctxt=10, size=10;
   NBComm nbComm = NBCommFactory.getNBComm("sci");
   PID [] ids = nbComm.init(args);
   PID myID = nbComm.id();
   PID peer = ids[1-myID];
   Buffer buf = new Buffer(BufferFactory.getBuffer(capacity));
   int[] data = new int[size];
   if (myId==0){
         buf.write(Type.Integer,data,0,size);
         Request req = nbComm.isend(buf,peer,tag,ctxt);
         req.iwait();
   } else if (myId==1) {
         Request req = nbComm.irecv(buf,peer,tag,ctxt);
        // Do Some Computation
         req.iwait();
        buf.read(data,0,size);
   }
   nbComm.finish();
}
```



Implementing Efficient Java Communication Libraries on Clusters

- The non-blocking communication is implemented in *NBComm* using a dedicated thread `receptor_thread`
  - The notification of message arrivals is made through:
    - a native `callback()` function
    - through Java NIO's `select()`
  - Arrived messages are kept in a message list using as key <PID,tag,context>

Implementing Efficient Java Communication Libraries on Clusters

- `receptor_thread` pseudocode

```
WHILE NBComm.finish() is not called
  IF pending_messages=0 THEN
       wait notification of new message arrival
  END IF
  receive new message header <PID,tag,context>
  IF message in message_list_to_process THEN
      receive message in the destination message buffer
      delete message from message_list_to_process
  ELSE
     receive message in temporal buffer
     add message to message_list_to_process
  END IF
  notify message reception to user_thread
END WHILE
```

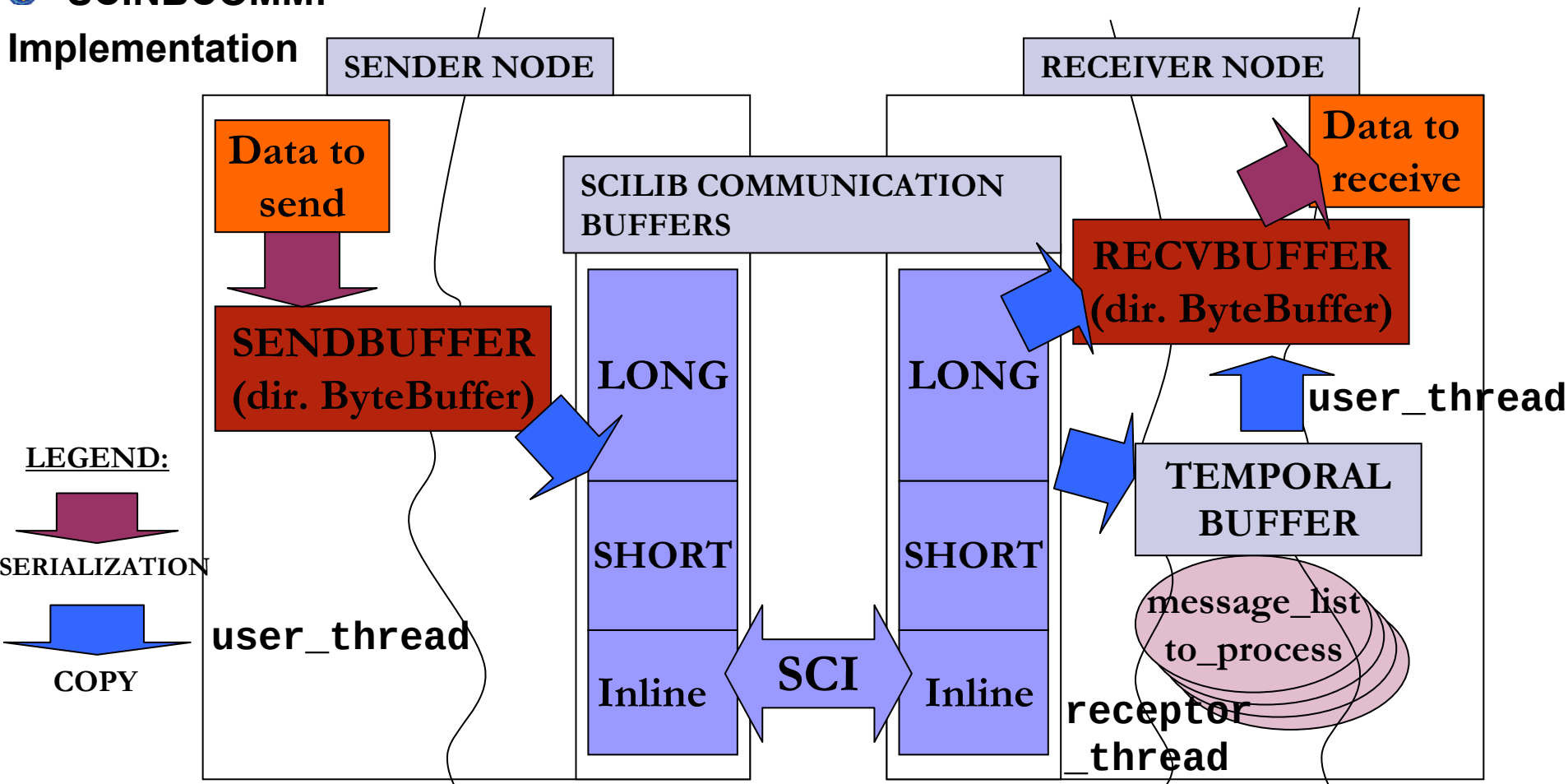
This is an active wait (polling) for a configured amount of time and then it changes to pasive wait

It is needed an agreed reception (rendezvous) for long messages (>128Kb configurable.)

If it is waiting in iwait() the reception is notified

Implementing Efficient Java Communication Libraries on Clusters

- **SCINBCOMM: Implementation**



Implementing Efficient Java Communication Libraries on Clusters

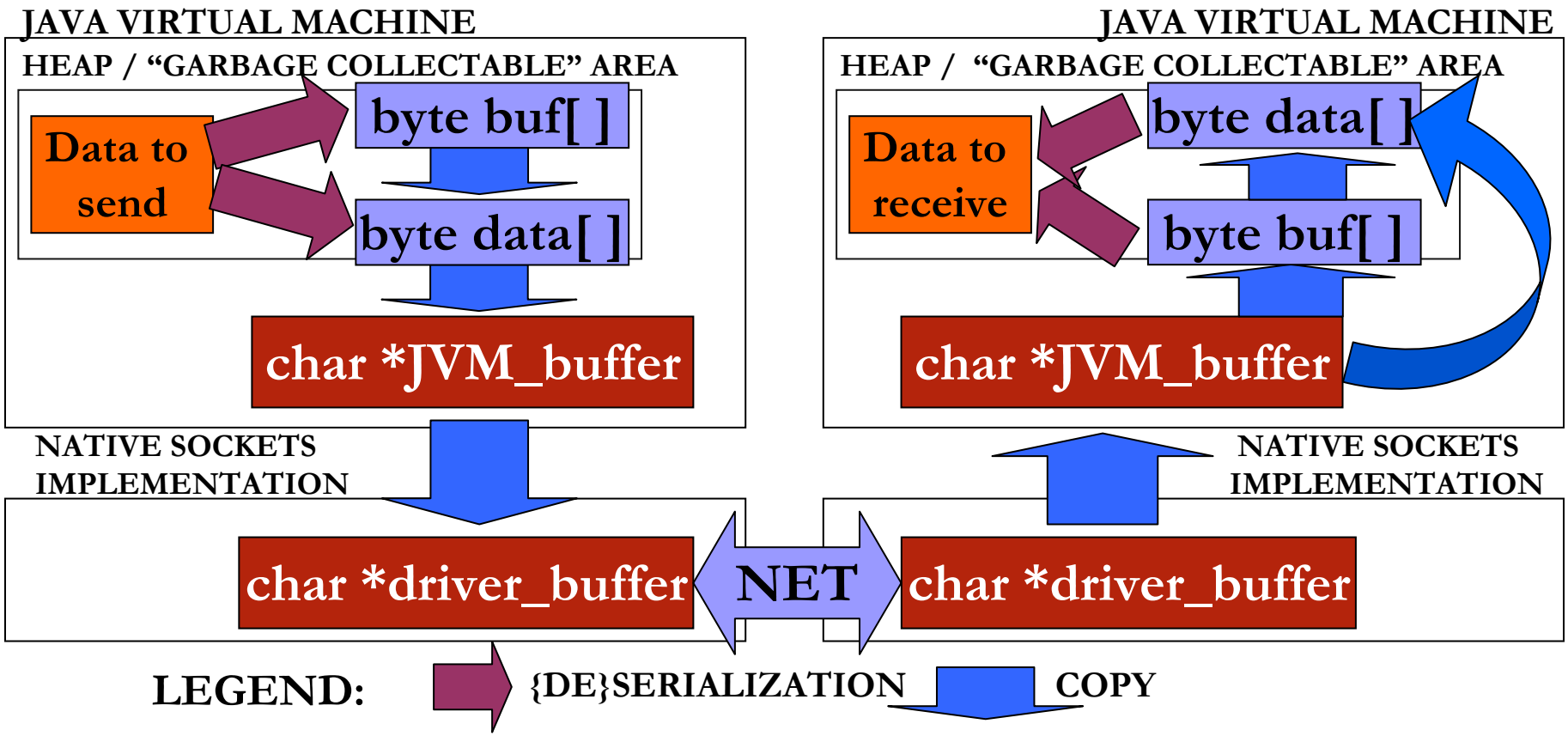
- Java Fast Sockets (JFS) implements Java Sockets API in a way:
  - Efficient & portable through:
    - general “pure” Java solution
    - Specific solutions that access native communication libraries (SCI Sockets)
    - The fail-over approach applied to the selection of libraries: the system tryes to use highly efficient native communication libraries. If this is not possible, uses the “pure” Java general solution
  - User transparency:
    - Setting *JFSFactory* as the default Sockets Factory in a small launcher application with `Socket.setSocketImplFactory()`.
      - This application will invoke using reflection the main method. All Sockets communications wil use JFS for then on.

Implementing Efficient Java Communication Libraries on Clusters

- Sun's Java Sockets implementation (Sun's JRE)
  - Only supports the TCP/IP stack communication library
  - Performs unnecessary copies
  - Do not implement communication optimization methods (*setPreferences() method*) related to:
    - Latency reduction
    - Maximizing bandwidth
- The use of Java NIO Sockets is more complex
  - Use of Socket Channels, Selectors, Buffers, etc…
  - Establishment of connections
  - Re-design communications of existing Socket-based applications

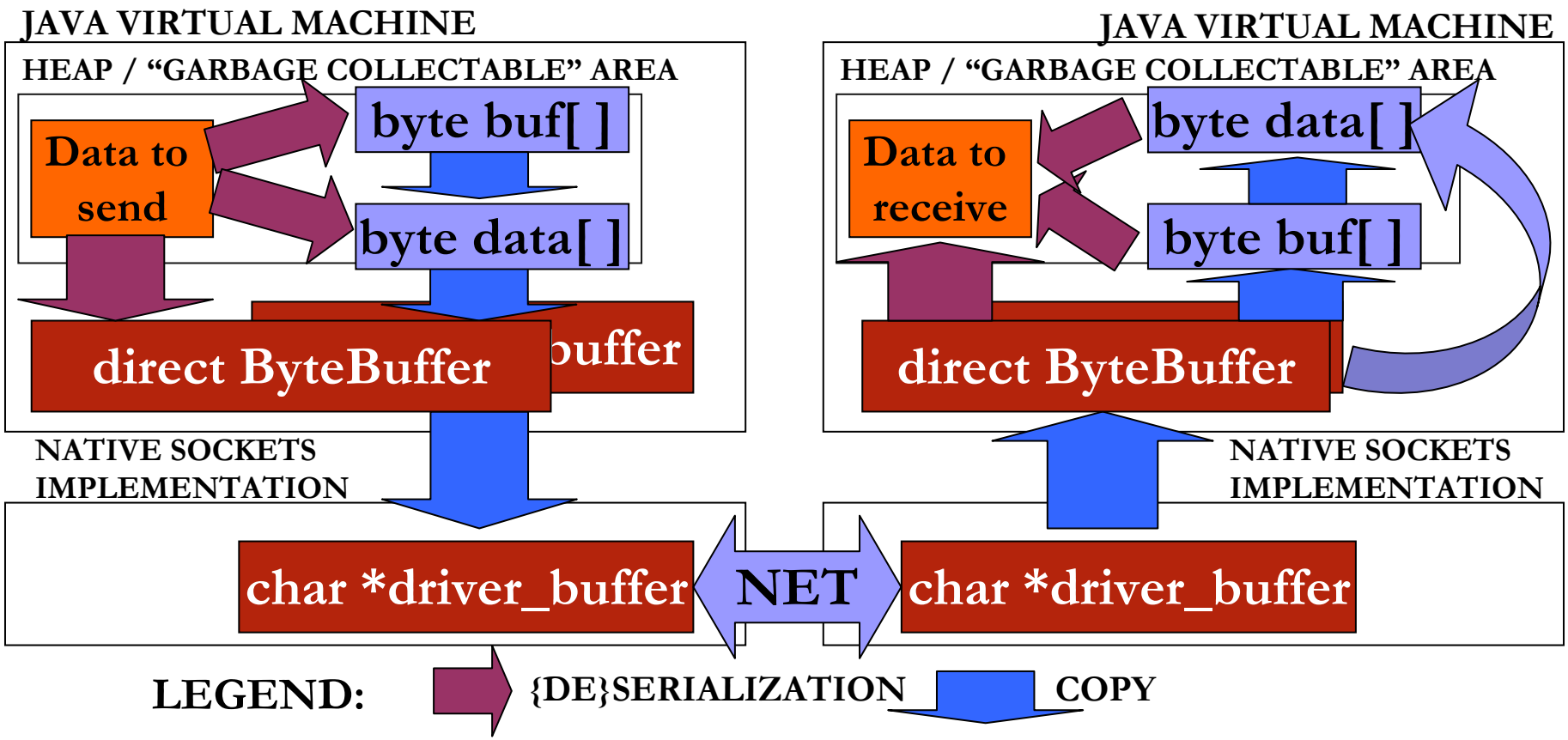
Implementing Efficient Java Communication Libraries on Clusters

- Default scenario in Sun's Java Sockets communication



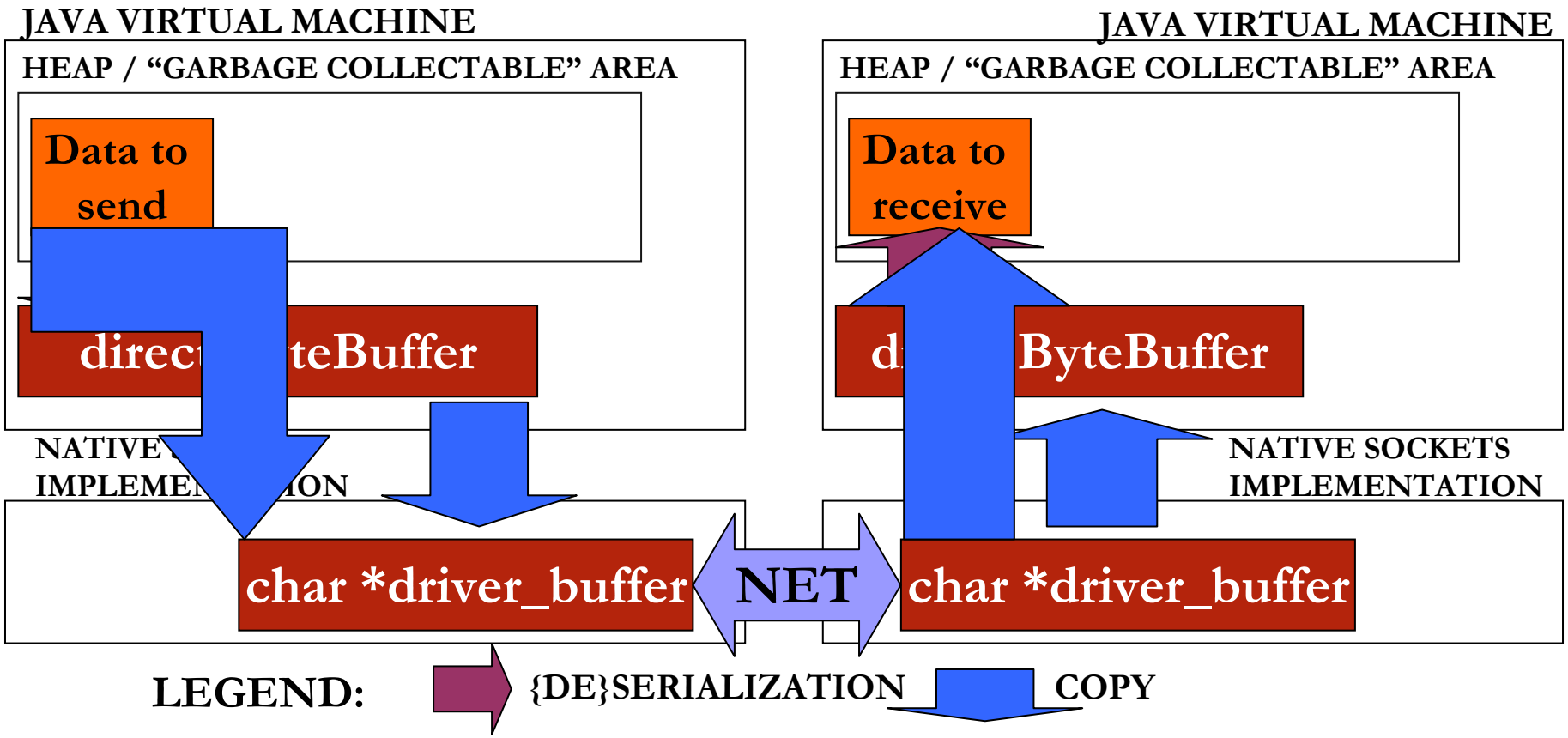
Implementing Efficient Java Communication Libraries on Clusters

- JFS communication using Java NIO direct ByteBuffer



Implementing Efficient Java Communication Libraries on Clusters

- JFS communication optimized (zero-copy)

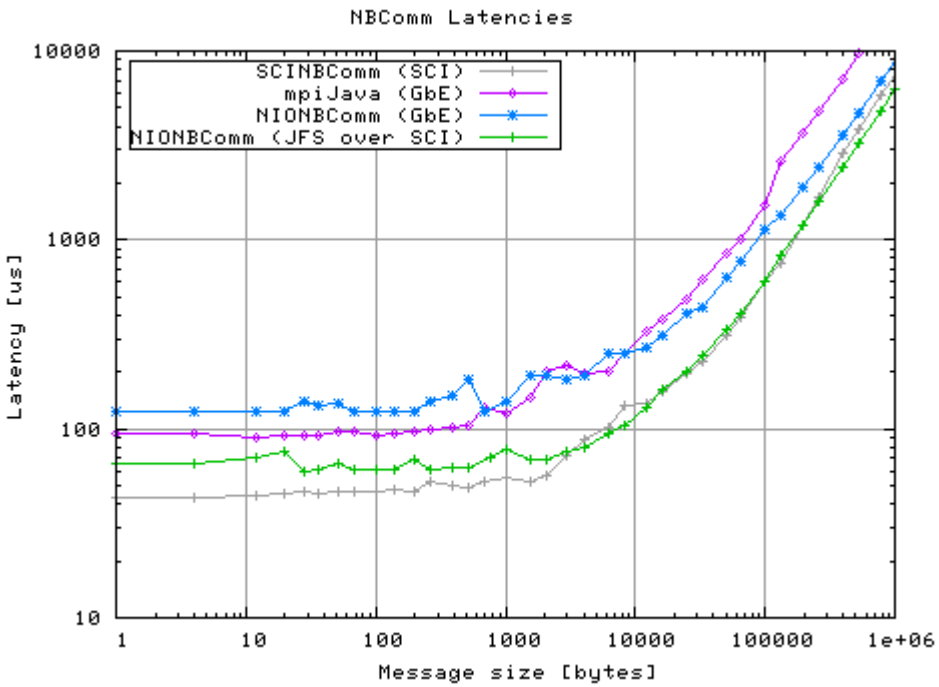


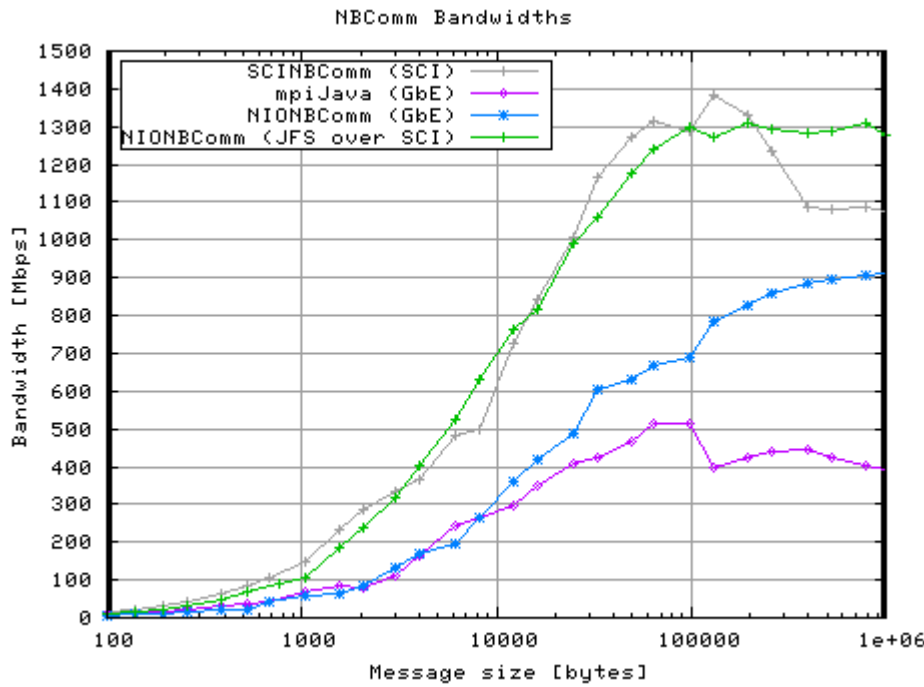
Performance Evaluation

- Experimental configuration:
  - PIV Xeon at 2.8 GHz 2GB mem (hyperthreading disabled)
  - SCI (Dolphin) and GbE (Marvell 88E8050)
  - Java: Sun JVM 1.5.0_05
  - gcc 3.4.4
  - Libraries:
    - mpiJava 1.2.5 over MPICH 1.2.5
    - SCI SOCKET 3.0.3 (Dolphin ICS)
    - DIS 3.0.3 (IRM/SISCI/SCILib/Mbox) (Dolphin ICS)
  - Linux CentOS 4 kernel 2.6.9

Performance Evaluation

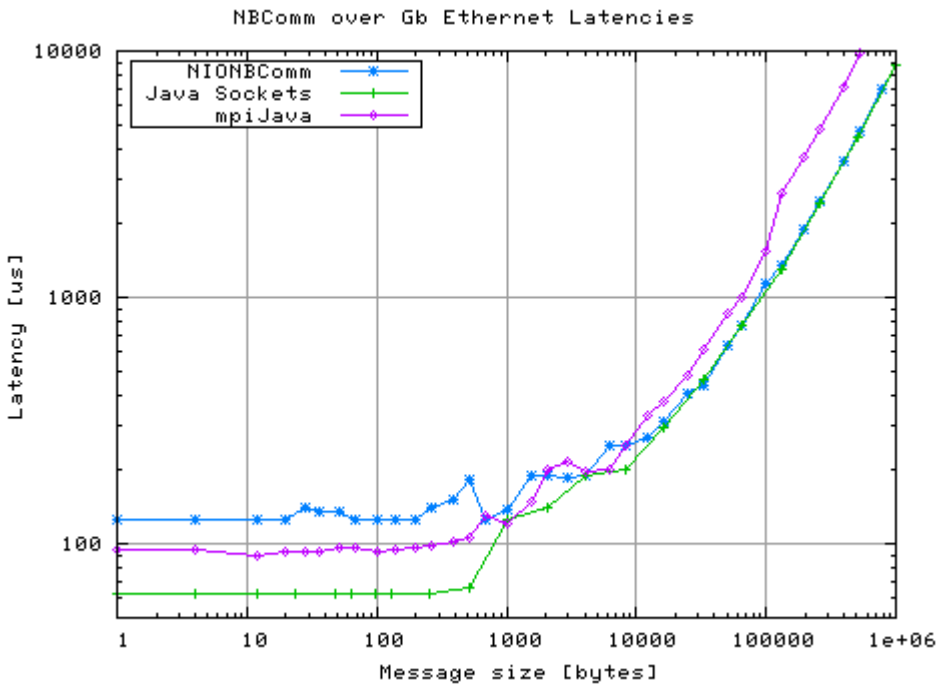
- NBComm implementations, native and 'pure' (Ping pong)

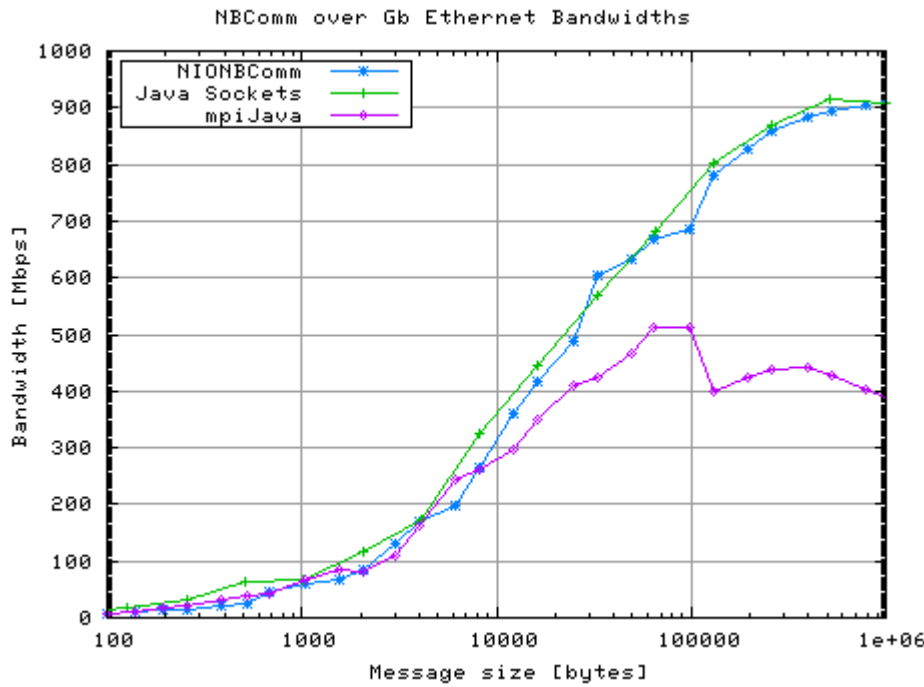




Performance Evaluation

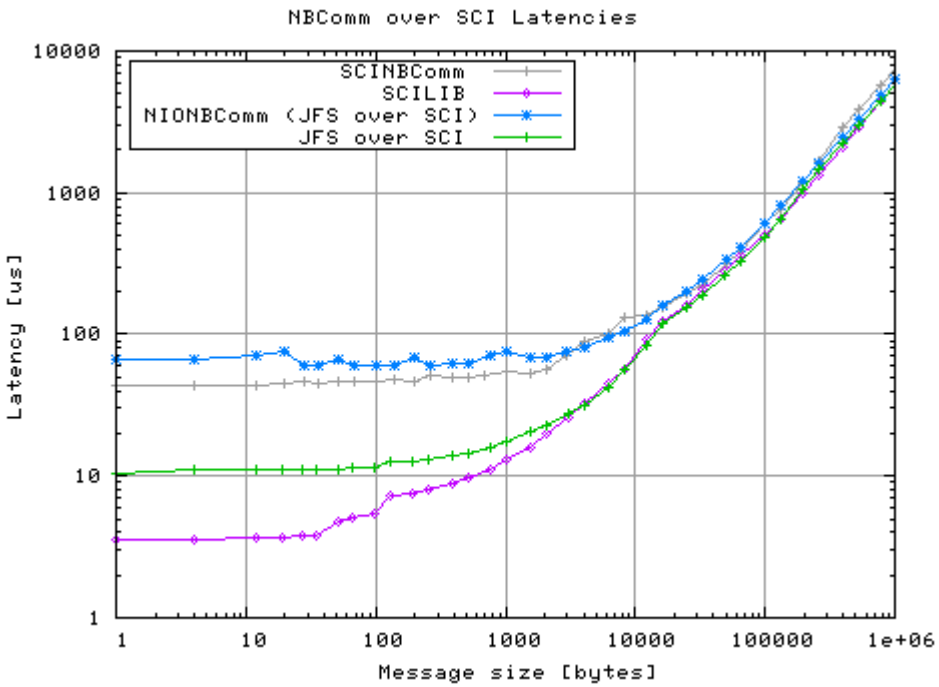
- **NIONBComm over GbE and underlying library**

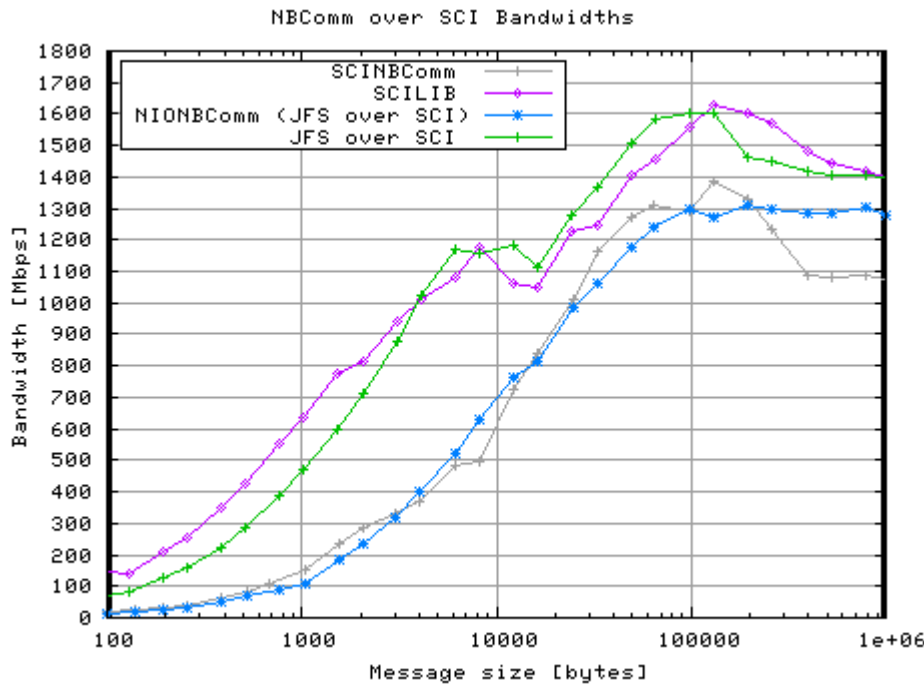




Performance Evaluation

- **NBComm over SCI and underlying libraries**





Performance Evaluation

- Overlaping computation/communication test
  - It has been experimentally obtained with 1Kb messages this overlapping:
    - SCINBComm 37%
    - NIONBComm 6%
    - mpiJava 40%
- Overlaping communication/communication test
  - It has been experimentally obtained with 8 simultaneous sendings this overlapping:
    - SCINBComm 44%
    - NIONBComm 33%
    - mpiJava 49%
- Latency reduction. Experimental example:
  - A 64Kb SCINBComm message, with *irecv()* posted after the message actual reception
    - Latency reduction: 49% in sender, non-blocking send (no wait)
    - Receiver node reduces the latency by 0.303 ms (≈50% global reduction)

Conclusions

- Java communications on clusters can significantly improve and increase their performance (up to ±50% latency reduction) thanks to the use of non-blocking communications
  - The implementation of efficent Java communication libraries on clusters poses several issues that must be addressed:
    - Message reception
    - Message arrival notification
    - Data movement efficiency
    - The definition of a suitable API
- It has been implemented *NBComm,* a Java library that provides with non-blocking communications and high performance cluster networks support
  - A native implementation, SCINBComm, in order to take advantage of native libraries on SCI
  - A "pure" Java implementation, NIONBComm, based on Java NIO
- Java Fast Sockets (JFS), a High Performance Java Sockets implementation supports System Area Networks through native code, delivering high performance to "pure" Java libraries and applications

Non-blocking Java Communications Support on Clusters

Guillermo L. Taboada*, Juan Touriño, Ramón Doallo

computer architecture group

HAC LUCE

UNIVERSIDADE DA CORUÑA
SPAIN

{taboada,juan,doallo}@udc.es

13th European PVM/MPI Users's Meeting (EuroPVM/MPI'06) Bonn