Java for High Performance Computing: Assessment of Current Research and Practice

Guillermo L. Taboada*, Juan Touriño, Ramón Doallo

Computer Architecture Group
University of A Coruña (Spain)
{taboada,juan,doallo}@udc.es

7th Intl. Conf. Principles and Practice of Programming in Java (PPPJ’09), University of Calgary, Alberta, Canada
Outline

1. Motivation
2. Java for High Performance Computing
3. Java for HPC: Current Research
4. Performance Evaluation
5. Conclusions
Java is an Alternative for HPC in the Multi-core Era

Interesting features:
- Built-in networking
- Built-in multi-threading
- Portable, platform independent
- Object Oriented
- Main training language

Many productive parallel/distributed programming libs:
- Java shared memory programming (high level facilities: Concurrency framework)
- Java Sockets
- Java RMI
- Message-Passing in Java (MPJ) libraries
HPC developers and users usually want to use Java in their projects.

Java code is no longer slow (Just-In-Time compilation)!

But still performance penalties in Java communications:

Pros and Cons:

- high programming productivity.
- but they are highly concerned about performance.
Java Adoption in HPC

- HPC developers and users usually want to use Java in their projects.
- Java code is no longer slow (Just-In-Time compilation)!
- But still performance penalties in Java communications:

**JIT Performance:**
- Like native performance.
- Java can even outperform native languages thanks to the dynamic compilation.
HPC developers and users usually want to use Java in their projects.

Java code is no longer slow (Just-In-Time compilation)!

But still performance penalties in Java communications:

- Poor high-speed networks support.
- The data copies between the Java heap and native code through JNI.
- Costly data serialization.
- The use of communication protocols unsuitable for HPC.
Emerging Interest in Java for HPC

Current State of Java for HPC

At the last javaOne I did a walk-on talk during the AMD keynote where I talked about how incredible HotSpot’s performance had become - beating the best C compilers. I ended my talk with a joking comment that “the next target is Fortran”. Afterwards, Denis Caromel of INRIA came up to me and said “you’re already there”. He and some colleagues had been working on some comparisons between Java and Fortran for HPC. Their final report Current State of Java for HPC has been made available as a tech report and makes pretty interesting reading. There are a lot of HPC micro benchmarks in it which look great. Thanks! Permalink Comments [3]
Current State of Java for HPC

Brian Amedro (OASIS)\textsuperscript{1}, Vladimir Bodnartchouk (\textsuperscript{2}), Denis Caromel\textsuperscript{1,3}, Christian Delbe\textsuperscript{1}, Fabrice Huet\textsuperscript{2} (OASIS)\textsuperscript{4}, Guillermo L. Taboada\textsuperscript{5} (OASIS)\textsuperscript{4} (2008)

Abstract: About ten years after the Java Grande effort, this paper aims at providing a snapshot of the current status of Java for High Performance Computing. Multi-core chips are becoming mainstream, offering many ways for a Java Virtual Machine (JVM) to take advantage of such systems for critical tasks such as Just-In-Time compilation or Garbage Collection. We first perform some micro benchmarks for various JVMs, showing the overall good performance for basic arithmetic operations. Then we study a Java implementation of the Nas Parallel Benchmarks, using the ProActive middleware for distribution. Comparing this implementation with a Fortran/MPI one, we show that they have similar performance on computation intensive benchmarks, but still have scalability issues when performing intensive communications. Using experiments on clusters and multi-core machines, we show that the performance varies greatly, depending on the Java Virtual Machine used (version and vendor) and the kind of computation performed.

\textsuperscript{1} Faculty of Informatics, University of A Coruña, Spain
\textsuperscript{2} OASIS (INRIA Sophia Antipolis / Laboratoire I3S)
\textsuperscript{3} INRIA - Université de Nice Sophia Antipolis - CNRS - UMR6599
\textsuperscript{4} ActiveEon
\textsuperscript{5} SLOOCH (INRIA Sophia Antipolis)
\textsuperscript{5} IRGA
\textsuperscript{4} University of A Coruña - Computer Architecture Group
University of A Coruña

G. L. Taboada\textsuperscript{*}, J. Touriño, R. Doallo
Current options in Java for HPC:

- Java Shared Memory Programming
- Java Sockets
- Java RMI
- Message-Passing in Java (MPJ)
Java for HPC

Java Shared Memory Programming:

- Java Threads
- Concurrency Framework (ThreadPools, Tasks ...)
- Parallel Java (PJ)
- Java OpenMP (JOMP and JaMP)
Listing 1: JOMP example

```java
public static void main(String argv[]) {
    int myid;
    //omp parallel private(myid)
    {
        myid = OMP.getThreadNum();
        System.out.println('Hello from' + myid);
    }

    //omp parallel for
    for (i=1;i<n;i++) {
        b[i] = (a[i] + a[i-1]) * 0.5;
    }
}
```
Java Communication Libraries Overview

Java HPC Applications

Java Message-passing libraries

Java RMI / Low-level messaging libraries

Java Sockets libraries

HPC Communications Hardware
Performance of current HPC networks (Theoretical/C/Java):

<table>
<thead>
<tr>
<th></th>
<th>Startup latency (microseconds)</th>
<th>Bandwidth (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gig. Ethernet</td>
<td>50/55/60</td>
<td>1000/920/900</td>
</tr>
<tr>
<td>10G Ethernet</td>
<td>5/10/50</td>
<td>10000/9000/5000</td>
</tr>
<tr>
<td>10G Myrinet</td>
<td>1/2/30</td>
<td>10000/9300/4000</td>
</tr>
<tr>
<td>InfiniBand</td>
<td>1/2/20</td>
<td>16000/12000/6000</td>
</tr>
<tr>
<td>SCI</td>
<td>1.4/3/50</td>
<td>5333/2400/800</td>
</tr>
</tbody>
</table>

G. L. Taboada*, J. Touriño, R. Doallo
Java Sockets

Standard and widely extended low-level programming interface for networked communications.

Current implementations:

- IO sockets
- NIO sockets
- Ibis sockets
- Java Fast Sockets

Pros and Cons:

- easy to use.
- but only TCP/IP support.
- lack non-blocking communication.
- lack HPC tailoring.
Java Sockets

Standard and widely extended low-level programming interface for networked communications.

Current implementations:
- IO sockets
- NIO sockets
- Ibis sockets
- Java Fast Sockets

Pros and Cons:
- provides non-blocking communication.
- but only TCP/IP support.
- lack HPC tailoring.
- difficult use.
Java Sockets

Standard and widely extended low-level programming interface for networked communications.

Current implementations:

- IO sockets
- NIO sockets
- Ibis sockets
- Java Fast Sockets

Pros and Cons:

- easy to use.
- with Myrinet support.
- but lack non-blocking communication.
- lack HPC tailoring.
Java Sockets

Standard and widely extended low-level programming interface for networked communications.

Current implementations:
- IO sockets
- NIO sockets
- Ibis sockets
- Java Fast Sockets

Pros and Cons:
- easy to use.
- efficient high-speed networks support.
- efficient shared memory protocol.
- with HPC tailoring.
- but lack non-blocking support.
## Remote Method Invocation

### RMI (Remote Method Invocation)

- Widely extended
- RMI-based middleware (e.g., ProActive)
- RMI Optimizations:
  - KaRMI
  - Manta
  - Ibis RMI
  - Opt RMI
Message-passing is the main HPC programming model.

- Implementation approaches in Java message-passing libraries.
- RMI-based.
- Wrapping a native library (e.g., MPI libraries: OpenMPI, MPICH).
- Sockets-based.
Listing 2: MPJ example

```java
import mpi.*;

public class Hello {

    public static void main(String argv[]) {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();

        if (rank == 0) {
            String[] msg = new String[1];
            msg[0] = new String("Hello");
            MPI.COMM_WORLD.Send(msg, 0, 1, MPI.OBJECT, 1, 13);
        } else if (rank == 1) {
            String[] message = new String[1];
            MPI.COMM_WORLD.Recv(message, 0, 1, MPI.OBJECT, 0, 13);
            System.out.println(message[0]);
        }
        MPI.Finalize();
    }
}
```
## Java for High Performance Computing

### Java for HPC: Assessment of Current Research and Practice

<table>
<thead>
<tr>
<th>Pure Java Impl.</th>
<th>Socket Impl.</th>
<th>High-speed network support</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java IO</td>
<td>Java NIO</td>
<td>Myrinet</td>
</tr>
<tr>
<td>MPJava</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Jcluster</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Parallel Java</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mpiJava</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>P2P-MPI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MPJ Express</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MPJ/Ibis</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JMPI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F-MPJ</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

G. L. Taboada*, J. Touriño, R. Doallo

Java for HPC: Assessment of Current Research and Practice
Java Communication Libraries Overview

<table>
<thead>
<tr>
<th>Java HPC Applications (Develop Efficient Codes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Message-passing libraries (Scalable Algorithms)</td>
</tr>
<tr>
<td>Low-level messaging libraries (MPJ Devices)</td>
</tr>
<tr>
<td>Java Sockets libraries (Java Fast Sockets)</td>
</tr>
<tr>
<td>HPC Hardware</td>
</tr>
</tbody>
</table>

G. L. Taboada*, J. Touriño, R. Doallo
Java for HPC: Assessment of Current Research and Practice
Java Fast Sockets (JFS):

- Provides efficient high-speed cluster interconnects support (SCI, Myrinet and InfiniBand).
- Optimizes Java IO sockets, more popular and extended than NIO sockets.
- Avoids the need for primitive data type array serialization.
- Significantly reduces buffering and unnecessary copies.
- Implements an optimized shared memory protocol.
- It is user and application transparent, no source code modification is necessary to use JFS.
Java Fast Sockets (JFS):

- Provides efficient high-speed cluster interconnects support (SCI, Myrinet and InfiniBand).
- Optimizes Java IO sockets, more popular and extended than NIO sockets.
- Avoids the need for primitive data type array serialization.
- Significantly reduces buffering and unnecessary copies.
- Implements an optimized shared memory protocol.
- It is user and application transparent, no source code modification is necessary to use JFS.
Java Fast Sockets (JFS):

- Provides efficient high-speed cluster interconnects support (SCI, Myrinet and InfiniBand).
- Optimizes Java IO sockets, more popular and extended than NIO sockets.
- Avoids the need for primitive data type array serialization.
- Significantly reduces buffering and unnecessary copies.
- Implements an optimized shared memory protocol.
- It is user and application transparent, no source code modification is necessary to use JFS.
Java Fast Sockets (JFS):

- Provides efficient high-speed cluster interconnects support (SCI, Myrinet and InfiniBand).
- Optimizes Java IO sockets, more popular and extended than NIO sockets.
- Avoids the need for primitive data type array serialization.
- Significantly reduces buffering and unnecessary copies.
- Implements an optimized shared memory protocol.
- It is user and application transparent, no source code modification is necessary to use JFS.
Java Fast Sockets (JFS):

- Provides efficient high-speed cluster interconnects support (SCI, Myrinet and InfiniBand).
- Optimizes Java IO sockets, more popular and extended than NIO sockets.
- Avoids the need for primitive data type array serialization.
- Significantly reduces buffering and unnecessary copies.
- Implements an optimized shared memory protocol.
- It is user and application transparent, no source code modification is necessary to use JFS.
Java Fast Sockets (JFS):

- Provides efficient high-speed cluster interconnects support (SCI, Myrinet and InfiniBand).
- Optimizes Java IO sockets, more popular and extended than NIO sockets.
- Avoids the need for primitive data type array serialization.
- Significantly reduces buffering and unnecessary copies.
- Implements an optimized shared memory protocol.
- It is user and application transparent, no source code modification is necessary to use JFS.
JFS Transparency

```java
SocketImplFactory factory = new jfs.net.JFSImplFactory();
Socket.setSocketImplFactory(factory);
ServerSocket.setSocketFactory(factory);

Class cl = Class.forName(className);
Method method = cl.getMethod("main", parameterTypes);
method.invoke(null, parameters);
```
JFS optimized protocol

SenderApplication

\[ \text{<primitive data type> sdata[ ]} \]

\[ \text{GetPrimitiveArrayCritical(sdata)} \]

\[ \text{Is dst local?} \]

\[ \begin{array}{c}
\text{Y} \\
\text{N}
\end{array} \]

\[ \begin{array}{c}
\text{RDMA?} \\
\text{Y} \\
\text{N}
\end{array} \]

\[ \text{Copy} \]

\[ \text{Native Socket Library} \]

\[ \text{native socket buffer} \]

\[ \text{JFS} \]

\[ \text{GetPrimitiveArrayCritical(rdata)} \]

\[ \text{Is src local?} \]

\[ \begin{array}{c}
\text{Y} \\
\text{N}
\end{array} \]

\[ \begin{array}{c}
\text{RDMA?} \\
\text{Y} \\
\text{N}
\end{array} \]

\[ \text{Copy} \]

\[ \text{Native Socket Library} \]

\[ \text{native socket buffer} \]

\[ \text{Network Communication} \]

ReceiverApplication

\[ \text{<primitive data type> rdata[ ]} \]
JFS extended API for communicating primitive data type arrays directly.

```java
jfs.net.SocketOutputStream.write(byte buf[], int offset, int length);
jfs.net.SocketOutputStream.write(int buf[], int offset, int length);
jfs.net.SocketOutputStream.write(double buf[], int offset, int length);
...
jfs.net.SocketInputStream.read(byte buf[], int offset, int length);
jfs.net.SocketInputStream.read(int buf[], int offset, int length);
jfs.net.SocketInputStream.read(double buf[], int offset, int length);
...
```
```java
int int_array[] = new int[20];

// Writing the first ten elements of int_array
if (os instanceof jfs.net.SocketOutputStream) {
    ((jfs.net.SocketOutputStream) os).write(int_array, 0, 10);
} else {
    int[] ints = (int[]) Array.newInstance(int.class, 10);
    System.arraycopy(int_array, 0, ints, 0, 10);
    oos = new ObjectOutputStream(os);
    oos.writeUnshared(ints);
}
```
JFS High-speed Networks Support

Parallel and Distributed Java Applications

Java Communication Middleware
(RMI-based, Socket-based or MPJ Middleware)

Java IO sockets
JFS
JVM IO sockets

JNI

UNIX Sockets
TCP/IP Sockets
Gigabit Ethernet Driver
Gigabit Ethernet NIC

SCIP
SCI Sockets/SCILib
SCI NIC

IPoMX
Sockets–MX
Myrinet Driver: MXoM
Myrinet NIC

IPoIB
SDP
Infiniband Driver: OFED
Infiniband NIC

Figure: Java communication middleware on high-speed multi-core clusters
JFS Micro-Benchmarking

JFS performance improvement compared to Sun JVM sockets

<table>
<thead>
<tr>
<th></th>
<th>JFS start-up reduction</th>
<th>JFS bandwidth increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCI</td>
<td>up to 88%</td>
<td>up to 1305%</td>
</tr>
<tr>
<td>Myrinet</td>
<td>up to 78%</td>
<td>up to 412%</td>
</tr>
<tr>
<td>InfiniBand</td>
<td>up to 65%</td>
<td>up to 860%</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>up to 10%</td>
<td>up to 119%</td>
</tr>
<tr>
<td>Shared memory</td>
<td>up to 50%</td>
<td>up to 4411%</td>
</tr>
</tbody>
</table>

G. L. Taboada*, J. Touriño, R. Doallo
Java for HPC: Assessment of Current Research and Practice
The use of pluggable low-level communication devices is widely extended in message-passing libraries.

**Message-passing Low-level Devices:**

- **MPICH/MPICH2 ADI/ADI3** (GM/MX for Myrinet, IBV/VAPI for InfiniBand, and shared memory).
- OpenMPI BTL (GM/MX for Myrinet, IBV/VAPI for InfiniBand, and shared memory).
- MPJ Express xdev (NIO sockets, MX for Myrinet, and shared memory).
The use of pluggable low-level communication devices is widely extended in message-passing libraries.

**Message-passing Low-level Devices:**

- MPICH/MPICH2 ADI/ADI3 (GM/MX for Myrinet, IBV/VAI for InfiniBand, and shared memory).
- OpenMPI BTL (GM/MX for Myrinet, IBV/VAI for InfiniBand, and shared memory).
- MPJ Express xdev (NIO sockets, MX for Myrinet, and shared memory).
The use of pluggable low-level communication devices is widely extended in message-passing libraries.

Message-passing Low-level Devices:

- MPICH/MPICH2 ADI/ADI3 (GM/MX for Myrinet, IBV/VAPI for InfiniBand, and shared memory).
- OpenMPI BTL (GM/MX for Myrinet, IBV/VAPI for InfiniBand, and shared memory).
- MPJ Express xdev (NIO sockets, MX for Myrinet, and shared memory).
xxdev API. Public interface of the xxdev.Device class

```java
public abstract class Device {
    static public Device newInstance(String deviceImpl);
    public int[] init(String[] args);
    public int id();
    public void finish();

    public Request isend(Object buf, int dst, int tag);
    public Request irecv(Object buf, int src, int tag, Status stts);

    public void send(Object buf, int dst, int tag);
    public Status recv(Object buf, int src, int tag);

    public Request issend(Object buf, int dst, int tag);
    public void ssend(Object buf, int dst, int tag);

    public Status iprobe(int src, int tag, int context);
    public Status probe(int src, int tag, int context);
    public Request peek();
}
```
**Motivation**

Java for High Performance Computing
Java for HPC: Current Research
Performance Evaluation
Conclusions

**JFS**
Java Communication Devices
MPJ Collectives Scalability
HPC Benchmarking

---

**Low-Level Java Communication Devices**

**xdev implementations**

- **Current**: niodev (Java NIO sockets), iodev (Java IO sockets, and hence JFS) and mxdev (Myrinet)
- **Ongoing**: smpdev (Shared memory) and ibdev (InfiniBand)
Low-Level Java Communication Devices

- **Current**: niodev (Java NIO sockets), iodev (Java IO sockets, and hence JFS) and mxdev (Myrinet)
- **Ongoing**: smpdev (Shared memory) and ibdev (InfiniBand)
Fast MPJ (F-MPJ) is the scalable and efficient Java message-passing library implemented on top of the low-level message-passing middleware iodev.

F-MPJ:

- shows efficient non-blocking communication (iodev) and high-speed multi-core clusters support (JFS).
- presents lower communication overhead through an extensive use of communications overlapping.
- achieves high scalability as it implements several algorithms per collective primitive, allowing their selection at runtime.
Fast MPJ (F-MPJ) is the scalable and efficient Java message-passing library implemented on top of the low-level message-passing middleware iodev.

F-MPJ:

- shows efficient non-blocking communication (iodev) and high-speed multi-core clusters support (JFS).
- presents lower communication overhead through an extensive use of communications overlapping.
- achieves high scalability as it implements several algorithms per collective primitive, allowing their selection at runtime.
Fast MPJ (F-MPJ) is the scalable and efficient Java message-passing library implemented on top of the low-level message-passing middleware iodev.

**F-MPJ:**
- shows efficient non-blocking communication (iodev) and high-speed multi-core clusters support (JFS).
- presents lower communication overhead through an extensive use of communications overlapping.
- achieves high scalability as it implements several algorithms per collective primitive, allowing their selection at runtime.
The design, implementation and runtime selection of efficient collective communication operations have been extensively discussed in the context of native message-passing libraries, but not in MPJ.

F-MPJ focuses on developing scalable MPJ collective primitives.
The design, implementation and runtime selection of efficient collective communication operations have been extensively discussed in the context of native message-passing libraries, but not in MPJ. F-MPJ focuses on developing scalable MPJ collective primitives.

Collective Algorithms:

- Flat Tree (FT)
- Minimum-Spanning Tree (MST)
- Binomial Tree (BT)
- Four-ary Tree (Four-aryT)
- Bucket (BKT) or cyclic
- BiDirectional Exchange (BDE) or recursive doubling
MPJ Collective Algorithms. MST

(a) Initial state

(b) $1^{st}$ Step

(c) $2^{nd}$ Step

(d) Final state

Figure: Minimum-spanning tree algorithm for Broadcast
MPJ Collective Algorithms. **BKT**

**Figure:** Bucket algorithm for Allgather (BKTAllgather)
**Figure:** Bidirectional exchange algorithm for Allgather (BDEAllgather). In the 2nd step, bidirectional exchanges occur between the two pairs of processes $p_0$ and $p_2$, and $p_1$ and $p_3$.
<table>
<thead>
<tr>
<th>Collective</th>
<th>F-MPJ</th>
<th>MPJ Express</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier</td>
<td>MST</td>
<td>nbFTGather + bFour-aryTBcast</td>
</tr>
<tr>
<td>Bcast</td>
<td>MST(^1)</td>
<td>bFour-aryT</td>
</tr>
<tr>
<td></td>
<td>MSTScatter + BKTAllgather(^2)</td>
<td></td>
</tr>
<tr>
<td>Scatter</td>
<td>MST(^1)</td>
<td>nbFT</td>
</tr>
<tr>
<td></td>
<td>nbFT(^2)</td>
<td></td>
</tr>
<tr>
<td>Scatterv</td>
<td>MST(^1)</td>
<td>nbFT</td>
</tr>
<tr>
<td></td>
<td>nbFT(^2)</td>
<td></td>
</tr>
<tr>
<td>Gather</td>
<td>MST(^1)</td>
<td>nbFT</td>
</tr>
<tr>
<td></td>
<td>nbFT(^2)</td>
<td></td>
</tr>
<tr>
<td>Gatherv</td>
<td>MST(^1)</td>
<td>nbFT</td>
</tr>
<tr>
<td></td>
<td>nbFT(^2)</td>
<td></td>
</tr>
<tr>
<td>Allgather</td>
<td>MSTGather + MSTBcast(^1) BKT(^2)/ BDE(^3)</td>
<td>nbFT</td>
</tr>
<tr>
<td>Allgatherv</td>
<td>MSTGatherv + MSTBcast</td>
<td>nbFT</td>
</tr>
<tr>
<td>Alltoall</td>
<td>nbFT</td>
<td>nbFT</td>
</tr>
<tr>
<td>Alltoallv</td>
<td>nbFT</td>
<td>nbFT</td>
</tr>
<tr>
<td>Reduce</td>
<td>MST(^1)</td>
<td>bFT</td>
</tr>
<tr>
<td></td>
<td>BKTReduce_scatter + MSTGather(^2)</td>
<td></td>
</tr>
<tr>
<td>Allreduce</td>
<td>MSTReduce + MSTBcast(^1) BKTReduce_scatter + BKTAllgather(^2)/ BDE(^3)</td>
<td>BT</td>
</tr>
<tr>
<td>Reduce_ -</td>
<td>MSTReduce + MSTScatterv(^1) BKT(^2)/ BDE(^3)</td>
<td>bFTReduce + nbFTScatterv</td>
</tr>
<tr>
<td>scatter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scan</td>
<td>nbFT</td>
<td>nbFT</td>
</tr>
</tbody>
</table>
## NPB-MPJ Characteristics (10,000 SLOC (Source LOC))

<table>
<thead>
<tr>
<th>Name</th>
<th>Operation</th>
<th>SLOC</th>
<th>Communicat. intensiveness</th>
<th>Kernel</th>
<th>Applic.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>Conjugate Gradient</td>
<td>1000</td>
<td>Medium</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>EP</td>
<td>Embarrassingly Parallel</td>
<td>350</td>
<td>Low</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>Fourier Transformation</td>
<td>1700</td>
<td>High</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>IS</td>
<td>Integer Sort</td>
<td>700</td>
<td>High</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td>Multi-Grid</td>
<td>2000</td>
<td>High</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SP</td>
<td>Scalar Pentadiagonal</td>
<td>4300</td>
<td>Medium</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
NPB-MPJ Optimization:

- JVM JIT compilation of heavy and frequent methods with runtime information
- Structured programming is the best option
  - Small frequent methods are better.
    - mapping elements from multidimensional to one-dimensional arrays (array flattening technique:
      \[ \text{arr3D}[x][y][z] \rightarrow \text{arr3D}[\text{pos3D}(\text{lengthx}, \text{lengthy}, x, y, z)] \])
- NPB-MPJ code refactored, obtaining significant improvements (up to 2800% performance increase)
Experimental Configuration:

**Departmental cluster (8 nodes)**
- Intel Xeon 5060 dual dual-core CPU (4 cores with hyper-threading per node)
- 4 GB RAM
- InfiniBand network (16 Gbps)
- Linux, OFED-1.4, Intel MPI/C Compiler
- Sun JDK 1.6, ProActive, F-MPJ, MPJ Express, mpiJava

**24-core machine**
- Quad Intel Xeon 7450 hexa-core CPU (24 cores)
- 32 GB RAM
- Linux, Sun JDK 1.6, Intel Open Compiler
Experimental Results on One Core (relative perf.)

NPB Class B Performance on 1 core (Xeon 5060)

- MPI (GNU Comp.)
- MPI (Intel Comp.)
- MPJ (F-MPJ)
- ProActive
- Java Threads

Speedup Relative to MPI (GNU Comp.)

CG EP FT IS MG SP

G. L. Taboada*, J. Touriño, R. Doallo

Java for HPC: Assessment of Current Research and Practice
NPB-MPJ Performance

CG (Class B)

Number of Cores

MOPS

NPB–MPI
NPB–MPJ (mpiJava)
NPB–MPJ (MPJ Express)
NPB–MPJ (F–MPJ)
NPB–PA
NPB–OMP
NPB–JAV

G. L. Taboada*, J. Touriño, R. Doallo
Java for HPC: Assessment of Current Research and Practice
NPB-MPJ Performance

EP (Class B)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (MPJ Express)
- NPB-MPJ (F-MPJ)
- NPB-PA
- NPB-OMP

Number of Cores

MOPS

G. L. Taboada*, J. Touriño, R. Doallo
Java for HPC: Assessment of Current Research and Practice
NPB-MPJ Performance

FT (Class B)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (MPJ Express)
- NPB-MPJ (F-MPJ)
- NPB-PA
- NPB-OMP
- NPB-JAV

Number of Cores

MOPS
NPB-MPJ Performance

IS (Class B)

Number of Cores

MOPS

NPB-MPI
NPB-MPJ (mpiJava)
NPB-MPJ (MPJ Express)
NPB-MPJ (F-MPJ)
NPB-PA
NPB-OMP
NPB-JAV

G. L. Taboada*, J. Touriño, R. Doallo
Java for HPC: Assessment of Current Research and Practice
NPB-MPJ Performance

MG (Class B)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (MPJ Express)
- NPB-MPJ (F-MPJ)
- NPB-PA
- NPB-OMP
- NPB-JAV

Number of Cores

MOPS

G. L. Taboada*, J. Touriño, R. Doallo

Java for HPC: Assessment of Current Research and Practice
NPB-MPJ Performance

SP (Class B)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (MPJ Express)
- NPB-MPJ (F-MPJ)
- NPB-OMP
- NPB-JAV

Number of Cores

MOPS

G. L. Taboada*, J. Touriño, R. Doallo
Java for HPC: Assessment of Current Research and Practice
Finis Terrae Supercomputer Configuration

**Finis Terrae** (142 HP Integrity rx7640 nodes).

Hybrid shared/distributed memory (up to 8 cores per node and up to 32 nodes).
- 16 Montvale Itanium2 (IA64) cores at 1.6 GHz (used 8 cores per node).
- 128 GB RAM
- Interconnected via InfiniBand (16 Gbps)

**Finis Terrae Integrity Superdome**

Shared memory performance evaluation of up to 64 cores:
- 128 Montvale Itanium2 (IA64) cores at 1.6 GHz
- 1 TB RAM
NPB-MPJ Performance Evaluation (Finis Terrae)

CG (Class C)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (MPJ Express)
- NPB-MPJ (F-MPJ)
- NPB-OMP
- NPB-JAV

Number of Cores

Speedup

G. L. Taboada*, J. Touriño, R. Doallo

Java for HPC: Assessment of Current Research and Practice
NPB-MPJ Performance Evaluation (Finis Terrae)

EP (Class C)

- NPB−MPI
- NPB−MPJ (mpiJava)
- NPB−MPJ (MPJ Express)
- NPB−MPJ (F−MPJ)
- NPB−OMP

G. L. Taboada*, J. Touriño, R. Doallo
NPB-MPJ Performance Evaluation (Finis Terrae)

FT (Class C)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (MPJ Express)
- NPB-MPJ (F-MPJ)
- NPB-OMP
- NPB-JAV

Number of Cores

Speedup

1 2 4 8 16 32 64 128 256

G. L. Taboada*, J. Touriño, R. Doallo
Java for HPC: Assessment of Current Research and Practice
NPB-MPJ Performance Evaluation (Finis Terrae)

IS (Class C)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (MPJ Express)
- NPB-MPJ (F-MPJ)
- NPB-OMP
- NPB-JAV

Speedup

Number of Cores

1 2 4 8 16 32 64 128 256

[Graph showing performance evaluation of different Java-based MPI implementations for IS (Class C) across various core counts]
NPB-MPJ Performance Evaluation (Finis Terrae)

MG (Class C)

- NPB-MPI
- NPB-MPJ (mpiJava)
- NPB-MPJ (F-MPJ)
- NPB-OMP
- NPB-JAV

Number of Cores

Speedup
G. L. Taboada*, J. Touriño, R. Doallo

Java for HPC: Assessment of Current Research and Practice
Gadget Cosmological Simulation Project Webpage

GADGET - 2
A code for cosmological simulations of structure formation

General
- Description
- Features
- Authors and History
- Acknowledgments
- News

Software
- Download
- Requirements
- License
- Mailing List
- Change Log
- Examples

Documentation
- Code Paper
- Users Guide
- Code Reference

Publications
- Scientific Papers
- Pictures
- Movies
- Links
- Contact Address

- Millennium Simulation
- Colliding disk galaxies
- Hydrodynamical simulations of cosmic structure formation
- Merging galaxies with quasar feedback
- Constrained Realizations of the Local Universe
- High-resolution simulations of a cluster of galaxies

G. L. Taboada*, J. Touriño, R. Doallo

Java for HPC: Assessment of Current Research and Practice
Gadget Cosmological Simulation Speedup

Gadget2

- MPI (IBV)
- MPI (SHM)
- MPJ (iodev over Gig. Eth.)
- MPJ (iodev over InfiniBand)
- MPJ (iodev over InfiniBand)
- MPJ (smpdev)
- MPJ (iodev over Sh. Mem.)

Number of Cores

Speedup

G. L. Taboada*, J. Touriño, R. Doallo

Java for HPC: Assessment of Current Research and Practice
Summary

- Current state of Java for HPC (interesting/feasible alternative)
- Available programming models in Java for HPC:
  - Shared memory programming
  - Distributed memory programming
  - Distributed shared memory programming
- Active research on Java for HPC (>30 projects)
- ...but still not a mainstream language for HPC
- Adoption of Java for HPC:
  - It is an alternative for programming multi-core clusters (tradeoff some performance for appealing features)
  - Performance evaluations are highly important
  - Analysis of current projects (promotion of joint efforts)
Questions?

JAVA FOR HIGH PERFORMANCE COMPUTING:
ASSESSMENT OF CURRENT RESEARCH AND PRACTICE

PPPJ’09

Guillermo López Taboada
Computer Architecture Group, University of A Coruña


For Further Reading II


Opt RMI

RMI Layers:
- Transport Protocol Optimization.
- Serialization Overhead Reduction.
- Object Manipulation Improvements.

Optimization:
- High Performance Sockets Support (JFS).
- Reduction of Data Block Information.
Opt RMI

RMI Layers:
- Transport Protocol Optimization.
- Serialization Overhead Reduction.
- Object Manipulation Improvements.

Optimization:
- Native Array Serialization.
Opt RMI

RMI Layers:
- Transport Protocol Optimization.
- Serialization Overhead Reduction.
- Object Manipulation Improvements.

Optimization:
- Versioning Information Reduction.
- Class Annotation Reduction.
- Array Processing Improvements.