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

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

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:

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.

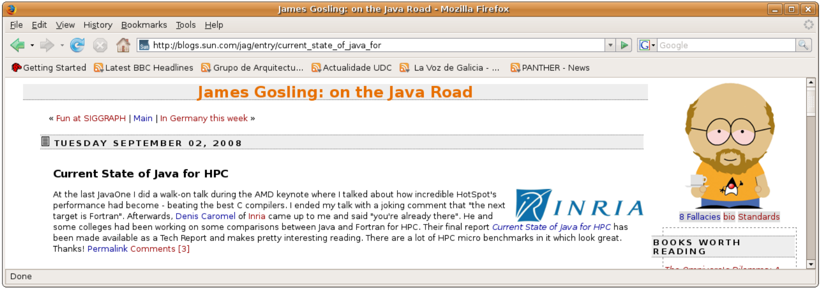
Java Adoption in HPC

- 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:

High Java Communications Overhead:

- 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

HAL - INRIA :: [inria-00312039, version 1] Current State of Java for HPC - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://hal.inria.fr/inria-00312039/en

Getting Started | Latest BBC Headlines | Grupo de Arquitectu... | Actualidade UDC | La Voz de Galicia - ... | PANTHER - News

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE — INRIA

Home | Submit | Browse | Search | Help | Services

english version | RSS

inria-00312039, version 1

See detailed view | BibTeX, EndNote,...

Current State of Java for HPC

Brian Amedro [1], Vladimir Bodnartchouk [2], Denis Caromel [1, 3], Christian Delbe [1], Fabrice Huet [1], Guillermo L. Taboada [a, 4]

(2008)

Access to full text: PDF

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.

a – Faculty of Informatics, University of A Coruna, Spain
1: OASIS (INRIA Sophia Antipolis / Laboratoire I3S)
INRIA – Université de Nice Sophia-Antipolis – CNRS : UMR6070
2: ActiveEon
ActiveEon
3: SLOOP (INRIA Sophia Antipolis)
INRIA
4: University of A Coruna - Computer Architecture Group
University of A Coruna

Done

Java for HPC

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)

JOMP

Listing 1: JOMP example

```
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

HPC Communications Hardware

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

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

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

Java Message-Passing Libraries

Message-passing is the main HPC programming model.

- Implementation approaches in Java message-passing libraries.

Implementation approaches

- RMI-based.
- Wrapping a native library (e.g., MPI libraries: OpenMPI, MPICH).
- Sockets-based.

Listing 2: MPJ example

```
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() ;
  }
}
```

	Pure Java Impl.	Socket impl.		High-speed network support			API		
		Java IO	Java NIO	Myrinet	InfiniBand	SCI	mpiJava 1.2	JGF MPJ	Other APIs
MPJava	✓		✓						✓
Jcluster	✓	✓							✓
Parallel Java	✓	✓							✓
mpiJava				✓	✓	✓	✓		
P2P-MPI	✓	✓	✓				✓		
MPJ Express	✓		✓	✓			✓		
MPJ/Ibis	✓	✓		✓				✓	
JMPI	✓	✓							✓
F-MPJ	✓	✓		✓	✓	✓	✓		

Java Communication Libraries Overview

Java HPC Applications (Develop Efficient Codes)
Java Message-passing libraries (Scalable Algorithms)
Low-level messaging libraries (MPJ Devices)
Java Sockets libraries (Java Fast Sockets)
HPC Hardware

Java Fast Sockets (JFS)

High Performance 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)

High Performance 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)

High Performance 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)

High Performance 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)

High Performance 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)

High Performance Java Fast Sockets (JFS):

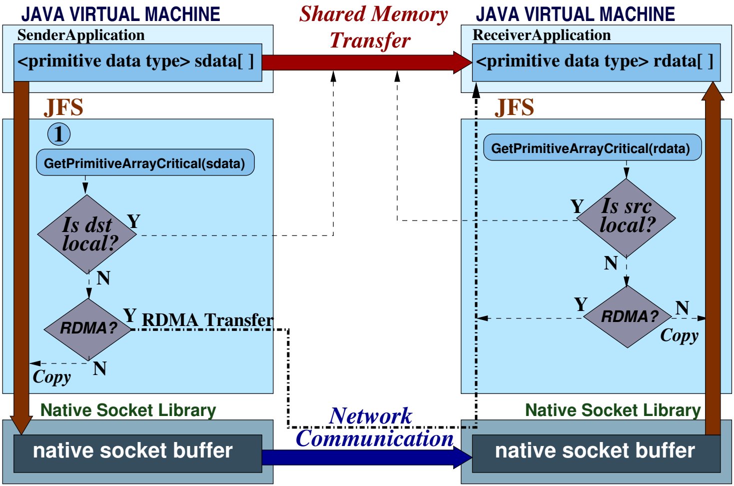
- Provides efficient high-speed cluster interconnects support (SCI, Myrinet and InfiniBand).
- Optimizes Java IO sockets, more popular and extended than NIO sockets.
- Avoids the need for primitive data type array serialization.
- Significantly reduces buffering and unnecessary copies.
- Implements an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

JFS Transparency

```
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



JFS Serialization Avoidance Feature

JFS extended API for communicating primitive data type arrays directly.

```
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);
  ...
```

JFS Portability and Performance (direct send of part of an integer array)

```
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

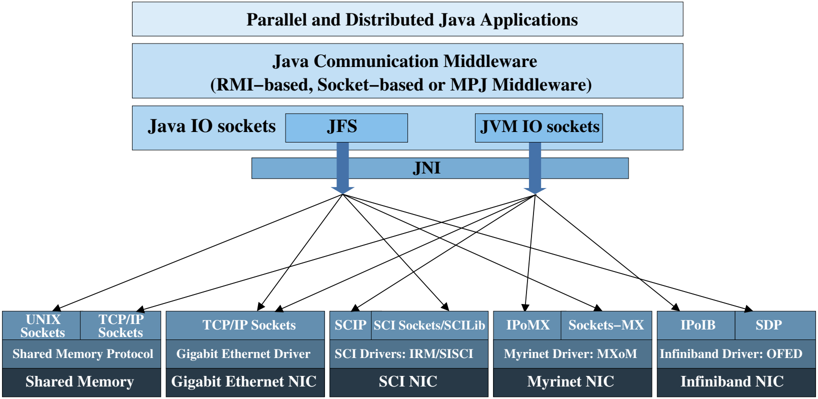


Figure: Java communication middleware on high-speed multi-core clusters

JFS Micro-Benchmarking

JFS performance improvement compared to Sun JVM sockets

	JFS start-up reduction	JFS bandwidth increase
SCI	up to 88%	up to 1305%
Myrinet	up to 78%	up to 412%
InfiniBand	up to 65%	up to 860%
Gigabit Ethernet	up to 10%	up to 119%
Shared memory	up to 50%	up to 4411%

iodev: Low-level Message-Passing Library

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).

iodev: Low-level Message-Passing Library

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).

iodev: Low-level Message-Passing Library

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

```
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();
}
```

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

xdev implementations

- 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)

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)

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)

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.

MPJ Collective Algorithms

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.

MPJ Collective Algorithms

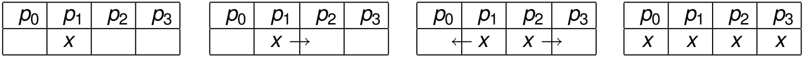
The design, implementation and runtime selection of efficient collective communication operations have been extensively discussed in the context of native message-passing libraries, but not in MPJ.

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

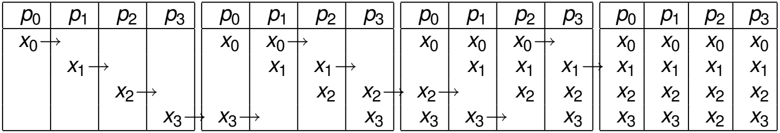
MPJ Collective Algorithms. MST



(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



(a) 1^{st} Step (b) 2^{nd} Step (c) 3^{rd} Step (d) Final state

Figure: Bucket algorithm for Allgather (BKTAllgather)

MPJ Collective Algorithms. BDE

p_0	p_1	p_2	p_3
x_0			
	x_1		
		x_2	
			x_3

(a) Initial state

p_0	p_1	p_2	p_3
$x_0 \rightarrow$			
	$\leftarrow x_1$		
		$x_2 \rightarrow$	
			$\leftarrow x_3$

(b) 1^{st} Step

p_0	p_1	p_2	p_3
x_0	$x_0 \rightrightarrows$		
x_1	$x_1 \rightrightarrows$		
		$\leftleftarrows x_2$	x_2
		$\leftleftarrows x_3$	x_3

(c) 2^{nd} Step

p_0	p_1	p_2	p_3
x_0	x_0	x_0	x_0
x_1	x_1	x_1	x_1
x_2	x_2	x_2	x_2
x_3	x_3	x_3	x_3

(d) Final state

Figure: Bidirectional exchange algorithm for Allgather (BDEAllgather). In the 2^{nd} step, bidirectional exchanges occur between the two pairs of processes p_0 and p_2, and p_1 and p_3

Collective	F-MPJ	MPJ Express
Barrier	MST	nbFTGather+ bFour-aryTBcast
Bcast	MST[1] MSTScatter+BKTAllgather[2]	bFour-aryT
Scatter	MST[1] nbFT[2]	nbFT
Scatterv	MST[1] nbFT[2]	nbFT
Gather	MST[1] nbFT[2]	nbFT
Gatherv	MST[1] nbFT[2]	nbFT
Allgather	MSTGather+MSTBcast[1] BKT[2] / BDE[3]	nbFT
Allgatherv	MSTGatherv+MSTBcast	nbFT
Alltoall	nbFT	nbFT
Alltoallv	nbFT	nbFT
Reduce	MST[1] BKTReduce_scatter+ MSTGather[2]	bFT
Allreduce	MSTReduce+MSTBcast[1] BKTReduce_scatter+ BKTAllgather[2] / BDE[3]	BT
Reduce_-scatter	MSTReduce+MSTScatterv[1] BKT[2] / BDE[3]	bFTReduce+ nbFTScatterv
Scan	nbFT	nbFT

NPB-MPJ Characteristics (10,000 SLOC (Source LOC))

Name	Operation	SLOC	Communicat. intensiveness	Kernel	Applic.
CG	Conjugate Gradient	1000	Medium	✓	
EP	Embarrassingly Parallel	350	Low	✓	
FT	Fourier Transformation	1700	High	✓	
IS	Integer Sort	700	High	✓	
MG	Multi-Grid	2000	High	✓	
SP	Scalar Pentadiagonal	4300	Medium		✓

NPB-MPJ

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: arr3D[x][y][z]→arr3D[pos3D(lenghtx,lengthy,x,y,z)])
  - NPB-MPJ code refactored, obtaining significant improvements (up to 2800% performance increase)

Experimental Configuration:

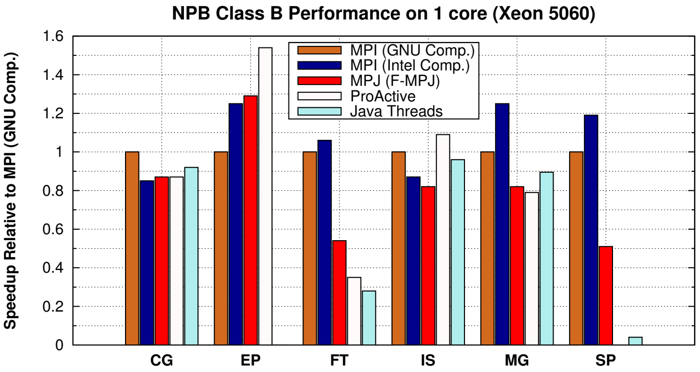
Departmental cluster (8 nodes)

- 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

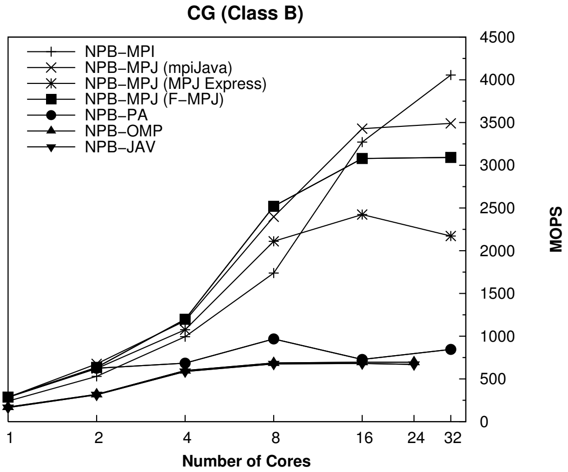
24-core machine

- Quad Intel Xeon 7450 hexa-core CPU (24 cores)
- 32 GB RAM
- Linux, Sun JDK 1.6, Intel Open Compiler

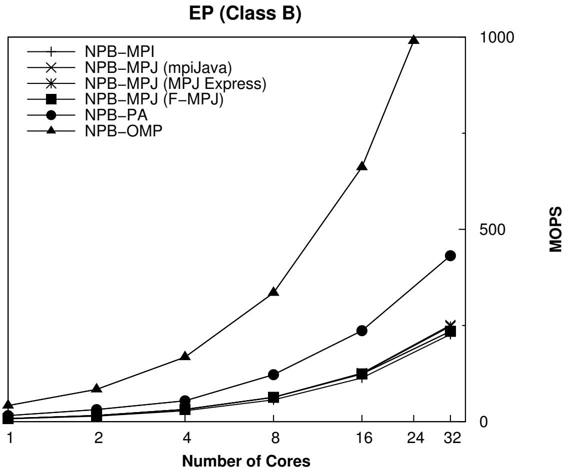
Experimental Results on One Core (relative perf.)



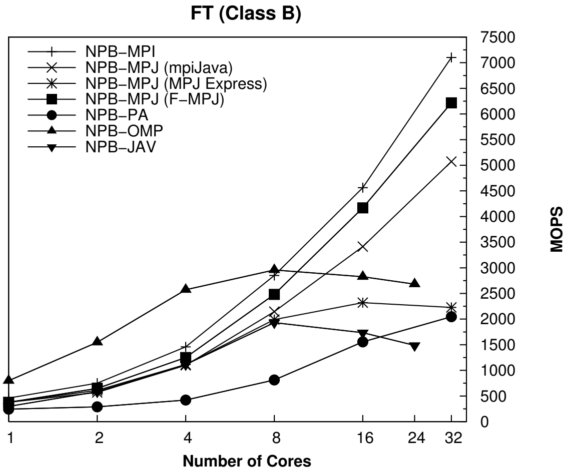
NPB-MPJ Performance



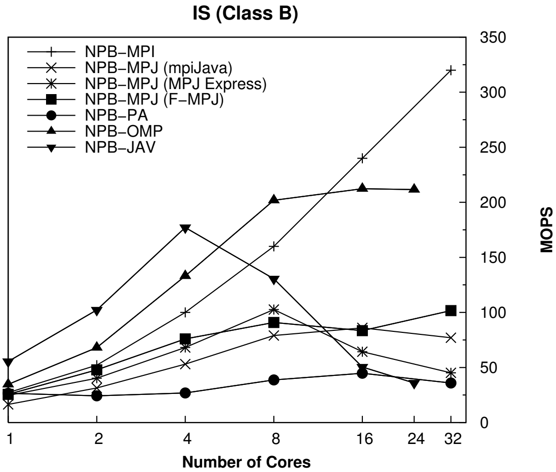
NPB-MPJ Performance



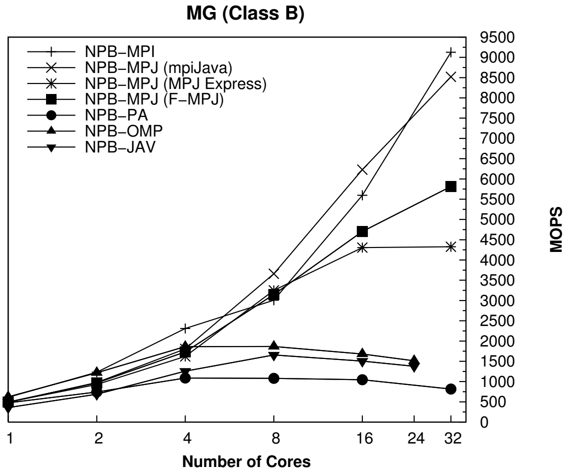
NPB-MPJ Performance



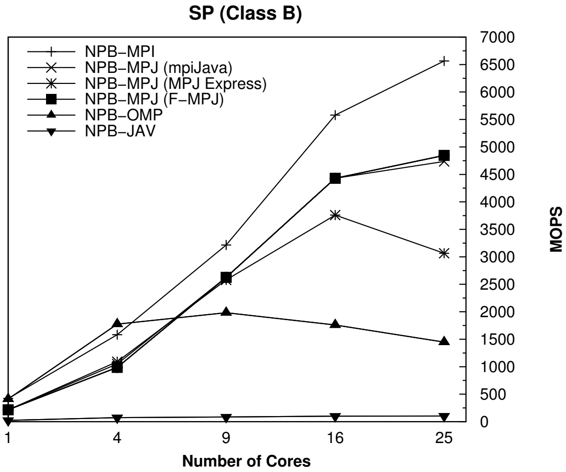
NPB-MPJ Performance



NPB-MPJ Performance



NPB-MPJ Performance



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).

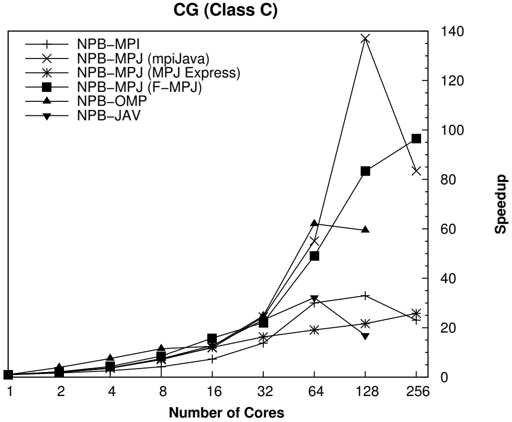
- 16 Montvale Itanium2 (IA64) cores at 1.6 GHz (used 8 cores per node).
- 128 GB RAM
- Interconnected via InfiniBand (16 Gbps)

Finis Terrae Integrity Superdome

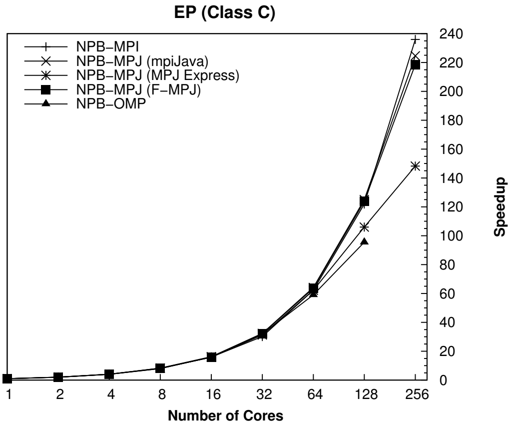
Shared memory performance evaluation of up to 64 cores:

- 128 Montvale Itanium2 (IA64) cores at 1.6 GHz
- 1 TB RAM

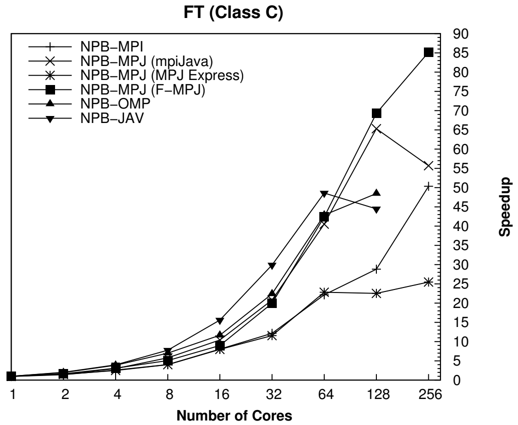
NPB-MPJ Performance Evaluation (Finis Terrae)



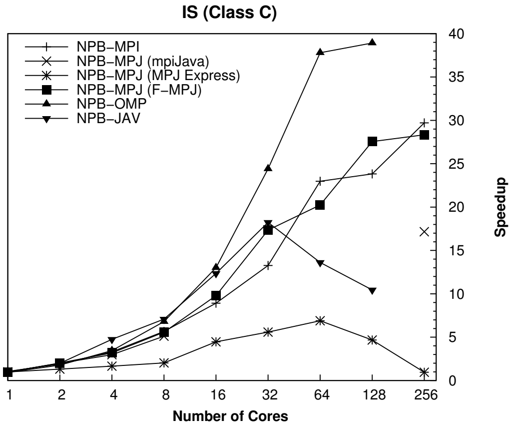
NPB-MPJ Performance Evaluation (Finis Terrae)



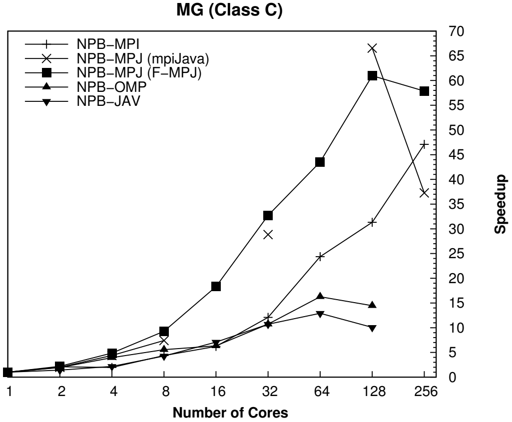
NPB-MPJ Performance Evaluation (Finis Terrae)



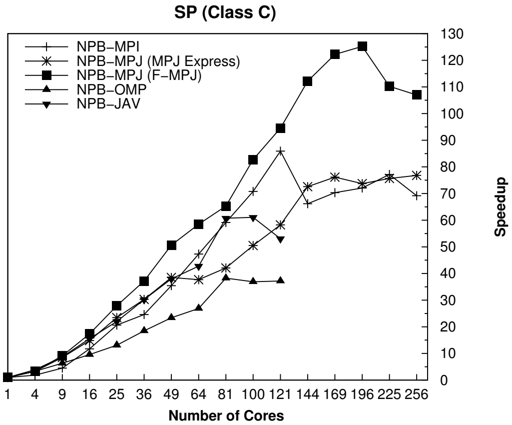
NPB-MPJ Performance Evaluation (Finis Terrae)



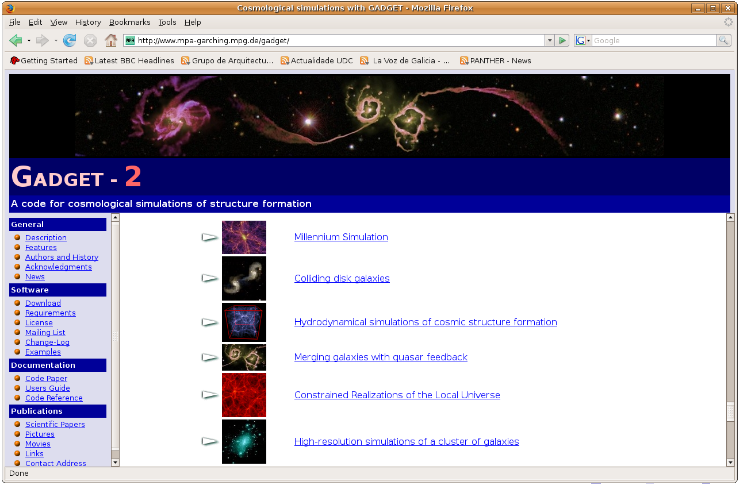
NPB-MPJ Performance Evaluation (Finis Terrae)



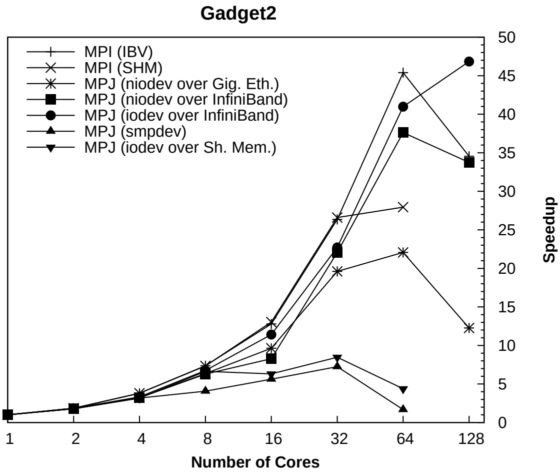
NPB-MPJ Performance Evaluation (Finis Terrae)



Gadget Cosmological Simulation Project Webpage



Gadget Cosmological Simulation Speedup



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 I

- G. L. Taboada, and J. Touriño, and R. Doallo, "Java Fast Sockets: Enabling High-speed Java Communications on High Performance Clusters," *Computer Communications*, vol. 31, no. 17, pp. 4049–4059, 2008.
- G. L. Taboada, J. Touriño, and R. Doallo, "F-MPJ: Scalable Java Message-passing Communications on Parallel Systems," *Journal of Supercomputing*, vol. In press, 2009.
- G. L. Taboada, J. Touriño, and R. Doallo, "Performance Analysis of Message-Passing Libraries on High-Speed Clusters," *Intl. Journal of Computer Systems Science & Engineering*, 2009 (In press).

For Further Reading II

- B. Amedro, V. Bodnartchouk, D. Caromel, C. Delbé, F. Huet, and G. L. Taboada. "Current State of Java for HPC", *INRIA Technical Report RT-0353*, pages 1–24, INRIA Sophia Antipolis, Nice, France, 2008, http://hal.inria.fr/inria-00312039/en/

- A. Shafi, B. Carpenter, and M. Baker. "Nested Parallelism for Multi-core HPC Systems using Java", *Journal of Parallel and Distributed Computing*, 2009 (In press).

- A. Shafi, B. Carpenter, M. Baker, and A. Hussain. "A Comparative Study of Java and C Performance in two Large-scale Parallel Applications", .*Concurrency and Computation: Practice and Experience*, In press, 2009.

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.

Gadget Cosmological Simulation Runtime