Java for High Performance Cloud Computing

Guillermo López Taboada

Computer Architecture Group
University of A Coruña (Spain)
taboada@udc.es

October 31st, 2012, Future Networks and Distributed Systems (FUNDS), Manchester Metropolitan University, Manchester (UK)
Outline

1. Java for High Performance Computing
   - Java Shared Memory Programming
   - Java Message-Passing
   - Java GPGPU
   - Development of Efficient HPC Benchmarks

2. High Performance Cloud Computing
   - AWS IaaS for HPC and Big Data

3. Performance Evaluation
   - Evaluation of Java HPC
   - Evaluation of HPC in the Cloud
   - Case study: ProtTest-HPC
Java is an Alternative for HPC in the Multi-core Era

C and Java are the most popular languages. Fortran, popular language in HPC, is the 27# (0.42%)
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
- Apache Hadoop
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.
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 (C, Fortran) 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

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

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/NPCI 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.
Current options in Java for HPC:

- Java Shared Memory Programming (popular)
- Java RMI (poor performance)
- Java Sockets (low level programming)
- Message-Passing in Java (MPJ) (extended, easy and reasonably good performance)
- Apache Hadoop (for High Throughput Computing)
Java for High Performance Computing

High Performance Cloud Computing

Performance Evaluation

Conclusions

Java Shared Memory Programming

Java Message-Passing

Java GPGPU

Development of Efficient HPC Benchmarks

Java for HPC

Java Shared Memory Programming:

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

class BasicThread extends Thread {
    // This method is called when the thread runs
    public void run() {
        for (int i = 0; i < 1000; i++)
            increaseCount();
    }
}

class Test {
    int counter = 0L;
    increaseCount() {
        counter++;
    }

    public static void main(String argv[]) {
        // Create and start the thread
        Thread t1 = new BasicThread();
        Thread t2 = new BasicThread();
        t1.start();
        t2.start();
        System.out.println("Counter=" + counter);
    }
}
Listing 2: Java Concurrency Framework highlights

class Test {

    public static void main (String argv[]) {

        // create the thread pool
        ThreadPool threadPool = new ThreadPool(numThreads);

        // run example tasks (tasks implement Runnable)
        for (int i = 0; i < numTasks; i++) {
            threadPool.runTask(createTask(i));
        }

        // close the pool and wait for all tasks to finish.
        threadPool.join();
    }

    // Another interesting classes from concurrency framework:
    // CyclicBarrier
    // ConcurrentHashMap
    // PriorityBlockingQueue
    // Executors ...
}
JOMP is the Java OpenMP binding

Listing 3: 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;
    }
}
```
MPJ is the Java binding of MPI

Listing 4: MPJ example

```java
import mpi.*;

public class Hello {

    public static void main(String argv[]) {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int msg_tag = 13;

        if (rank == 0) {
            int peer_process = 1;
            String[] msg = new String[1];
            msg[0] = new String("Hello");
            MPI.COMM_WORLD.Send(msg, 0, 1, MPI.OBJECT, peer_process, msg_tag);
        } else if (rank == 1) {
            int peer_process = 0;
            String[] msg = new String[1];
            mpi.COMM_WORLD.Recv(msg, 0, 1, MPI.OBJECT, peer_process, msg_tag);
            System.out.println(msg[0]);
        }
        MPI.Finalize();
    }
}
```

Guillermo López Taboada
## Java for High Performance Computing

### High Performance Cloud Computing

- Performance Evaluation
- Conclusions

### Java Shared Memory Programming
- Java Message-Passing
- Java GPGPU
- Development of Efficient HPC Benchmarks

<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>Myrinet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Java NIO</td>
<td>InfiniBand</td>
<td>SCI</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>

Guillermo López Taboada

Java for High Performance Cloud Computing
FastMPJ

MPJ implementation developed at the Computer Architecture Group of University of A Corunna.

Features of FastMPJ include:

- High performance intra-node communication
- Efficient RDMA transfers over InfiniBand and RoCE
- Fully portable, as Java
- Scalability up to thousands of cores
- Highly productive development and maintenance
- Ideal for multicore servers, cluster and cloud computing
FastMPJ runs on InfiniBand through IB Verbs, can rely on TCP/IP and has shared memory support through Java threads:

<table>
<thead>
<tr>
<th>MPJ Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPJ API (mpiJava 1.2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FastMPJ Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPJ Collective Primitives</td>
</tr>
<tr>
<td>MPJ Point-to-Point Primitives</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The xxdev layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxdev</td>
</tr>
<tr>
<td>psmdev</td>
</tr>
<tr>
<td>ibvdev</td>
</tr>
<tr>
<td>niodev/iodev</td>
</tr>
<tr>
<td>smdev</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java Native Interface (JNI)</th>
<th>Java Sockets</th>
<th>Java Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>MX/Open-MX</td>
<td>InfiniPath PSM</td>
<td>IBV API</td>
</tr>
<tr>
<td>Myrinet/Ethernet</td>
<td>InfiniBand</td>
<td>Ethernet</td>
</tr>
<tr>
<td></td>
<td>Shared Memory</td>
<td></td>
</tr>
</tbody>
</table>
### Table: Available solutions for GPGPU computing in Java

<table>
<thead>
<tr>
<th>Java bindings</th>
<th>User-friendly</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CUDA</strong></td>
<td></td>
</tr>
<tr>
<td>JCUDA</td>
<td>Java-GPU Rootbeer</td>
</tr>
<tr>
<td>jCuda</td>
<td></td>
</tr>
<tr>
<td><strong>OpenCL</strong></td>
<td></td>
</tr>
<tr>
<td>JOCL</td>
<td>Aparapi</td>
</tr>
<tr>
<td>JogAmp’s JOCL</td>
<td></td>
</tr>
</tbody>
</table>
## Table: Description of the GPU-based testbed

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>1 × Intel(R) Xeon hexacore X5650 @ 2.67GHz</td>
</tr>
<tr>
<td><strong>CPU performance</strong></td>
<td>64.08 GFLOPS DP (10.68 GFLOPS DP per core)</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>1 × NVIDIA Tesla “Fermi” M2050</td>
</tr>
<tr>
<td><strong>GPU performance</strong></td>
<td>515 GFLOPS DP</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>12 GB DDR3 (1333 MHz)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>Debian GNU/Linux, kernel 3.2.0-3</td>
</tr>
<tr>
<td><strong>CUDA</strong></td>
<td>version 4.2 SDK Toolkit</td>
</tr>
<tr>
<td><strong>JDK version</strong></td>
<td>OpenJDK 1.6.0_24</td>
</tr>
</tbody>
</table>
Figure: Matrix multiplication kernel performance (SHOC GEMM)
Figure: Stencil 2D kernel performance (SHOC Stencil2D)
**Java GPGPU**

---

**Figure:** Stencil 2D kernel performance (SHOC Stencil)
Java GPGPU

Figure: FFT kernel performance (SHOC FFT)
Java GPGPU

FFT performance (Single precision)

FFT performance (Double precision)

Figure: FFT kernel performance (SHOC FFT)
## 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>

*Guillermo López Taboada*
Java HPC optimization: NPB-MPJ success case

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: \( arr3D[x][y][z] \rightarrow arr3D[pos3D(lenghtx,lengthy,x,y,z)] \))
- NPB-MPJ code refactored, obtaining significant improvements (up to 2800% performance increase)
Experimental Results on One Core (relative perf.)

NPB Kernels Serial Performance on Amazon EC2 (C Class)

GNU compiler
Intel compiler
JVM

MOPS: Millions of Operations Per Second

CG (cc1.4xlarge) CG (cc2.8xlarge) FT (cc1.4xlarge) FT (cc2.8xlarge) IS (cc1.4xlarge) IS (cc2.8xlarge) MG (cc1.4xlarge) MG (cc2.8xlarge)
IaaS: Infrastructure-as-a-Service

The access to infrastructure supports the execution of computationally intensive tasks in the cloud. There are many IaaS providers:

- Amazon Web Services (AWS)
- Google Compute Engine (beta)
- IBM cloud
- HP cloud
- Rackspace
- ProfitBricks
- Penguin-On-Demand (POD)

IMHO, AWS is the most suitable in terms of available resources,
HPC and Big Data in AWS

AWS

- Provides a set of instances for HPC
  - CC1: dual socket, quadcore Nehalem Xeon (93 GFlops)
  - CG1: dual socket, quadcore Nehalem Xeon con 2 GPUs (1123 GFlops)
  - CC2: dual socket, octcore Sandy Bridge Xeon (333 GFlops)
- Up to 63 GB RAM
- 10 Gigabit Ethernet high performance network
**HPL Linpack in AWS (TOP500)**

*Table: Configuration of our EC2 AWS cluster*

<table>
<thead>
<tr>
<th><strong>CPU</strong></th>
<th>2 × Xeon quadcore <a href="mailto:X5570@2.93GHz">X5570@2.93GHz</a> (46.88 GFlops DP each)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EC2 Compute Units</strong></td>
<td>33.5</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>2 × NVIDIA Tesla “Fermi” M2050 (515 GFlops DP each)</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>22 GB DDR3</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>1690 GB</td>
</tr>
<tr>
<td><strong>Virtualization</strong></td>
<td>Xen HVM 64-bit platform (PV drivers for I/O)</td>
</tr>
<tr>
<td><strong>Number of CGI nodes</strong></td>
<td>32</td>
</tr>
<tr>
<td><strong>Interconnection network</strong></td>
<td>10 Gigabit Ethernet</td>
</tr>
<tr>
<td><strong>Total EC2 Compute Units</strong></td>
<td>1072</td>
</tr>
<tr>
<td><strong>Total CPU Cores</strong></td>
<td>256 (3 TFLOPS DP)</td>
</tr>
<tr>
<td><strong>Total GPUs</strong></td>
<td>64 (32.96 TFLOPS DP)</td>
</tr>
<tr>
<td><strong>Total FLOPS</strong></td>
<td>35.96 TFLOPS DP</td>
</tr>
</tbody>
</table>
HPL Linpack in AWS (TOP500)

- HPL Linpack ranks the performance of supercomputers in TOP500 list
- In AWS 14 TFlops are available for everybody! (67 USD$/h on demand, 21 USD$/h spot)

![HPL Performance on Amazon EC2](chart1)

![HPL Scalability on Amazon EC2](chart2)
HPL Linpack in AWS (TOP500)

References

- Finis Terrae (CESGA): 14,01 TFlops
- Our AWS Cluster: 14,23 TFlops
- AWS (06/11): 41,82 TFlops (# 451 TOP500 06/11)
- MareNostrum (BSC): 63,83 TFlops (# 299 TOP500)
- MinoTauro (BSC): 103,2 TFlops (# 114 TOP500)
- AWS (11/11): 240,1 TFlops (# 42 TOP500)
Radiography enhanced through Montecarlo (MC-GPU)
Image processing with Montecarlo (MC-GPU)

- Time: originally 187 minutes, in AWS with HPC 6 seconds!

- Cost: processing 500 radiographies in AWS: aprox. 70 USD$, 0.14 USD$ per unit

**MC-GPU Execution Time on Amazon EC2**

**MC-GPU Scalability on Amazon EC2**
AWS HPC and Big Data Remarks

HPC and Big Data in AWS

- High computational power (new Sandy Bridge processors and GPUs)
- High performance communications over 10 Gigabit Ethernet
- Efficient access to file network systems (e.g., NFS, OrangeFS)
- Without waiting times in accessing resources
- Easy applications and infrastructure deployment, ready-to-run applications in AMIs
### Java HPC in an InfiniBand cluster

#### Point-to-point Performance on DAS-4 (IB-QDR)

![Graph showing point-to-point performance on DAS-4 (IB-QDR)](image)

<table>
<thead>
<tr>
<th>Message size (bytes)</th>
<th>Latency (μs)</th>
<th>Bandwidth (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>4K</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>16K</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>64K</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>256K</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>16M</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

- MVAPICH2
- Open MPI
- FastMPJ (ibvdev)

**Guillermo López Taboada**

**Java for High Performance Computing**
Java HPC in an InfiniBand cluster

NPB CG Kernel Performance on DAS-4 (IB-QDR)

- MVAPICH2
- Open MPI
- FastMPJ (ibvdev)

Number of Cores vs MOPS graph
Java HPC in an InfiniBand cluster

NPB FT Kernel Performance on DAS-4 (IB-QDR)

- MVAPICH2
- Open MPI
- FastMPJ (ibvdev)

Number of Cores vs. MOPS for NPB FT Kernel Performance on DAS-4 (IB-QDR)
Java HPC in an InfiniBand cluster

NPB IS Kernel Performance on DAS-4 (IB-QDR)

- MVAPICH2
- Open MPI
- FastMPJ (ibvdev)

MOPS vs. Number of Cores
Java HPC in an InfiniBand cluster

NPB MG Kernel Performance on DAS-4 (IB-QDR)

- MVAPICH2
- Open MPI
- FastMPJ (ibvdev)

Number of Cores:
- 1
- 16
- 32
- 64
- 128
- 256
- 512

MOPS:
- 0
- 50000
- 100000
- 150000
- 200000
- 250000
- 300000
- 350000
- 400000
NPB Performance Evaluation in AWS

Point-to-point Performance on 10 GbE (cc1.4xlarge)

Latency (μs)

Bandwidth (Gbps)

Message size (bytes)
### NPB Performance Evaluation in AWS

#### Point-to-point Performance on 10 GbE (cc2.8xlarge)

<table>
<thead>
<tr>
<th>Message size (bytes)</th>
<th>Latency (µs)</th>
<th>Bandwidth (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>55</td>
<td>7</td>
</tr>
<tr>
<td>64</td>
<td>60</td>
<td>8</td>
</tr>
<tr>
<td>256</td>
<td>65</td>
<td>9</td>
</tr>
<tr>
<td>1K</td>
<td>70</td>
<td>10</td>
</tr>
<tr>
<td>4K</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>16K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Software Comparison:**
- MPICH2
- OpenMPI
- FastMPJ

**Message size (bytes):**
- 1 4 16 64 ... 4K 16K 64K 256K 1M 4M 16M 64M

**Bandwidth (Gbps):**
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
NPB Performance Evaluation in AWS

Point-to-point Performance on SHMEM (cc1.4xlarge)

- MPICH2
- OpenMPI
- FastMPJ

Guillermo López Taboada
NPB Performance Evaluation in AWS

Point-to-point Performance on SHMEM (cc2.8xlarge)

Latency (µs)

Bandwidth (Gbps)

MPICH2
OpenMPI
FastMPJ

Message size (bytes)
NPB Performance Evaluation in AWS

CG C Class Performance (cc1.4xlarge)

- MPICH2
- OpenMPI
- FastMPJ

MOPS vs. Number of Cores graph.
NPB Performance Evaluation in AWS

FT C Class Performance (cc1.4xlarge)

- MPICH2
- OpenMPI
- FastMPJ

```
Number of Cores

MOPS
```

Guillermo López Taboada
NPB Performance Evaluation in AWS

IS C Class Performance (cc1.4xlarge)

- MPICH2
- OpenMPI
- FastMPJ

Graph showing MOPS (Million Operations Per Second) vs. Number of Cores for different MPI implementations.
NPB Performance Evaluation in AWS

MG C Class Performance (cc1.4xlarge)

MOPS vs Number of Cores for MPICH2, OpenMPI, and FastMPJ.
NPB Performance Evaluation in AWS

CG C Class Performance (cc2.8xlarge)

MOPS

Number of Cores

MPICH2
OpenMPI
FastMPJ
NPB Performance Evaluation in AWS

FT C Class Performance (cc2.8xlarge)

MOPS vs Number of Cores

- MPICH2
- OpenMPI
- FastMPJ

Guillermo López Taboada
NPB Performance Evaluation in AWS

IS C Class Performance (cc2.8xlarge)

Number of Cores

MOPS

MPICH2
OpenMPI
FastMPJ

Guillermo López Taboada
Java for High Performance Cloud Computing
NPB Performance Evaluation in AWS

MG C Class Performance (cc2.8xlarge)

- MPICH2
- OpenMPI
- FastMPJ

Graph showing the performance of MG C class on AWS with different MP implementations.
HPC in AWS

- Efficient Cloud Computing support

CG Kernel OpenMPI Performance on Amazon EC2

- Default
- Default (Fillup)
- Default (Tuned)
- 16 Instances (Fillup)
- 16 Instances (Tuned)

CG Kernel OpenMPI Performance on Amazon EC2

- Default
- 16 Instances
- 32 Instances
- 64 Instances
- 128 Instances
- 256 Instances
- 512 Instances
ProtTest-HPC Phylogeny application

ProtTest is a Java application widely used in Phylogeny (4000 users)

- Implements 112 models to validate the maximum likelihood (ML) of the evolution of several taxa
- The input data are amino acid sequences from several taxa
- Base on Markov Chains / Monte Carlo simulations
- **Time consuming** sequential application. A fairly simple example can last weeks/months
- Relies on a C application for the computational intensive part of the application (ML optimization)
ProtTest-HPC Phylogeny application
ProtTest-HPC design

- **Estimators**
  - `RunEstimator`
  - `RunEstimatorImpl`
  - `ConcreteEstimator`

- **ProtTest-HPC**
  - `<<interface>> ProtTestFacade`
  - `ProtTestFacadeImpl`
  - `ProtTestFacadeSequential`
  - `ProtTestFacadeMPJ`
  - `ProtTestFacadeThread`

- **Façades**
  - `<<interface>> ProtTestFacade`
  - `ProtTestFacadeImpl`

- **Information Criteria**
  - `InformationCriterion`
  - `SelectionModel`
  - `AIC`
  - `AICSSelectionModel`

- **Models**
  - `AminoAcidModel`
  - `Model`
  - `Tree`
  - `TreeFacade`
  - `TreeFacadeImpl`

- **Strategies**
  - `Distributor`
  - `DynamicDistributionStrategy`
  - `StaticDistributionStrategy`
ProtTest-HPC Implementation (Shared Memory)
ProtTest-HPC Implementation (MPJ)
Figure: ProtTest-HPC shared memory (8-core machine)
ProtTest-HPC Performance

Figure: ProtTest-HPC shared memory (24-core machine)
ProtTest-HPC Performance

Figure: ProtTest-HPC distributed memory (256-core cluster)
ProtTest-HPC Performance

Figure: ProtTest-HPC hybrid shared/distributed memory (28 nodes)
ProtTest-HPC Performance

Figure: ProtTest-HPC hybrid shared/distributed memory (8 nodes)
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 in the cloud:
  - It is an alternative for the cloud (tradeoff some performance for appealing features)
  - Performance evaluations are highly important
  - Analysis of current projects (promotion of joint efforts)
Questions?

JAVA FOR HIGH PERFORMANCE CLOUD COMPUTING
FUNDS SEMINAR SERIES’12
Guillermo López Taboada
Email: taboada@udc.es
WWW: http://gac.udc.es/~gltaboada/
Computer Architecture Group, University of A Coruña