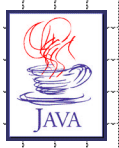
High Performance Java

Marco Ronchetti
Dipartimento di Informatica e Telecomunicazioni
Università di Trento

marco.ronchetti@dit.unitn.it

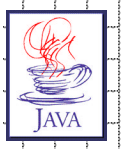


High Performance Java

Why Java?

What is Java?

Is Java a viable solution for scientific computing?



Why Java?

- A *clean* object-oriented programming language
- A safer language (no pointers...)
- An easier language (no pointers...)

No Pointers?

Well, almost...

- Primitive data types
- Objects id's

In C++: `Point *p = new Point();`

In Java: `Point p = new Point();`

The main problems in C/C++

The Java solutions

- int a= 4294967296+ 1;

  Primitive data types are well defined
- if (a=3) b=5;

  Boolean is a data type!
- int x,y; int *z=&x; *(z+1)=-1;

  No pointer arithmetic
- int z[10]; for (int k=0;k<=10; z[k++]=k;) {}

  Automatic array and string bounds check
- void f(){ O * a = new O(); }

  Automatic Memory Management (Garbage Collection)
- char *s; strcpy(s,"hello");

  Strings are a data type (Objects)

The result is...

- Developing in Java is estimated to be 10 times faster than developing in C++...
- ...but still 2 times slower than developing in VB...
- ... Visual Basic though is not (yet) as powerful as C++...
- ...but Java is.
  - Programs developed in Java are known to run slower than those developed in C++...
  - ...and programs written in C++ are known to run slower than those developed in C...
  - ...but in the end programs written by good programmers run much faster than programs written by less experienced ones.

10 good reasons...

- ***Language***. The Java programming language includes features beneficial for large-scale software engineering projects, including object-orientation, single inheritance, garbage collection, and unified data formats. Since threads and concurrency-control mechanisms are part of the language, parallelism can be expressed directly at the user level.
- ***Class libraries***. Java provides a variety of additional class libraries, including functions essential for Grid computing, such as the ability to perform secure socket communication and message passing.
- ***Components***. A component architecture is provided through JavaBeans and Enterprise JavaBeans to enable component-based program development.
- ***Deployment***. Java's bytecode allows for easy deployment of the software through Web browsers and automatic installation facilities.
- ***Portability***. Besides the unified data format, Java's bytecode guarantees full portability as represented by the concept "write once, run anywhere.''

From Getov et al.

10 good reasons...

- ***Maintenance***. Java contains an integrated documentation facility. Components written as JavaBeans can be integrated within commercially available integrated development environments.
- ***Performance***. Recent research results prove the performance of many Java applications can come very close to that of their C and Fortran counterparts.
- ***Gadgets***. Java-based smart cards, PDAs, and smart devices will expand the working environment for scientists.
- ***Industry***. Scientific projects are sometimes required to evaluate the longevity of a technology before it can be used. Strong vendor support helps make Java a technology of current and future consideration.
- ***Education***. Universities all over the world are teaching Java to their students.

From Getov et al.

The Question:

So what is this Java?

Java History

- Java was born as "**Oak**" at the beginning of the 90's
  target: **intelligent consumer electronics.**
- In 1994: recycled as "**the language for the Web**" (Client side: applets)
- Later: the champion of the "anti-Bill crusade" ("network computing" etc.)

Java Success Stories:

- "the language for portable Graphic Interfaces" (Swing)
- "the language for the Web" (Server side: servlets, Java Server Pages, Enterprise Java Beans)
- "the language for the B2B" (Strong support for XML: SAX, JAXP, SOAP...)
- "the easy language for ... serious Microsoft development (!)" — Visual C++, Visual J++, C#, Visual Basic

Applications, Applets & Servlets

Java can be written and run:

- as a **standalone program** (full fledged application running over a JVM)
- as an "**applet**" (Java code embedded into an HTML Page, which borrows resources by a JVM-enabled host application, typically a Browser)
- as a "**servlet**" (Java code running into an HTTP server, triggered by am HTTP request (I.e. from an HTML Page)

Hello World (application)

```
class Applicazione{

    /* Hello World, my first Java application */
    public static void main (String args[]) {
      Applicazione p= new Applicazione();
    }

    Applicazione() {
      System.out.println("Hello World!");
      // here goes the rest of the main
    }
}
```

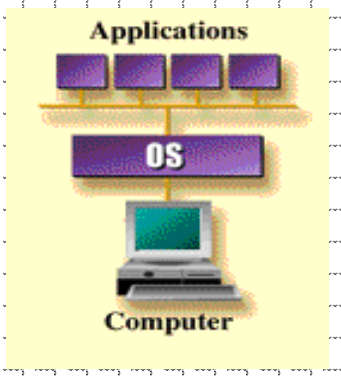
Java - Introduction

Applications are built in the frame of the

OPERATING SYSTEM

Which in turn is built over a particular

HARDWARE

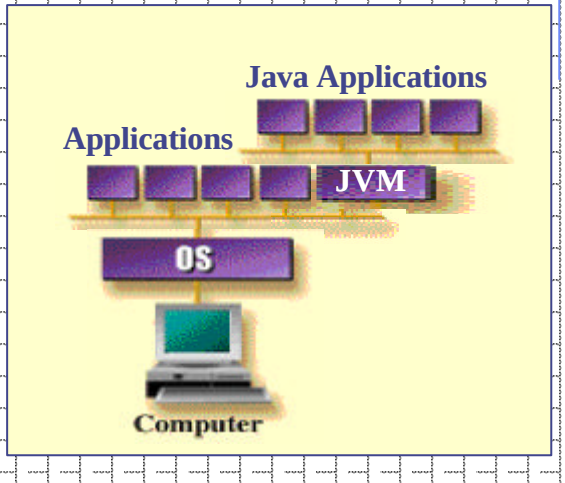


Java - Introduction

Java defines a HW-OS neutral

SOFTWARE LAYER

on top of which its code runs



The Java Virtual Machine

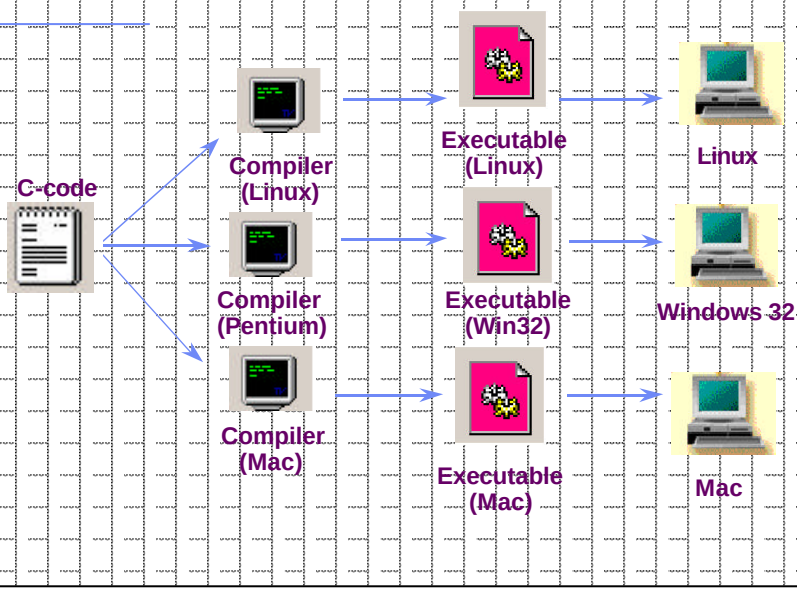
The Software Layer is called

Java Virtual Machine

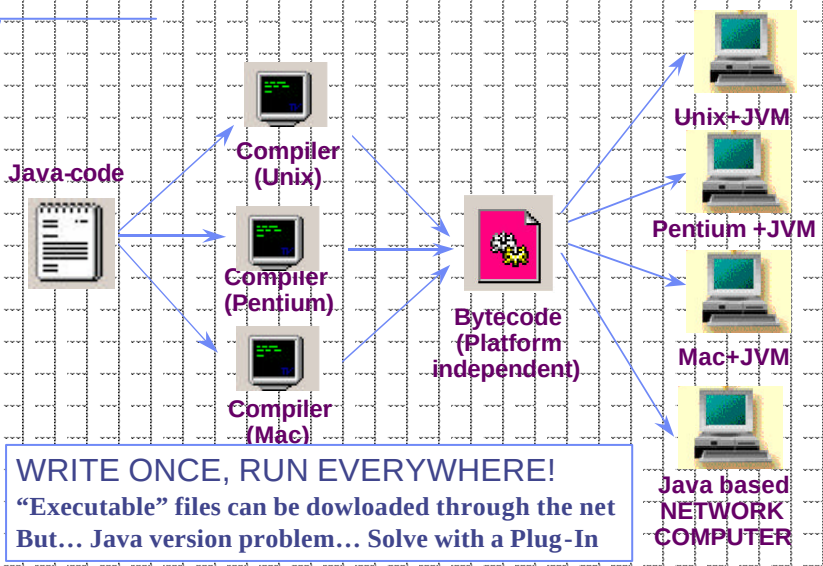
It is a (smart) *interpreter* of an assembly-like language called

ByteCode

Traditional "portability" (ideal)



Portability of Java programs



WRITE ONCE, RUN EVERYWHERE!

"Executable" files can be dowloaded through the net

But… Java version problem… Solve with a Plug-In

The Java Virtual Machine

The Java Virtual Machine can:

- be an application
- live inside an application (e.g. a Browser)
- live inside the core of the Operating System (e.g. JavaOS, an abandoned option)

Executing Java code

- Interpreters
- JIT Compilers (& Hot Spot)
- Direct compilers
- Bytecode to source

"the first universal software platform"

It is:

- Hardware independent
- Scalable
- Open

It consists of:

- The language — Easy!
- The Virtual Machine — You don't care!
- (Many) class libraries and API — That's the difficult part!

The Core Class libraries -1

The core API gives you the following features:

- **The Essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by Java applets.
- **Networking**: URLs, TCP and UDP sockets, and IP addresses.
- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.

The Core Class libraries -2

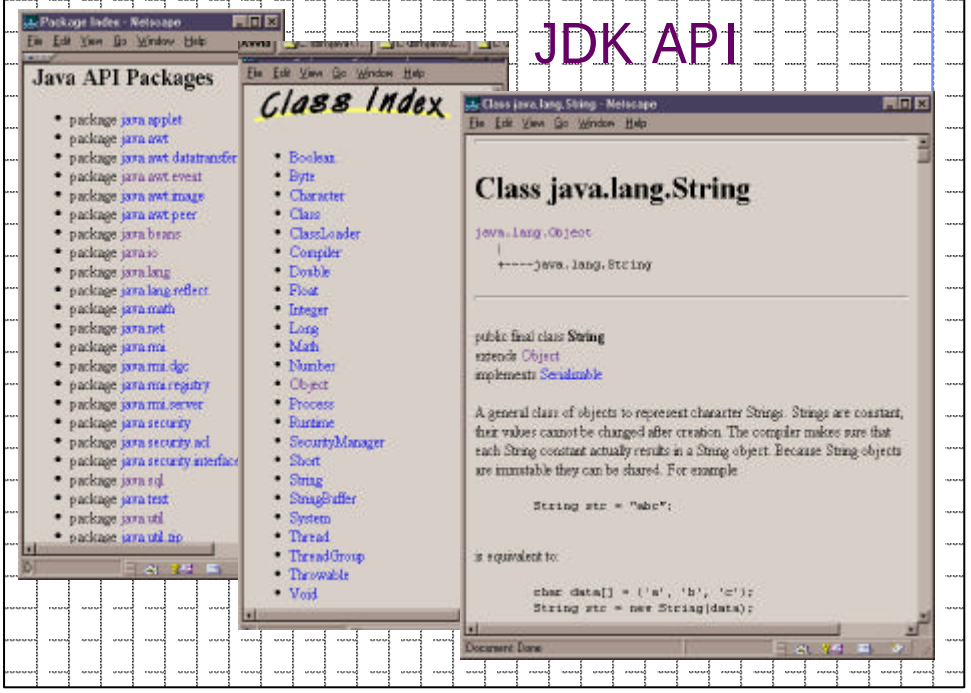
- **Security**: Both low-level and high-level, including electronic signatures, public/private key management, access control, and certificates.
- **Software components**: Known as JavaBeans, can plug into existing component architectures such as Microsoft's OLE/COM/Active-X architecture. They tipically contain the business logic, and are supported by "Application servers".

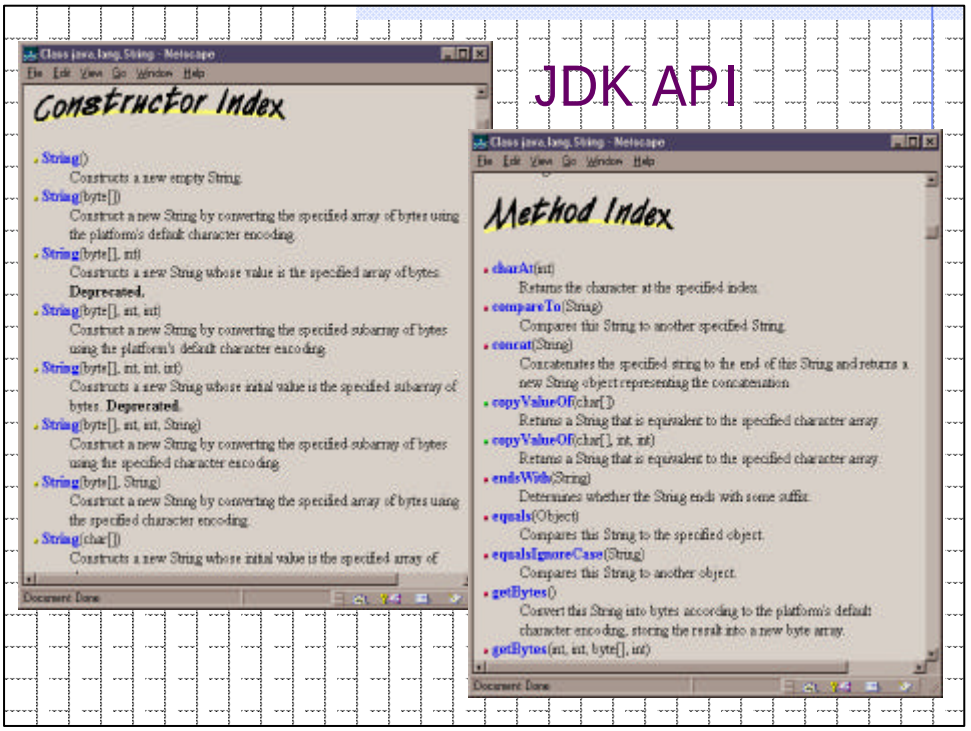
The Core Class libraries -3

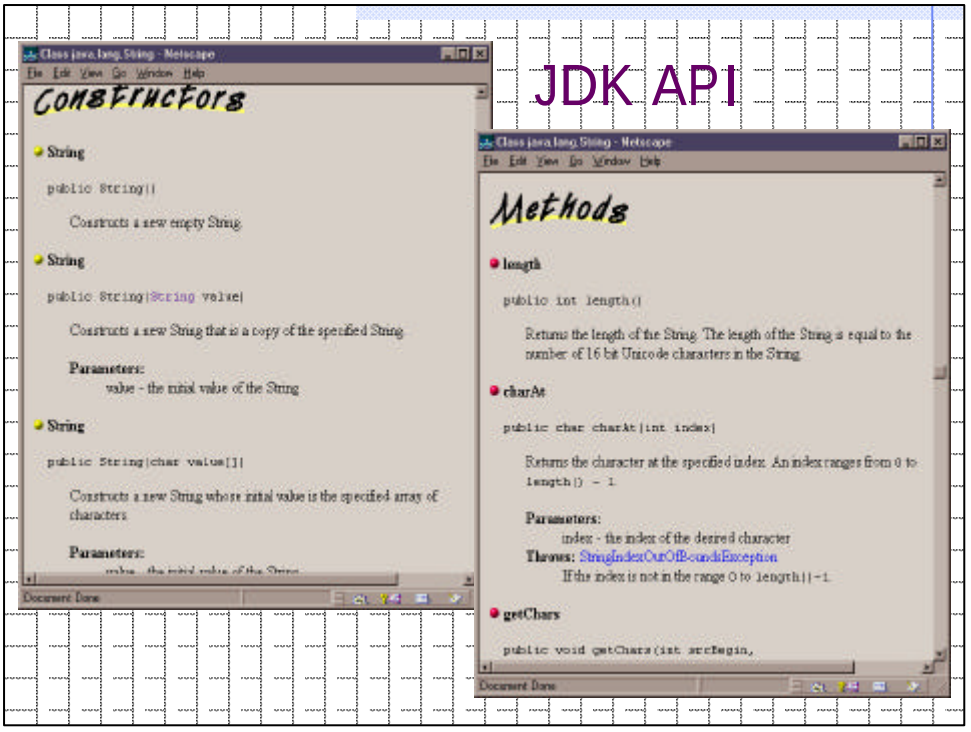
- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC)**: Provides uniform access to a wide range of relational databases.

Java not only has a core API, but also standard extensions. The standard extensions define APIs for 3D, servers, collaboration, telephony, speech, animation, and more.

JDK API




JDK API
Class java.lang.String - Netscape
File Edit View Go Window Help
Constructor Index
String()
Constructs a new empty String.
String(byte[])
Construct a new String by converting the specified array of bytes using the platform's default character encoding.
String(byte[], int)
Constructs a new String whose value is the specified array of bytes.
Deprecated.
String(byte[], int, int)
Construct a new String by converting the specified subarray of bytes using the platform's default character encoding.
String(byte[], int, int, int)
Constructs a new String whose initial value is the specified subarray of bytes. Deprecated.
String(byte[], int, int, String)
Construct a new String by converting the specified subarray of bytes using the specified character encoding.
String(byte[], String)
Construct a new String by converting the specified array of bytes using the specified character encoding.
String(char[])
Constructs a new String whose initial value is the specified array of
Document Done
Class java.lang.String - Netscape
File Edit View Go Window Help
Method Index
charAt(int)
Returns the character at the specified index.
compareTo(String)
Compares this String to another specified String.
concat(String)
Concatenates the specified string to the end of this String and returns a new String object representing the concatenation.
copyValueOf(char[])
Returns a String that is equivalent to the specified character array.
copyValueOf(char[], int, int)
Returns a String that is equivalent to the specified character array.
endsWith(String)
Determines whether the String ends with some suffix.
equals(Object)
Compares this String to the specified object.
equalsIgnoreCase(String)
Compares this String to another object.
getBytes()
Convert this String into bytes according to the platform's default character encoding, storing the result into a new byte array.
getBytes(int, int, byte[], int)
Document Done


JDK API
Class java.lang.String - Netscape
File Edit View Go Window Help
Constructors
String
public String()
Constructs a new empty String.
String
public String(String value)
Constructs a new String that is a copy of the specified String.
Parameters:
value - the initial value of the String
String
public String(char value[])
Constructs a new String whose initial value is the specified array of characters.
Parameters:
Document Done
Class java.lang.String - Netscape
File Edit View Go Window Help
Methods
length
public int length()
Returns the length of the String. The length of the String is equal to the number of 16 bit Unicode characters in the String.
charAt
public char charAt(int index)
Returns the character at the specified index. An index ranges from 0 to length() - 1.
Parameters:
index - the index of the desired character
Throws: StringIndexOutOfBoundsException
If the index is not in the range 0 to length()-1.
getChars
public void getChars(int srcBegin,
Document Done

Java ha supports “OS” features

It has primitives for:

- multithreading
- synchronization
- object distribution (RMI)

Multithreading

```
public class C extends Thread {
        public void run() {
                //implementation
        }
}

C myThreadedObject = new C();
…
myThreadedObject.start();
```

Starts a new thread,
Executes “run” in it,
Does not wait completion

Synchronization

Monitors

```
Class A {
        synchronized void f() {
                ...
        }
        synchronized int g(int k) {
        ...
        }
}
```

Synchronization

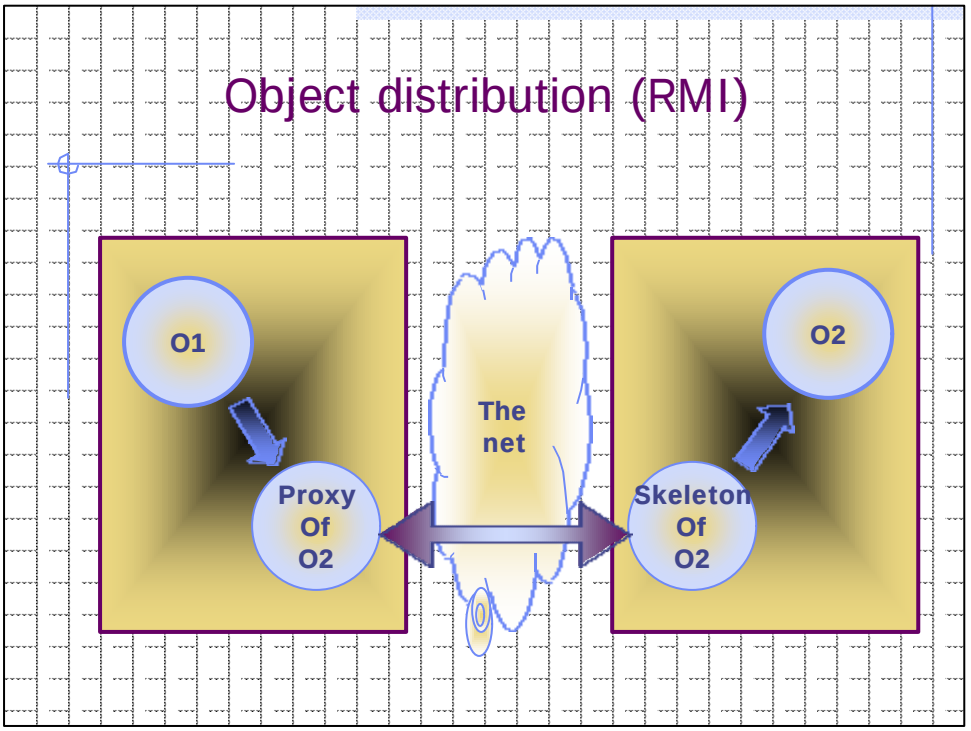
Condition variables, Wait and notify

```
Object foo;

synchronized (foo) {
        while (! condition) {
                // this thread enters the waiting queue on foo
                foo.wait();
        }
}

synchronized (foo) {
        // wake up the first thread in queue on foo
        foo.notify();
}
```

Object distribution (RMI)



The Question:

Is Java a viable solution
for scientific calculation?

A first answer

"The nature of many scientific applications makes them well suited to Java execution environments.

They typically spend a large amount of execution time in a small number of user-written methods, making them good candidates for JIT compilation, and less susceptible than other applications to poor implementations of the Java API.

Scientific applications, when written in a traditional, non-objected oriented manner, often have large and persistent persistent data structures, resulting in low garbage collection overheads."

J.M.Bull et al.

A first answer

Benchmarking Java against C and Fortran for Scientific Applications, Bull et al.

4.1 Intel Pentium, Windows NT
The ratio of the best Java to best C execution time has a mean of 1.23 and exceeds 2.0 only in the case of the FFT benchmark.

4.2 Intel Pentium, Linux
The mean ratio of fastest Java to fastest C execution times is only 1.07.

4.3 Sun Ultrasparc
Taking the best Java execution time and comparing to the fastest C execution time, we observe a mean ratio of 1.61, with a range from 1.29 (HeapSort) to 2.61 (SparseMatmult).

A first answer

Design and Evaluation of a Linear Algebra Package for Java, Almasi et al.

"Even though it is still in earlier stages of development, our Java linear algebra package has already achieved 65-85% of the performance of one of the most respected industrial-strength numerical libraries (ESSL)."

AJaPACK: Experiments in Performance Portable Parallel Java Numerical Libraries, Itou et al.

"... we record nearly 300MFlops, which is approximately about 1/2 of C performance."

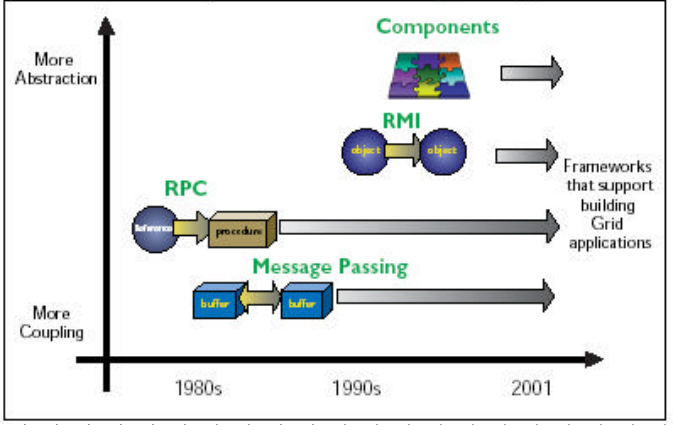
"Parallel execution on the SMP platforms scaled well, but not so on a PC, where practically no benefit and even performance loss is incurred with parallelization."

Research lines

- bytecode optimization
- dynamic compilation
- improved JVM techniques
  - Thread synchronization
  - RMI
  - Garbage collection
- improved language features
  - Better Floating point
  - Complex numbers
  - Support for regularly shaped arrays

Research lines – distributed computation

Figure 1. Multiple communication frameworks help program the diverse infrastructure in Grids. Remote procedure calls, message passing, remote method invocation, and component frameworks are the technologies of choice for building Grid applications.



From Getov et al.

Research lines – distributed computation

- Hyperion, cVJM, Java/DSM, Jessica
  - View a cluster of processors as executing a single JVM
- Manta, JavaParty
  - store shared data in objects accessible via RMI
- MPJ (Message Passing using Java)
- JOMP (Java Open Message Passing)

"When tested on a Sun E3500/8 UltraSPARC, the original serial code took 80.46 seconds, the parallel code on one processor took 81.02 seconds, and the parallel code on all eight processors took 12.23 seconds. This represents a speedup factor of 6.58, and an efficiency of 82.2%. Similar results were obtained when the same code was parallelised by hand." Bull et al.

Papers:

From: Communications of the ACM

- High-performance Java
- The NINJA project
- Enabling Java for high-performance computing
- Multiparadigm communications in Java for grid computing

From: ACM Computing Surveys

- Techniques for obtaining high performance in Java programs

Papers:

From: Proceedings of the ACM 2000 conference on Java Grande

- JOMP—an OpenMP-like interface for Java
- Development routes for message passing parallelism in Java
- A JaPACK: experiments in performance portable parallel Java numerical libraries
- Design and evaluation of a linear algebra package for Java
- A high performance cluster JVM presenting a pure single system image

Papers:

From: ISCOPE Conference on ACM 2001 Java Grande

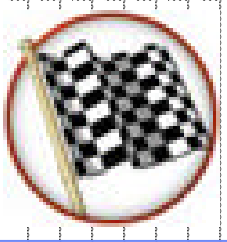
- High performance Java code in computational fluid dynamics
- Benchmarking Java against C and Fortran for scientific applications
- Object-based collective communication in Java
- A scalable, robust network for parallel computing
- Automatic translation of Fortran to JVM bytecode

Useful links:

- Java Grande Forum
- The Java Memory Model
- Java Home (Sun site)
- Java Platform Performance Strategies and tactics (Book on line)

High Performance Java

Thanks for your attention!



marco.ronchetti@dit.unitn.it