Using Java for Scientific Computing

Mark Bul
EPCC, University of Edinburgh
markb@epcc.ed.ac.uk

Java and Scientific Computing?

- Benefits of Java for Scientific Computing
  - Portability
  - Network centricity
  - Software engineering
  - Security
  - GUI development
  - Trained programmers
  - Availability and cost

- Problems
  - Performance
  - Numerical problems
  - Parallel programming models

Portability

- Java is platform neutral
  - Compiler generates byte-code for the Java Virtual Machine (JVM)
  - Byte-code is platform independent - runs on any platform with a JVM
- Language Specification
  - No platform dependent aspects of the language specification
  - e.g. size of primitive data types is specified, not platform dependent
- Hence both Java source and byte-code is extremely portable

Why Portability?

- Rapidly changing technology
  - Applications codes typically have a longer lifetime than hardware (3-5 years)
  - Much effort spent on porting codes between systems
  - Fortran and C only portable with care and expert knowledge
- Publishing applications
  - great way to share applications via the WWW
  - no problem of conditional compilation, nasty configure scripts
  - don’t need to publish source code (just publish byte code)
- Heterogeneous Grid computing
  - The user has a single meta-resource for solving their problem
  - How to compile if target hardware unknown at job submission time?
  - Java is a natural language choice for the Grid

Network Centricity

- Java has considerable built-in support for distributed computing
  - e.g. remote method invocation (RMI) - allows Java to invoke methods of remote Java objects as if they were local
  - Also stream based connections via sockets
  - Dynamic class loading facilities allow a JVM to download and run code from across the internet
- Important for remote visualisation, computational steering
- Natural candidate at least for the gluing applications together, if not programming the computational kernels themselves

Software Engineering

- Java is an Object-Oriented Language
  - well establish programming paradigm
- Encapsulation and polymorphism
  - facilitates code re-use
  - reduced development time
- Some scientific applications don’t fit the O-O model nicely, but you don’t need to use it
  - can write Java codes in a procedural manner
- Simpler and cleaner than C++

Software Engineering

- Java has many nice features
  - No pointers
  - Garbage collection
  - Type checking
  - Array and string bounds checking
  - Exception handling
  - Standard debugger with JDK
  - Extensive standard class libraries

- Faster development times
  - Rapid prototyping
  - Less buggy code

Security

- Java has a number of security features
  – Essential for a distributed language
- No direct access to memory
  – Cannot forge pointers to memory, overflow arrays, read memory outside array bounds
- Byte verification process
  – Performed on any untrusted code
  – Ensure code is well formed - prevents corrupted byte code
- Sandbox
  – Untrusted code runs within a “sandbox”
  – Has restrictions on what it can do. e.g. no access to local file system
- Digital Signatures
  – Can be attached to Java code - trusted code can run without sandbox restrictions

Other Benefits

- Java provides a portable and easy to use GUI library
  - Advantage over C/C++ which has platform specific libraries
  - Allows GUIs / applets to be developed for scientific applications
  - Easier to view and share results

- Trained Programmers
  - Java is rapidly becoming the language of choice in undergraduate courses
  - Students / teenagers interested in Java - creating applets for their web pages.
  - Will become easier to recruit good Java programmers, Fortran programmers (even C programmers?) will become rare.

Availability

- Java is available on almost every platform
  - PC (Windows and Linux), Sun, SGI, HP, IBM, Hitachi, ….
- Cheap
  - Java technology is free for almost all platforms
- Reliable
  - A decent Java implementation is seen as important by most vendors

Issues

- Performance
  - is Java performance unacceptable compared to traditional languages (e.g. C and Fortran)?

- Numerics
  - number of concerns relating to complex numbers, floating point arithmetic, multidimensional arrays..

- Parallel programming models
  - does Java support any standard parallel programming models?

Performance

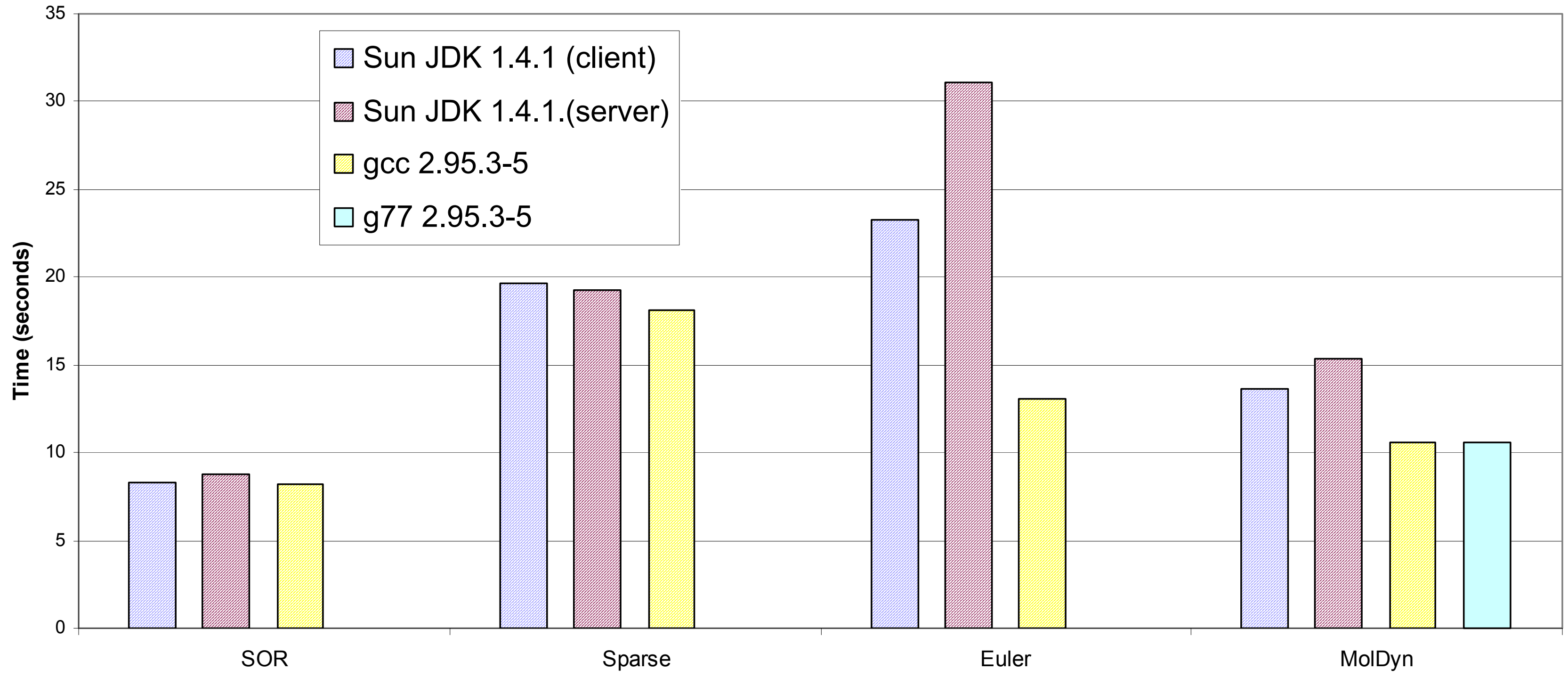
- Java has a bad name for performance
  - early implementations were interpreters
  - Java based GUIs can be very poor
- Much effort has been expended on just-in-time compilers.
- Performance is dependent on how code is written
  - Standard class libraries often much worse than user-written code
  - Heavy OO design can be costly in performance
- In best case
  - Within a factor of 2 of highly optimising Fortran and C compilers
  - Competitive with gcc

Java Grande Benchmarks

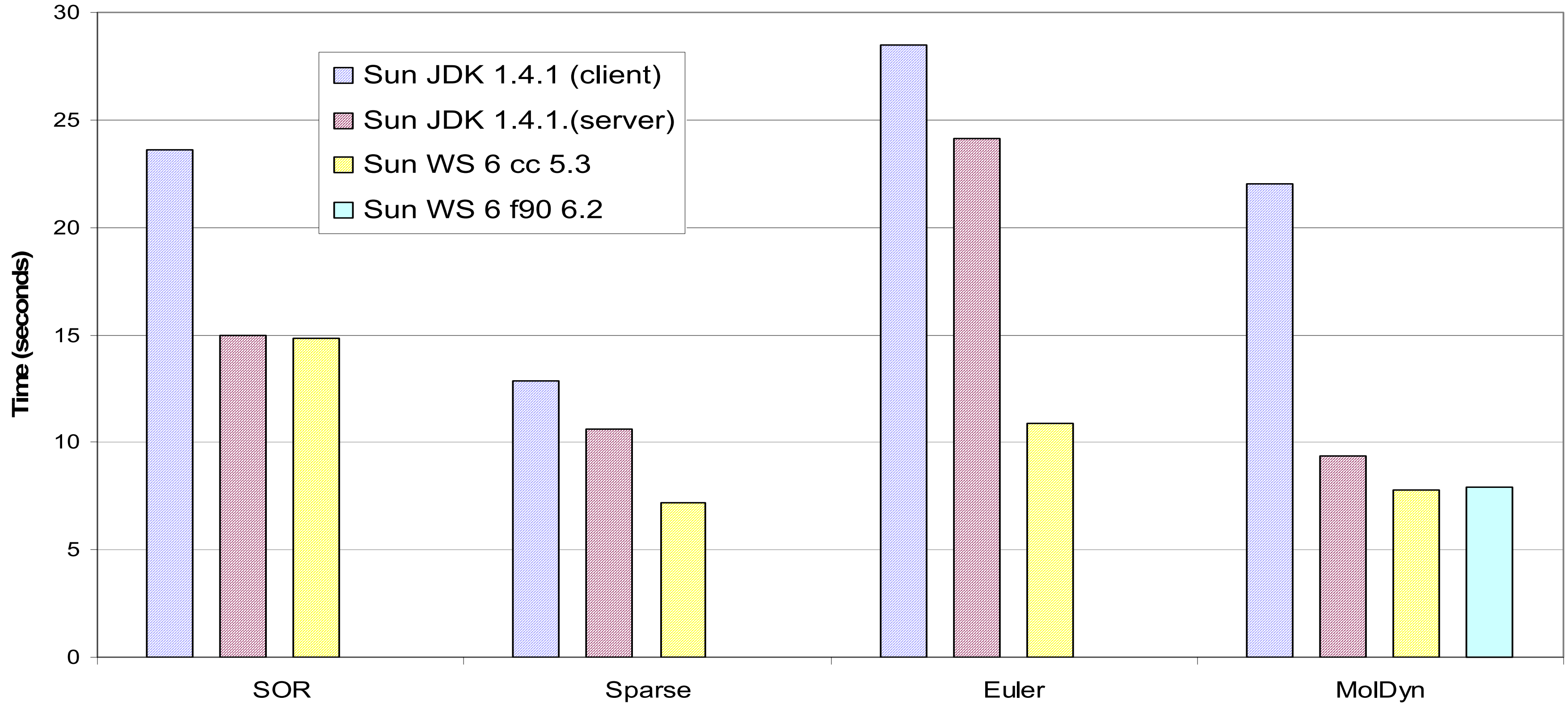
See **www.epcc.ed.ac.uk/javagrande/** for details

- SOR
  - 100 iterations of successive over-relaxation on an **NxN** grid
- Sparse
  - matrix vector multiplication using an unstructured sparse matrix stored in compressed-row format with a prescribed sparsity structure
- MolDyn
  - a simple $O(N^2)$ N-body code modelling particles interacting under a Lennard-Jones potential in a cubic spatial volume with periodic boundary conditions
- Euler
  - solves the time-independent Euler equation for flow in a channel with a bump on one of the walls

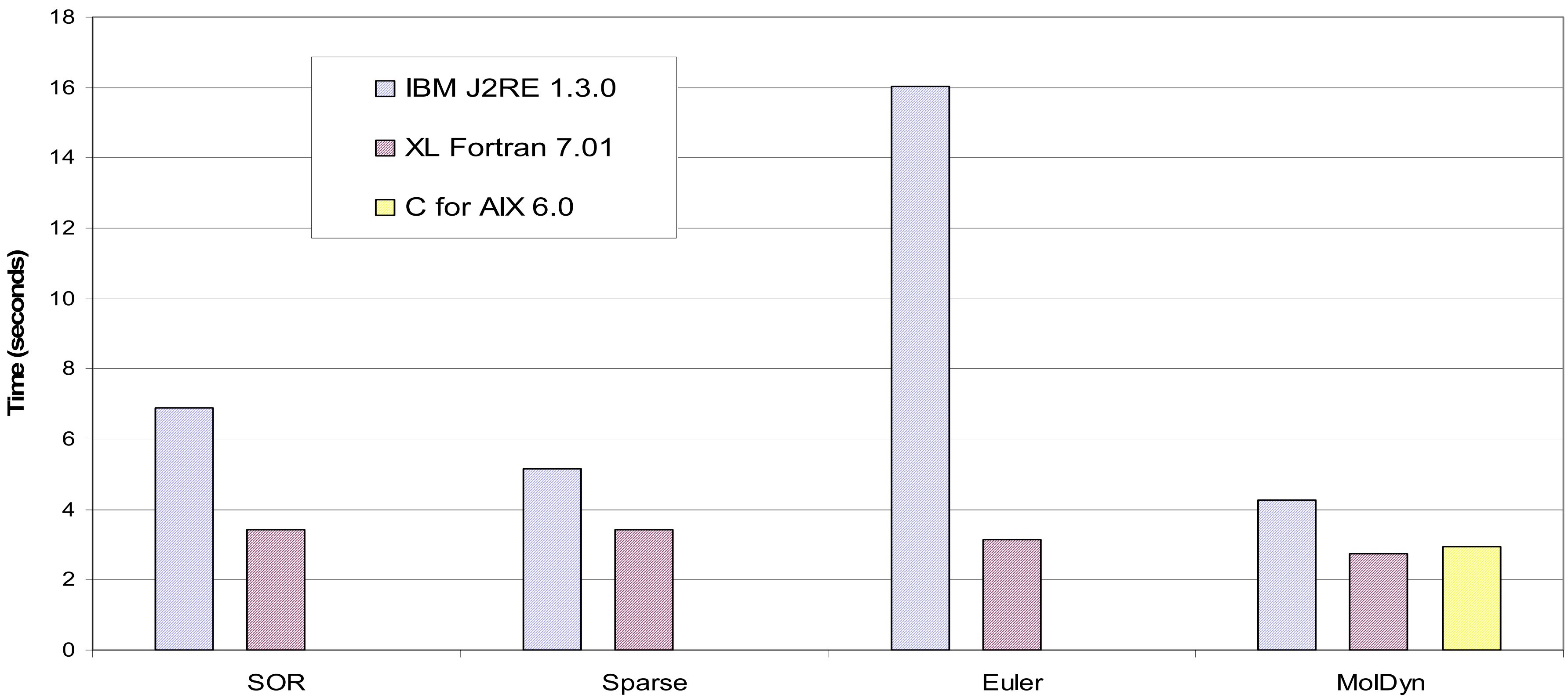
Pentium III PC



Sun Ultrasparc III



IBM Power4



IBM J2RE 1.3.0
XL Fortran 7.01
C for AIX 6.0
Time (seconds)
0
2
4
6
8
10
12
14
16
18
SOR
Sparse
Euler
MolDyn

Numerics Issues

- Java was not primarily designed for numerically intensive computation.

- Some of the early design decisions in the language reflect this, and now seem cast in stone...

Support for IEEE 754

- Java's floating point arithmetic mostly follows IEEE 754

However:

  - Java only supports Round-to-nearest
  - Java cannot trap IEEE floating point exceptions
  - Java only defines one bit pattern for NaNs

- For most applications this is OK, but some users really do care about this!

Complex Numbers

- Lack of efficient support in Java
- Currently need a Complex class, objects contain e.g. two doubles
  - rather complicated method calls
  - behave differently from primitive types
  - performance hit involved compared to primitive types
- Technically, there are a number of possible solutions, but none seem likely to be adopted.

Multidimensional Arrays

- In Java multidimensional arrays are arrays of one dimensional arrays
  - optimisation problems
  - different row lengths, multiple bounds checking at run-time, not contiguous in memory
- To improve performance requires true rectangular arrays (all rows the same length)
- A multi-dimensional array package is available, but the interface is not very pleasant...

Libraries

- Java is a young language
  - Fewer standardised numerical libraries available than traditional HPC languages
- Java Native Interface (JNI) provides Java codes with access to native code (e.g. MPI and LAPACK)
- Less than ideal
  - loss of security, portability, reproducibility, robustness
  - run-time overhead in invoking a native method
- Libraries written in Java
  - Java Numerical Library (JNL), Visual Numerics
  - JAMA, Mathworks + NIST
  - see: **`http://math.nist.gov/javanumerics`**

Parallel programming

- Java has some built in parallel programming paradigms:
- Java Threads
  - shared memory paradigm
  - tolerable efficiency on moderate size SMP systems
  - utility should be improved by addition of Concurrency package in Java 1.5
- RMI (Remote Method Invocation)
  - invoke methods on objects in another JVM
  - high latency
  - very different paradigm from message passing
- BSD Sockets
  - high latency
  - more suitable for client/server style

MPI and OpenMP

- No standardised equivalents in Java
  - some research grade projects, but nothing of industrial strength
- Possible to use JNI to access native MPI libraries
  - mpiJava, MPJ define interfaces
  - not portable solution
  - serialization of objects is a bottleneck
- Possible to have a pure Java implementation
  - so far performance is disappointing
- Can implement a pure Java OpenMP-like interface on top of Java threads
  - see **`www.epcc.ed.ac.uk/research/jomp`**

Summary

- Java has some very attractive features as a programming language
- It was never designed for scientific computing, so there are some inherent disadvantages
  - don't expect them to be fixed any time soon.....
- In the end, it depends on your priorities for your application.