Java Grande Forum

Industrial/Federal/Academic consortium

http://www.javagrande.org

Applications: computationally intensive.

Commercial: datamining, financial modeling, reservoir simulation, seismic data processing, aircraft design etc.

Government: nuclear stockpile stewardship, climate and weather modeling, environmental impact etc.

Academic: fundamental physics, biochemistry, earthquake prediction etc.

Currently, the realm of Fortran and C

Forum believes that Java could be the best development environment.

Forum gives recommendations to this end.

Forum is open to any qualified member willing to play an active role.

Contact: Geoffrey C. Fox gcf@npac.syr.edu

Java Grande Forum

Series of High Performance workshops

Dec. '96, June '97, Mar. '98

http://www.npac.syr.edu/projects/javaforcse/

Java Grande Forum initiated

Java Grande Forum Meetings

Mar. '98 JavaOne BOF

May '98 JGF

Aug. '98 JGF

Sep. '98 Supercomputing

Latest JGF meeting now

Numeric Working Group

Java performance comparable to Fortran/C.
Language features and standard libraries.

http://math.nist.gov/javanumerics

Critical Java and JVM issues:

- Complex arithmetic
- Lightweight classes
- Operator overloading
- Floating-point hardware standard
- Multidimensional arrays

Constraints:

- Small changes to Java and JVM specifications.
- Good execution speed.
- Predictability of results is important.

Concurrency and Applications Working Group

Improving Java numeric performance is just the first step.

Leverage solutions from enterprise server applications: require high throughput and large memory.

- DB2, ESSL, LAPACK

Technical computing also has separate requirements, particularly, parallelism.

Expect techniques to become ubiquitous.

- Parallelism

Java and JVM Issues

Need benchmarks that are representative of Java Grande applications.

Better Remote Method Invocation (RMI):
- More efficient object serialization.
- More efficient network protocols.

Message Passing Interface (MPI) for Java.
- Most current parallel codes Fortran/C+MPI

Better Java Remote Native Interface (JNI)
- Important for legacy libraries

Case Studies

Performance of existing Java environments not competitive, but things are changing rapidly.

Compiler techniques:
- Safe regions
- Optimizing bytecodes
- Optimizing Java code
- Dynamic optimizations

Libraries:
- Idiomatic programming
- Interfacing to legacy codes
- Semantic expansion/lightweight methods

Dialects

Safe Regions

Java requires index/null-pointer checks:

```
for (int i=0; i<m; i++)
  for (int j=0; j<p; j++)
    for (int k=0; k<n; k++)
      C[i][j] += A[i][k]*B[k][j];
```

Requires 6 index and 6 null-pointer checks per iteration.

The possibility of exceptions prevent iteration reordering.

Solution: indicate regions where exception can be proven not occur.

M. Gupta, S. Midkiff and J. Moreira
http://www.research.ibm.com/ninja

Versioning

```
if ((m <= rows(C)) && (p <= cols(C)) &&
    (m <= rows(A)) && (n <= cols(A)) &&
    (n <= rows(B)) && (p <= cols(B)) {
// Safe region, so checks disabled.
  for (int i=0; i<m; i++)
    for (int j=0; j<p; j++)
      for (int k=0; k<n; k++)
        C[i][j] += A[i][k]*B[k][j];
} else {
// Unsafe region, so checks enabled.
  for (int i=0; i<m; i++)
    for (int j=0; j<p; j++)
      for (int k=0; k<n; k++)
        C[i][j] += A[i][k]*B[k][j];
}
```

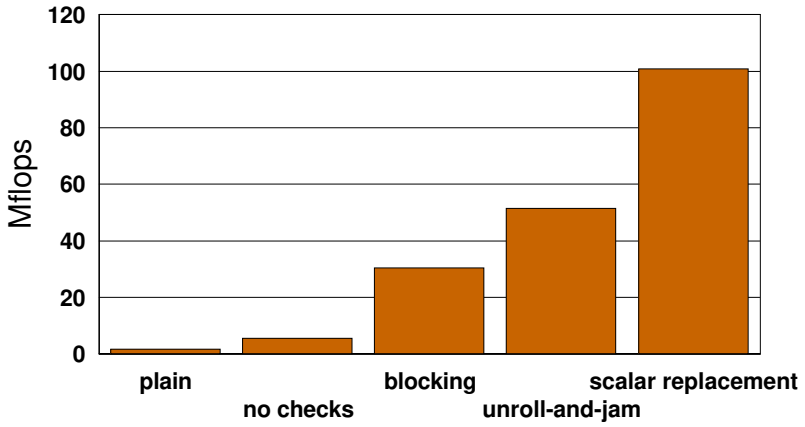
Optimizing Safe Regions

Safe regions do not require explicit index checks.

Safe regions can be optimize with standard transformations:

- Tiling: improved memory performance
- Unroll-and-Jam: improved data reuse
- Scalar replacement: reduced access to elements
- Etc.

Performance results for Matrix Multiplication on RS/600 590



Optimizing Bytecode

Issues:

- JVM is stack based machine
  - Stack operations inherently serial
- Bytecode is higher level than machine code
- Usually no knowledge of whole program

- OOP style programming uses lots of methods
  - Inlining important

- Most optimizations on control/data-flow graph
- Most machines are register-based

So must recreate program structure

Trilema:

- functionality vs. portability vs. performance
  - http://www.npac.syr.edu/gfc/PPTjavarice/

Optimizing native code

IBM's HPCJ

Compilation prior to execution

Leverages existing optimizing backend

Uses C stack convention

Platform specific

No dynamically load classes

Java Enhanced with Traditional compilation:

- Execute stable code with HPCJ
  - Turn off array bounds checking
- Execute rest with JIT
  - For dynamic linking

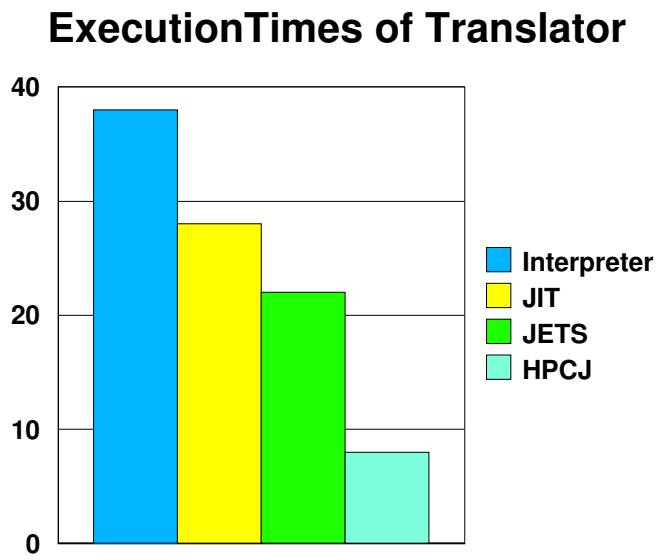
Uses SUN/TRL garbage collector

Exceptions handled with trap instructions

- If JIT pass to JIT trap handler
- else locate JET frame with try block

HPCJ Performance

Test case: Portion of HPCJ



Dynamic Compilation

- JIT Compiler with profiler and optimizer
  - At runtime monitor program behavior
  - Highly optimize most frequent code
  - Tradeoff analysis and execution time
  - Fast start up time
  - Target machine specific
  - Rich research area
  - Dynamic class loading complicates matters
  - May need to decompile code
    - Debugging
    - Exceptions
- Examples: HotSpot,Jalapeño

Hot Spot

http://www.java.cun.com/products/hotspot/index.html

- Sun execution engine
  - 100% performance improvements
- Written in Java and C
  - Share stack
- Handleless objects
  - Pointers
- Threads implemented as OS threads
  - Suitable for SMP
  - Ultra fast thread synchronization
- Dynamic analysis and optimization
  - Heavy inlining
- Dynamic deoptimization
- Garbage collector
  - Train algorithm: Pauseless, Generational
  - Accurate
  - Mark and compact

Jalapeño

http://www.research.ibm.com/jalapeno/

- Written in Java
  - Can optimize itself
  - Break strong typing for TOCs and Stacks
- Targeted to servers
  - Fast response time
  - Throughput
  - Scalable
  - High availability
- Targeted to SMPs
  - Memory resources not critical factor
- Subsystems
  - Runtime
  - Compilers
  - Threads
  - Memory management

Jalapeño Runtime

- Runtime services
  - Exception handling
  - Dynamic type checking
  - Dynamic linking
    - Method offset obtained via table
    - Backpatch at link time
  - Interface invocation
  - Reflection
  - I/O
- Magic class (circumvents restrictions)
  - Invoke OS services
  - Use architecture specific instructions
  - Access registers and memory
  - Coerce object references to addresses and back
    - Must disable garbage collection
  - Transfer execution to arbitrary address

Jalañeno Compilers

- Baseline compiler
  - Naive emulation of JVM specification
  - Stack based
  - Easy to implement
- Quick compiler
  - Low compile-time overhead
  - Basic optimizations
    - Register allocation
    - (investigating inlining)
- Optimizing compiler
  - Profile-driven dynamic optimization
  - Full optimizing backend

Different parts of code can be compiled by different compilers at different times

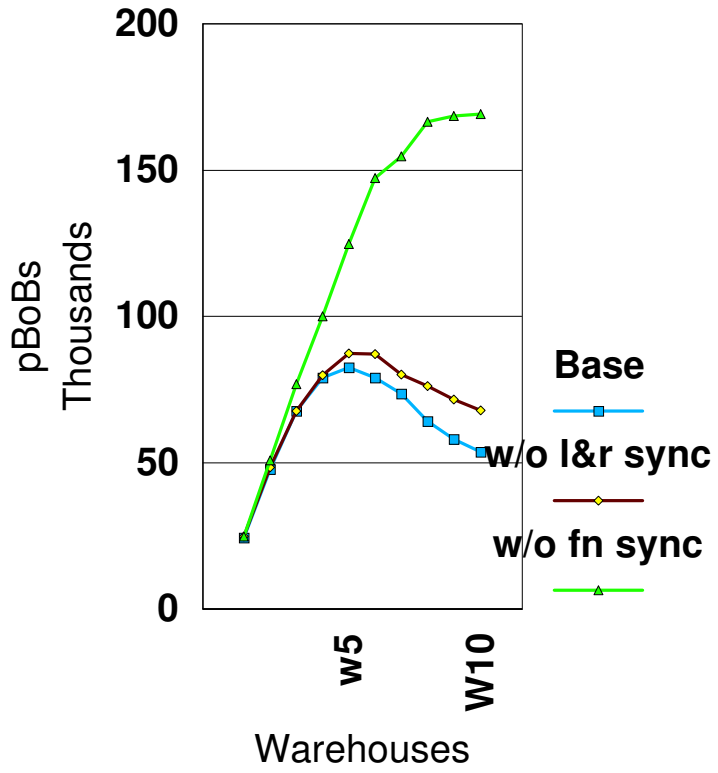
Jalapeño Threads

- User level threads
  - Java threads multiplexed on virtual processors
  - Reduces thread switching overhead
  - Supports rapid transition to garbage collection
- Quasi preemption
  - Thread switching only at predefined yield points
    - Bounded time between yield points
  - Motivation
    - Voluntary switching at odds with fast response time
    - Fully preemptive switching complicates GC
- Bimodal locking
  - A variant of thin locks for uncontended locks
  - Even contended locks do use OS calls

Jalapeño Memory Management

- Family of memory managers
  - Different spaces for large and small objects
  - Object allocator
  - Garbage collector
- Concurrent object allocation
- Garbage collection
  - Stop-the-world
  - Parallel
  - Type accurate
    - All object references must be located
    - No non-reference mistaken for reference
    - (Incremental to be investigated)
  - Copying and non-copying
  - Generational and non-generational

Jalapeño Performance



Running with 8 Virtual processors

Java Libraries

Package of related methods
- Ease of use, maintainability, reuse etc..
- Highly tuned
- Widely used
- Bread and butter of HP computing

Java libraries have two implementations:
- One pure Java.
- One highly optimized.
  - Utilize hardware and system features

Interface to non-Java libraries (JNI)
- Java wrappers

Examples:
- Matrix Package
- SPMD constructs
- JLAPACK

Matrix Package

100% pure Java implementation and performance of Fortran 90 arrays.

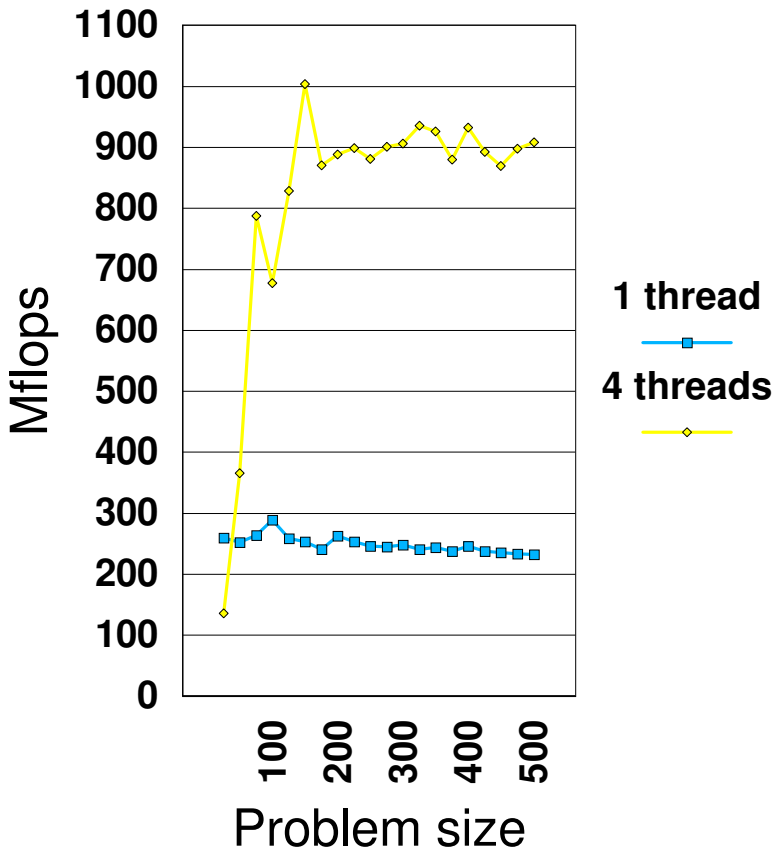
One class for each elemental data type and rank, e.g. doubleArray2D and intArray2D

Array operations are atomic
- Can be highly optimized and parallelized
- More optimization possible...

Example:

```
doubleArray2D A, B, C;
for (i=0; i<m, i++)
  for (j=0; j<p; j++)
    for (k=0; k<n; k++)
      C.set(i,j,(C.get(i,j)+A.get(i,k)*B.get(k,j));
```

Java DGEMM for Array Package



FMA

A fused multiply-add (fma) operation computes a*b+c with a single rounding.

Motivated by linear algebra

Not support by Java today

On POWER/PowerPC cuts peak by half.

Java Grande proposal would make it legal

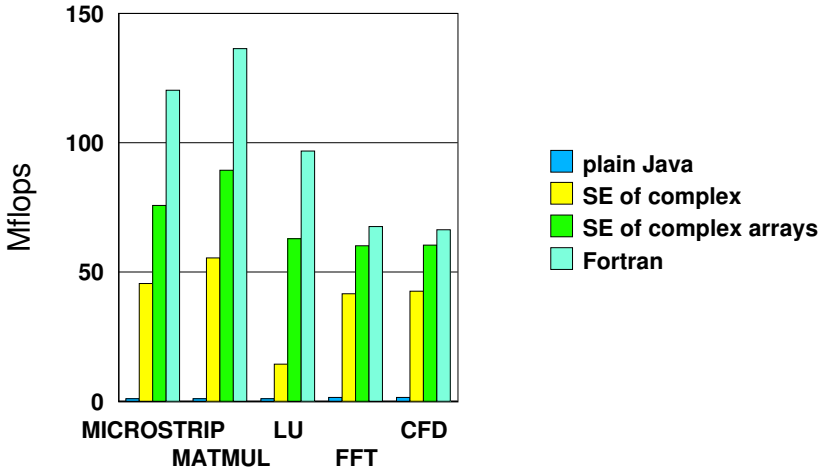
Java did alter spec enough for Intel MMX
- SIMD execution

FMA Performance

4-processor RS/6000 F50

MATMUL: matrix multiplication (500 x 500 matrices).
TOMCATV: mesh generation and solver (513x513 mesh).
BSOM: data mining neural-network training (16 nodes).
CHOLESKY: Cholesky factorization (1000x1000 dense).
SHALLOW: shallow water simulation (256x256 grid).

100% Java achieves ~900 Mflops for large matrices
SMP ESSL achieves 1200 Mflops on same machine

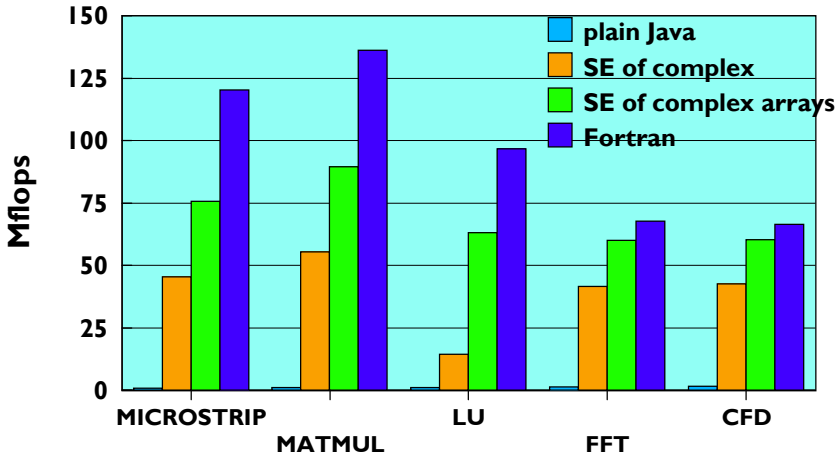


Semantic Expansion

Compiler "knows" the semantics of operations and can highly optimize
- Transparent to the language and programmer

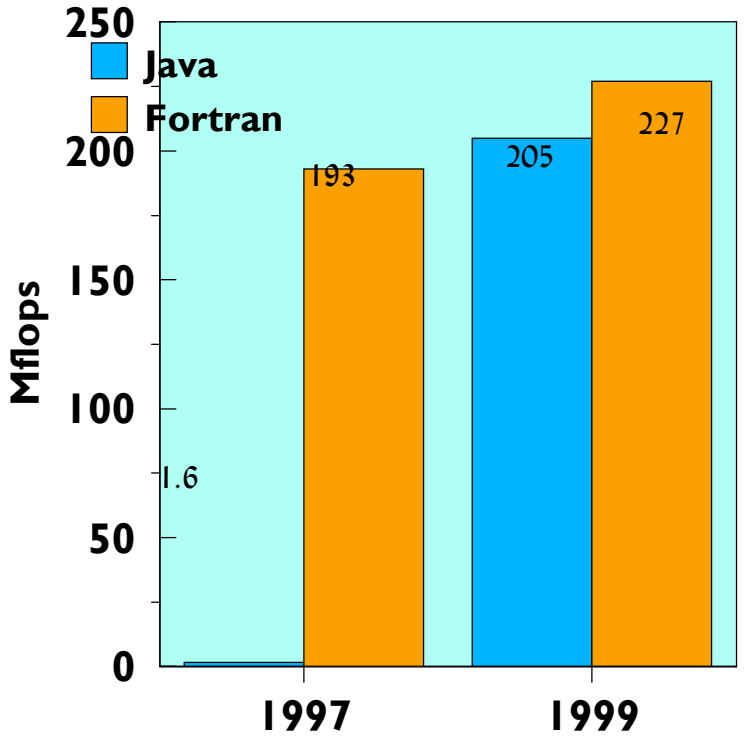
Lightweight Objects
- New kind of object: More general approach
- Value semantics
  - Different than inlining
- No references
- No dynamic dispatch
- Highly optimized

Java semantic expansion on RS/6000 590



Have we made progress?

MATMUL on RS/6000 590



SPMD Library

Parallelism in numeric codes
- Homogeneous, regular
- Cooperate (producer/consumer)
- Performance

Parallelism in systems
- Heterogeneous, irregular
- Competition
- Inherent

Package of SPMD constructs

www.npac.syr.edu/projects/javaforcse/javameettalks.html
- Two versions:
  - Pure Java
  - Java wrappers and C+MPI code
- forall: parallel loop expressed as Thread array
- Producer/consumer expressed with wait/notify
- topology
- collective communication: barrier etc.

SPMD Example

```
for (t = 0; t < SPMDsor.ConvTime; t++) {
  forAll((sor,0,N,1);  }
void SPMDloop(i) {  // Method of sor
     if (i != 0) {
        if (thread_id() == 0) {
          z=SPMDsor.threads[thread_id()-1].vOld[N-1];
          y = (z + Old[i+1])/2;
        } elsey  = vOld[i+1]/2;
     } else if (i == N - 1) {
        if (thread_id() == N - 1) {
          z = SPMDsor.threads[thread_id()+1].vOld[0];
          y = (z + vOld[i-1])/2;
        } else y  = valuesOld[i-1]/2;
    } else y  = (vOld[i-1]+vOld{i+1])/2;
    y=max(obstacle[i],vOld[i]+Psor.omega*(y-vOld[i]));
    vNew[i] = y; }
  Barrier(Psor.barrier);
  for (i = 0; i < N; i++)
    vOlld = vNew[i];
  Barrier(Psor.barrier);}
```

Producer/Consumer

wait and notify "..especially appropriate in situations where threads have a producer-consumer relationship..."

Notifies not queued.

Must use mutual exclusion and counters to enforce ordering

Low-level programming.

This does not work:

```
produce();
notify(data);

wait(data);
consume();
```

Legacy Libraries

LAPACK, BLAS etc.

JLAPACK

- Many, many available over the net
- Created using Fortran-to-Java tool, or
- Dutiful recoding of Fortran in Java, or
- Written in OOP style from scratch

See Scientific Computing July 1999

Java Linpack available at:

- http://www.netlib.org/benchmark/linpackjava/
- Performance times
  - 0.001 Mflops/s 1/18/97
  - 2148 Mflops/s 3/9/99

JCI

Tool for automatically generating Java wrappers for native library functions.

http://www.perunhscs.wmin.ac.uk

Input:

Header file of C function prototypes

Output:

File of C stub-functions

Files of Java class and native method declarations

Shell scripts for compiling and linking

Difficulties

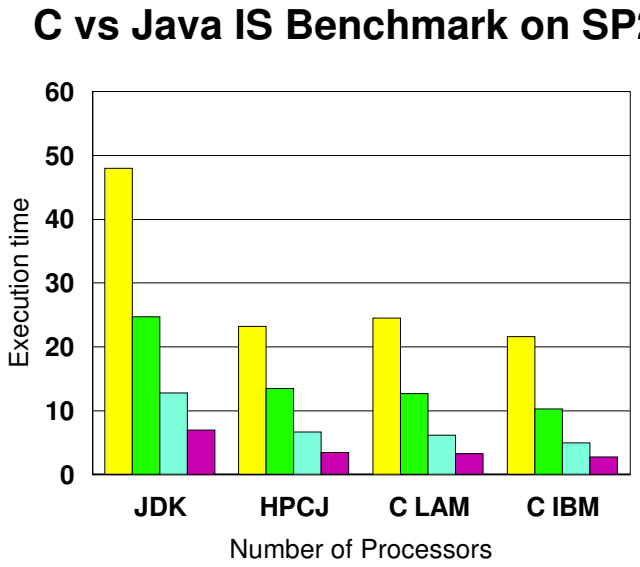
Pointers/Addresses

Array offsets, indexing and multiple dimensions

Has been applied to:

MPI, BLACS, LAPACK, ScaLAPACK etc.

JCI Results



Dialects

- Superset of Java to add functionality
  - Expand application areas
    - SIMD programming
    - Functional programming
  - Preprocess program for portability
    - Can lead to House of Babel
- Examples:
  - Pizza
  - Titanium

Titanium

- Java dialect for large-scale scientific codes.
  - http:/www.cs.berkeley.edu/Research/Projects/titanium
  - Available over the net
  - Goals:
    - Safety
    - Portability
    - Support for building complex data structures
- Targeted to multiprocessors and NOWs.
  - Same program runs on both
- Main additions:
  - Collective communication
    - Barriers etc.
  - Multidimensional arrays
  - Explicit SPMD programming model
    - Checked synchronization
    - Programmer control of thread/data alignment
  - Zone-based memory management

Pizza

Java dialect for functional programming

- http://www.cis.unisa.edu.au/~pizza
- Available over the net

Goal: easier to use

Main additions:

- Paraametric polymorphism
  - Templates
- First-class functions
  - Can pass as parameters etc.
- Calss cases and pattern matching
  - Set opertions
  - Associative lookup