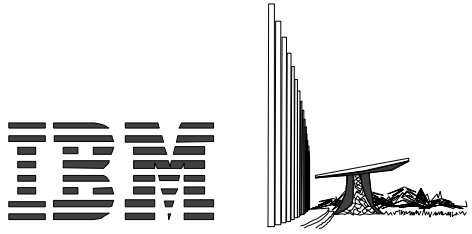
Floating-Point Performance in Java

Marc Snir

Jose Moreira, Manish Gupta, Lois Haibt,
Sam Midkiff



Thomas J. Watson Research Center
PO Box 218
Yorktown Heights, NY 10598

May 1998

Q: can Java replace C (C++, Fortran) for Numeric-Intensive Codes?

- **What are inherent impediments to floating point performance due to language design?**
  - How can those, if any, be fixed?
  - When will be those fixed?

- **What are impediments to floating-point performance due to to current state-of-the art in Java support?**
  - How can those, if any, be fixed?
  - When will be those fixed?

- **IBM goal: Java should become language of choice for server code, including technical computing code.**
  - Ongoing collaboration with Sun to make this happen.

Why bother, at all?

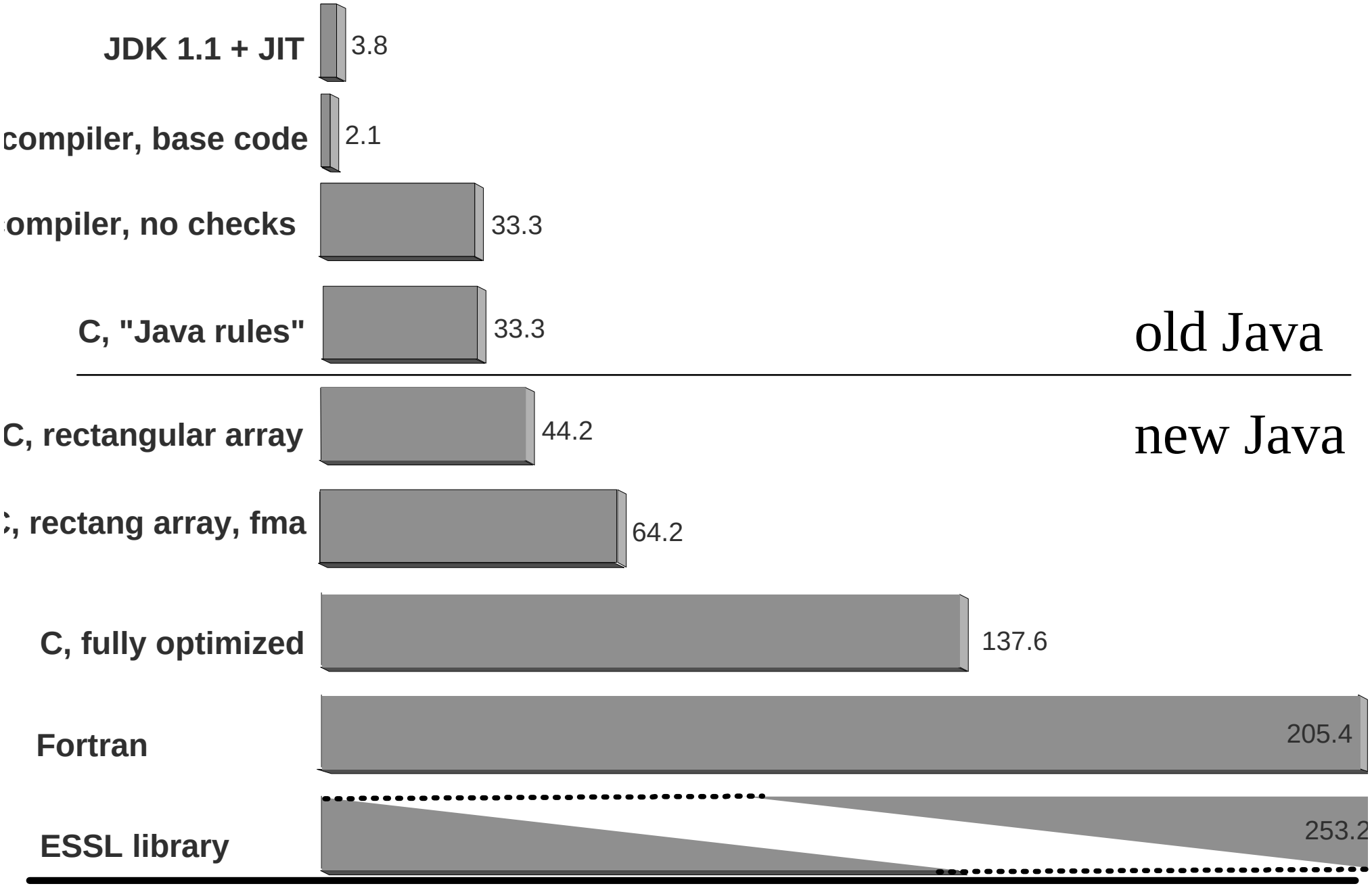
- **Because Java is increasingly used as first programming language in College.**
- **Because Java supports well OO programming, which is goodness.**
  - (but lacks templates, and other heavy-duty C++ features,...)
- **Because Java will be pervasive in the environment surrounding and driving numeric-intensive applications (GUI, client-server code, embedded software,...)**
  - Java interfaces well with Java ... less well with Fortran or C++
- **Because Java ports easily ("writes once, run everywhere")**
- **Because distributed object frameworks that can support large, complex systems are becoming available in Java**
  - (e.g., San Fransisco, Enterprise Beans).

Simple Test-Case: Matmul

```
do i = 1, m
   do j = 1, p
      do k = 1, n
         C[i][j] = C[i][j] + A[i][k]*B[k,j]
      end do
   end do
end do
```

m = n = p = 64

Matmul Performance on RS6000 590



IBM

Performance Inhibitors (1)

- **Run-time checks**
  - indices have to be checked for out-of-bound exception
  - pointers have to be checked for **NULL** exception

    An access to A[i][j] requires two pointer checks (A, A[i]) and two index checks (i, j)

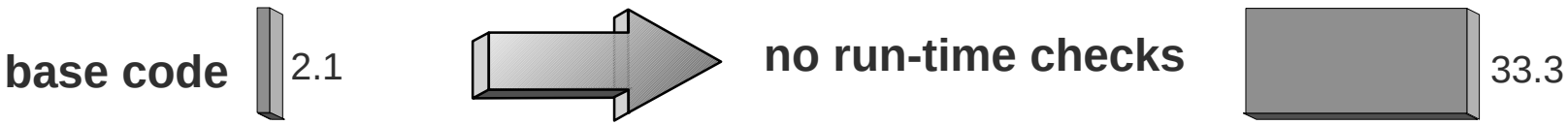
    C[i][j] = C[i][j] + A[i][k]*B[k,j]

  ➡ 8 pointer checks and 8 index checks per iteration, in a naive implementation!

  82 machine instructions in inner loop

  inhibits compiler optimizations (e.g., code motion), even if those preserve Java semantics in normal execution (precise exception problem)
- **Solution:**
  - compiler optimization
  - language enhancements (?)

    imprecise exceptions
- **Prognosis:**
  - problem largely taken care by compiler technology within 1 -- 2 years.



Array Index Checking Optimization

```
double [] a = new double[Na]
double [] b = new double[Nb]

for (i=lb, i<ub, i++)                                  run-time checks
    a[i] = b[i]
```

compiler transformation

```
for(i = lb, i<min(0, ub), i++)                         run-time checks
a[i] = b[i]

for(i=min(0,ub), i<min(Na,Nb,ub), i++)                 no run-time checks
a[i] = b[i]

for(i=min(Na,Nb,ub), i<ub, i++)                        run-time checks
a[i] = b[i]
```

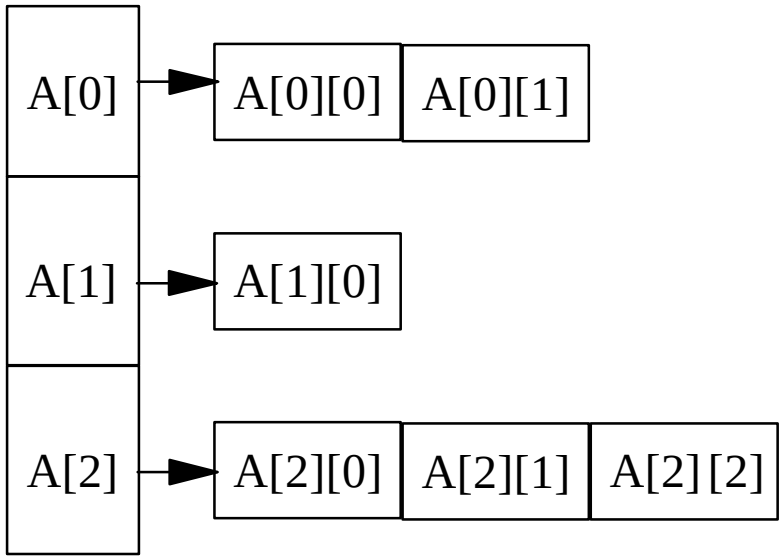
Compiler Transformation Results (PRELIMINARY)

Code version	CBC (1MB) MB/s	MATMUL (64x64) Mflops	SHALLOW (256x256) Mflops	BSOM (256x256) MCUP/s
C with Java rules	1.28	33.3	34.3	10.1
Java unoptimized	0.19	2.2	3.1	0.6
Java no checks	0.95	33.3	39.2	9.2
Java optimized	0.70	30.8	39.2	7.6
Improvement	x3.7	x14.0	x12.6	x12.7

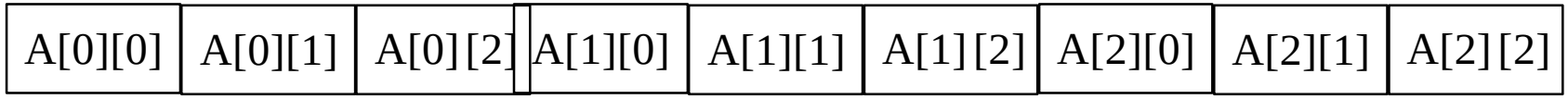
Performance Inhibitors (2)

- **Java arrrays**
  - Java multidimensional arrays are organized as arrays of arrays (can be jagged); C and Fortran support multidimensional rectangular arrays.

Java



C



Java generates less efficient code for dense, multidimensional arrays

Rectangular Arrays -- Solution

- **Ignore problem**

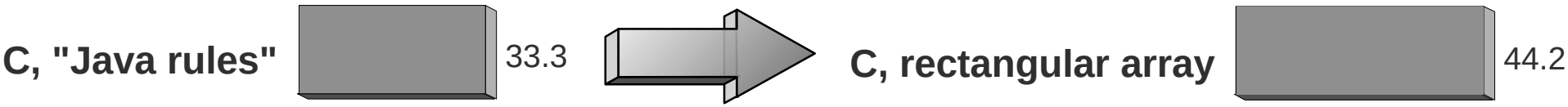
  (dense multidimentional arrays are increasingly rare in numerical codes)

- **Add to Java "true" multidimentional arrays**
  - major language change

    (significant number of new opcodes)

- **Add to Java array classes; change compilers to "inline" special classes**
  - part of general solution for "lightweight" object support in Java

- **Prognosis:**
  - Java will add mechanisms for "special-casing" predefined classes
  - Compilers will take advantage of these mechanisms for efficient support of arrays -- **assuming customers demand it.**



Matrix Package Proposal

- **Fortran-like syntax for rectangular array**

```
float[,] zero(int n, int m) {
  float a[,] = new float[n,m];
  for (int i=0; i<n; i++)
    for (int j=0; j<m; j++)
      a[i,j] = 0;
  return a;
}
```

- **A standard package of rectangular array classes**

```
FloatMatrix2D zero(int n, int m) {
  FloatMatrix2D a = new FloatMatrix2D(n,m);
    for (int i=0; i<n; i++)
      for (int j=0; j<m; j++)
        a.set(i,j,0);
  return a;
}
```

  - rectangular array syntax (a[i,j], rather than a[i][j]) is syntactic sugar; two codes are equivalent.

    Byte code is generated in both cases as if right-hand code was compiled.
    - no new opcodes; changes in Java language nice, but not essential (for performance); no changes in JVM
    - inlining of special methods (e.g., a.set(i,j,0)) is essential -- compiler technology
    - no templates: a different array class for each scalar class and rank

  - (IBM) proposal for Matrix Package is being discussed with Sun.

Overloading

```
Complex[] c = new Complex[n];
c[i] <- c[i-1] + c[i+1]
```

- same as

```
Complex[] c = new Complex[n];
c[i].set( add( c[i-1], c[i+1]))
```

- no new byte codes needed -- no change in JVM.
- compiler inlines assign and add methods
- new assignment operator to avoid object creation.
  - operator overloading and assignment operator are nice syntactic sugar

- how general do we expect this machinery to be? (only predefined packages, all final methods,...)

On the Importance of Light-Weight Objects

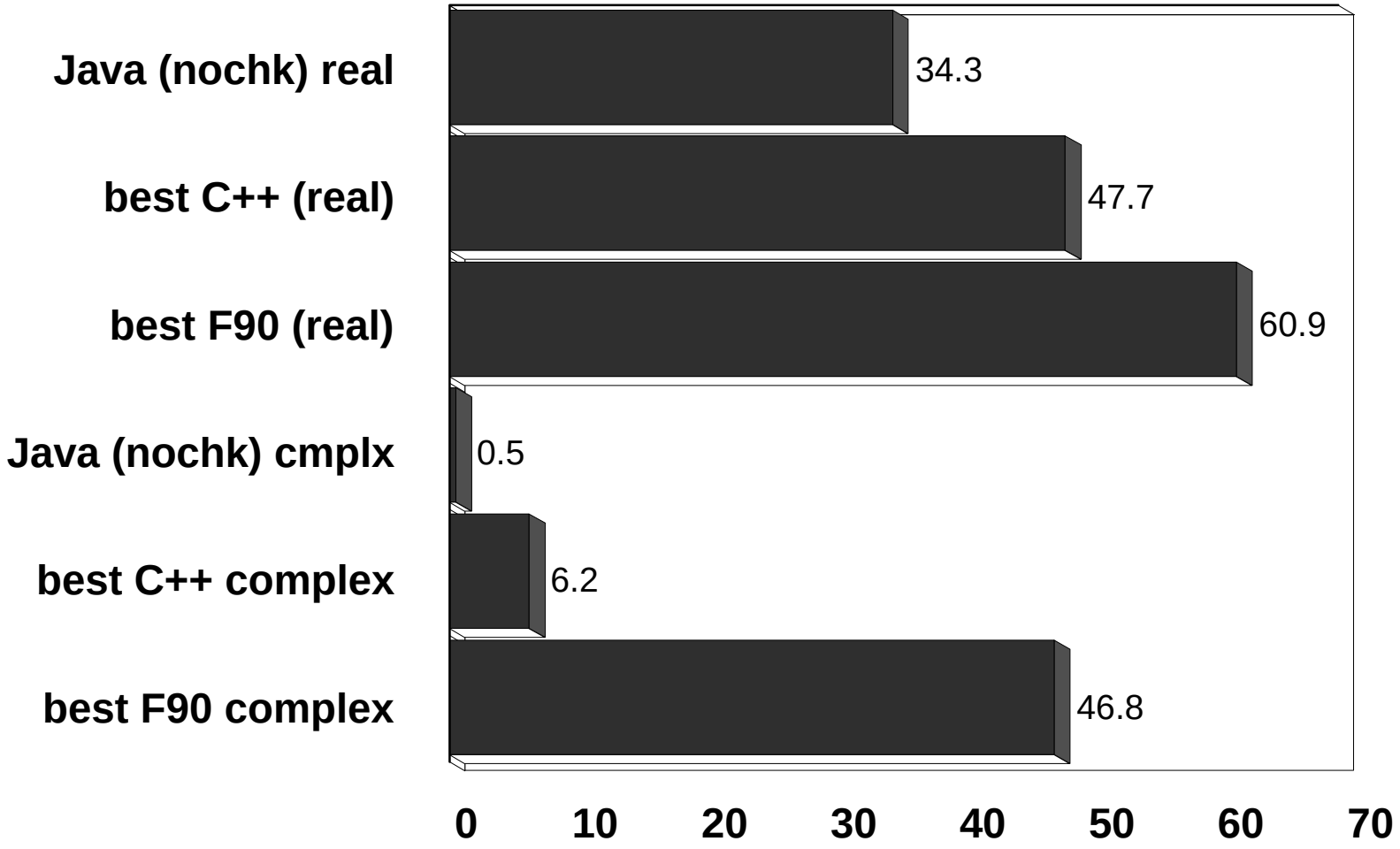
```
do i = 1, m-1
   do j = 1, n-1
      B[i][j] = 0.25*(A[i-1,j] + A[i+1][j] + A[i][j-1] + A[i][j+1])
   end do
end do
do i = 0, m
   do j = 0, n
      r = r + |A[i][j] - B[i][j]|
   end do
end do
```

n = m = 999

(i) A, B are real

(ii) A, B are complex (use user-defined Java class for complex arithmetic)

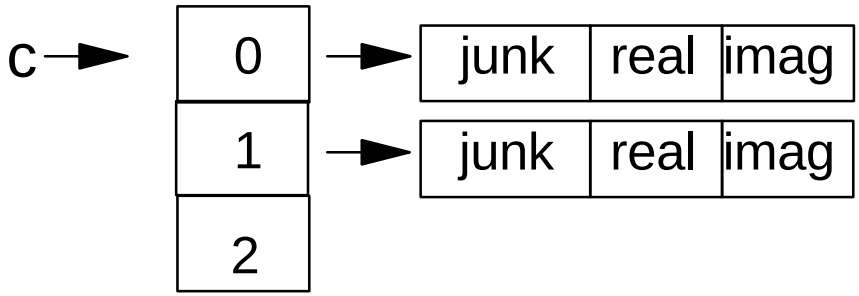
Performance on RS6000 590



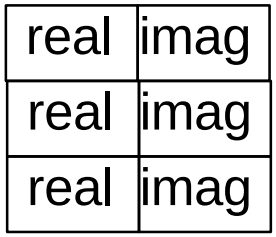
- **One method invocation per complex operation is too expensive**
- **... Is much more expensive in Java than in C++**
  - IBM proposal for standard Complex package is being discussed with Sun.

Light-weight objects (continued)

complex[] c = new complex[n]

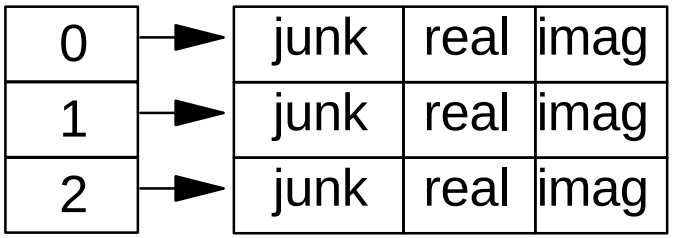


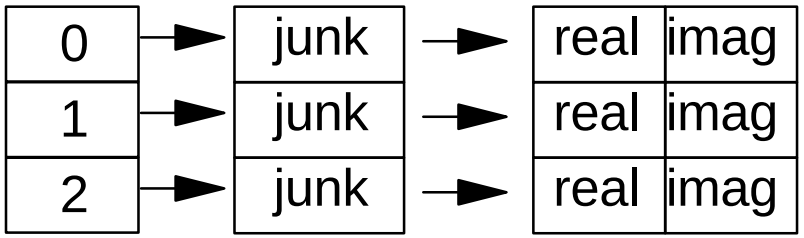
can a compiler generate, instead



?

possible alternatives





Issues: compiler analysis, run-time (garbage collection)
Helps language interoperability!

Performance Inhibitors (3)

- **Fused Multiply Add (FMA) is disallowed**
  - each intermediate result has to be rounded back to 64 (32) bits

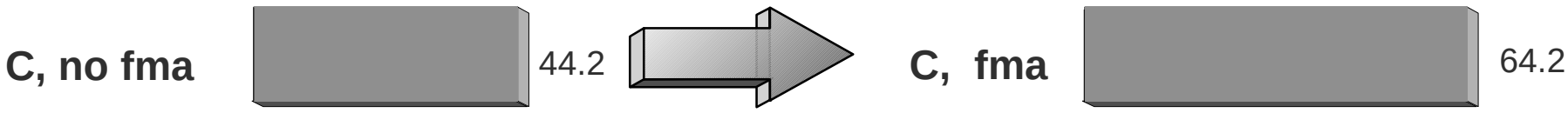
    a[i][j] = a[i][j] + b[i][k]*c[k][j]

    one fma operation

    one multiply, followed by one add, if fma is not used
  - hurts machines with fma operations
  - **hurts Pentium** (which uses 80 bit precision in FP registers)

- **Solution:**
  - relaxed fp semantics: higher precision is allowed in intermediate results

    (current proposal from Sun for modification of Java standard)
  - largely solves performance problem
  - introduces discrepancies between results computed on different platforms!



LooseNumerics

- **Sun current (?) proposal (based on Intel input): allow unrestricted use of extended precision for intermediary results.**
- **Problems:**
  - no way for user to disallow extended precision (e.g., for testing purposes)
  - must round back to 64 (32) bits whenever Java variable is assigned.
- **Alternative (IBM proposal, Gosling draft,...):**
  - methods can be tagged with LooseNumerics attribute.
  - user can specify strict or loose numeric environment for JVM.
  - compiler can freely use extended precision if method is tagged loose numerics and if user specified loose numeric environment.
    - extended precision used only when both code writer and user agree to it!

      LooseNumerics attribute is passed in byte code (with other method attributes)

- **Some issues:**
  - granularity of attribute (per class, per method, per block,...)

    smaller granularity looks nice, but may inhibit compiler optimizations
  - does attribute affect standard (Matrix, Complex) methods invoked within the scope of tagged methods?

    if not, need two version of library.

Extended Precision Arithmetic

- **How important is strict bit-by-bit reproducibility of floating point results across platforms?**
  - Not significant, from a numerical viewpoint
    well-behaved numerical codes should not misbehave if higher precision is used.
  - Significant, for testing
    strict repeatability facilitates testing and validation

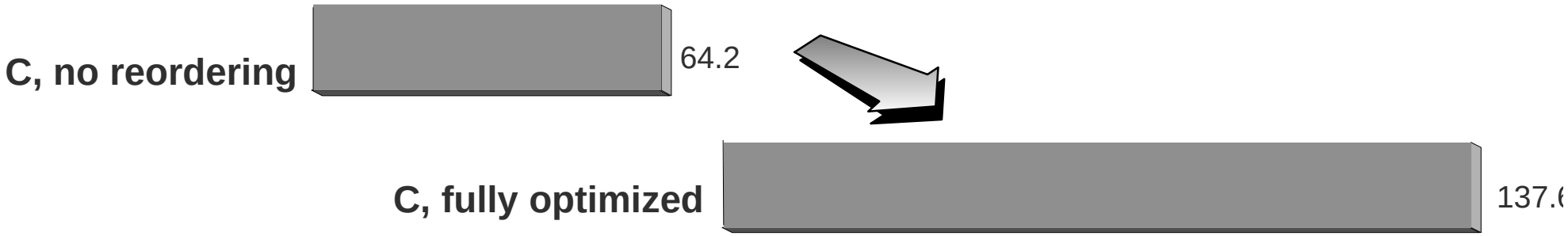
- **Opinion: use of relaxed floating point semantics should be optional**
  - codes are always tested with strict IEEE 754 arithmetic
  - user can always require strict IEEE 754 arithmetic

- **Prognosis:**
  - Use of higher-precision arithmetic will be soon legitimized in Java

Performance Inhibitors (4)

- **Operation reordering is disallowed, for two reasons**
  - precise exception
  - nonassociativity of floating point operations

    standard compiler optimizations, such as loop unrolling, are prohibited.
    - cannot take advantage of multiple floating point pipelines per CPU
    - cannot hide memory latency by software prefetching
    - cannot parallelize Java code
- **Solution:**
  - compiler optimization takes care of precise exception problem
  - language relaxation needed to allow for optimizations that take advantage of associativity
- **Prognosis:**
  - Java will provide support for code reordering



Code Reordering

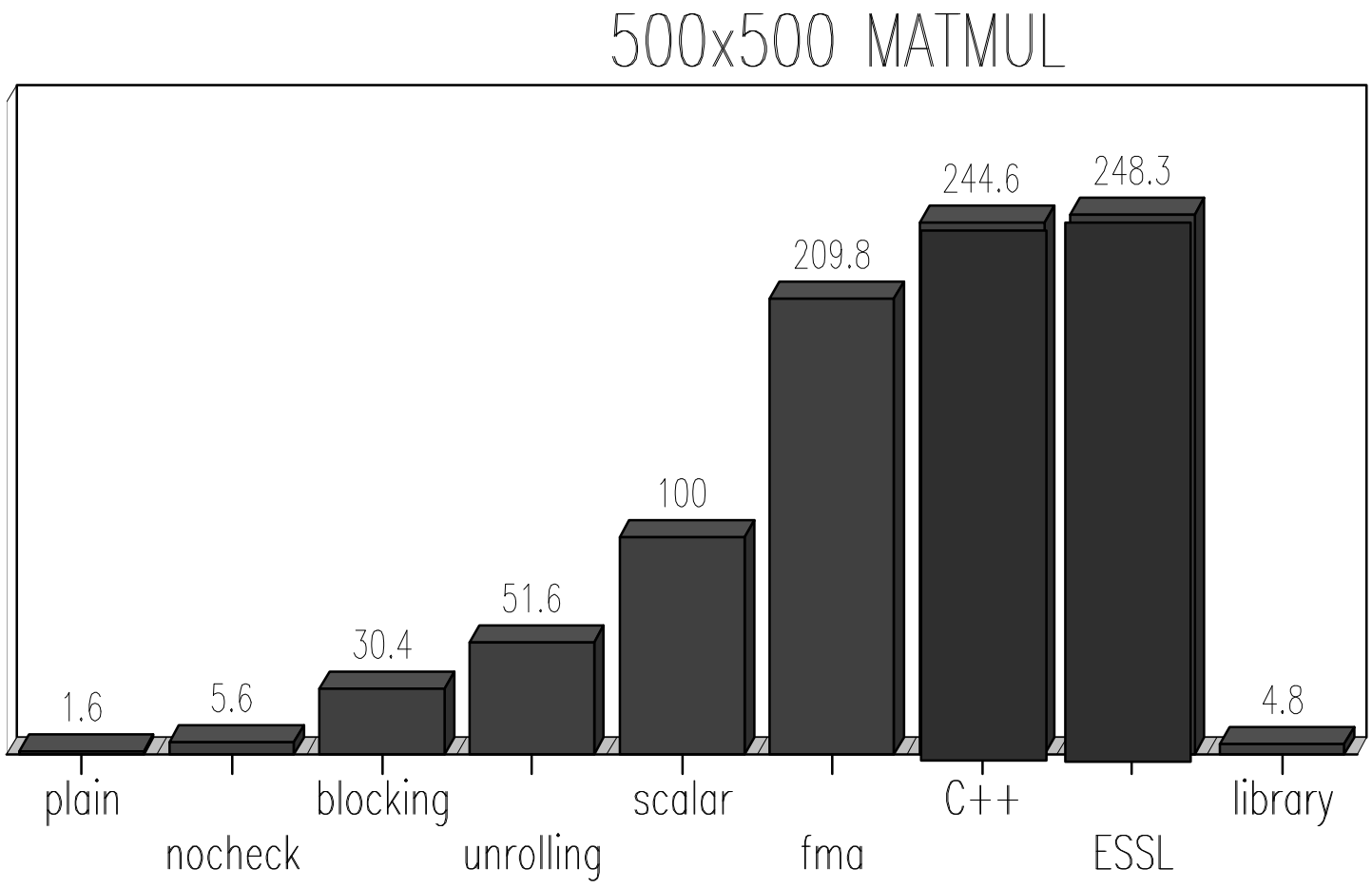
- **Example of optimization for dot-product**

$$\sum_{i=0}^{n-1} x_i y_i = \sum_{i=0}^{n/4-1} x_{4i} y_{4i} + \sum_{i=0}^{n/4-1} x_{4i+1} y_{4i+1} + \sum_{i=0}^{n/4-1} x_{4i+2} y_{4i+2} + \sum_{i=0}^{n/4-1} x_{4i+3} y_{4i+3}$$

- **Solution (IBM proposal, Gosling draft,...)**
  - IdealizedNumerics attribute. Code reordering using associativity allowed for methods tagged, with this attribute if the JVM environment allows this.
    must preserve exception behavior!
    same syntax, same implementation as for LooseNumerics
  - problems (as for LooseNumerics, only more important here):
    repeatability (solved by 2-way contract design)
    numeric instability (solved by restrictive use)
    granularity of attribute (block, method, class)
    does attribute apply to standard packages invoked within tagged scope.
  - extreme example of IdealizedNumerics optimization: can replace MATMULT code with Strassen's algorithm!
    (idiom recognition :-)

Why Fortran still faster than C (or Java)?

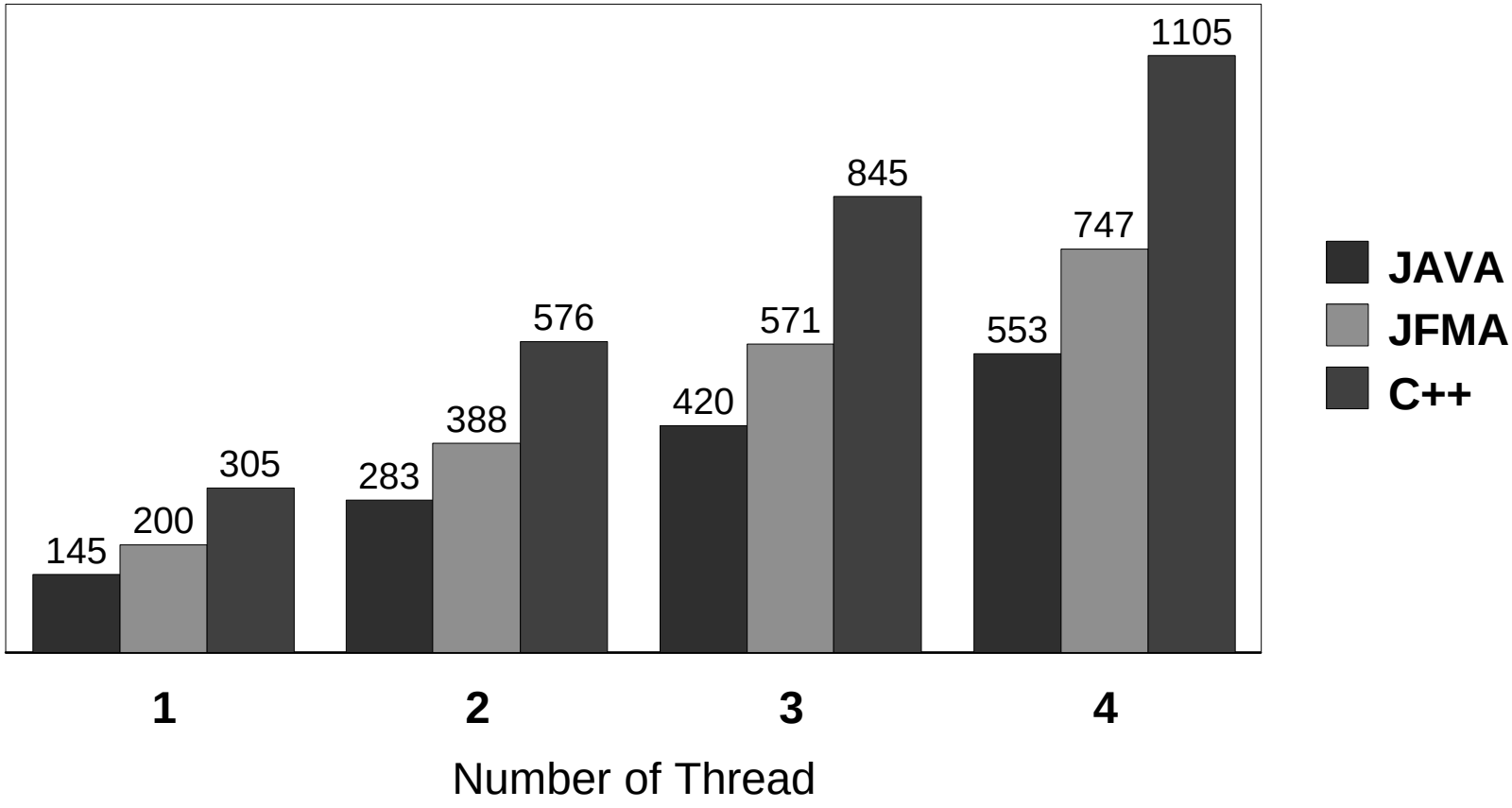
- **Nothing inherent: more aggressive high-level optimizations.**
  - will be fixed in future product releases.



library:
manual blocking and unrolling of MATMULT

Parallelizing MATMUL

MATMUL on 4w SMP (Mflops)



(parallel ESSL: 1183 Mflops)

Math Library

- **What is "right" definition of transcendental functions?**
  - current: not precise
  - 1st option: operational definition
    same outcome as reference implementation
    within given error bound of reference implementation
  - problems:
    reference implementation is arbitrary (current one is not good quality)
    error bound is not sufficient information (unbiased, exact results when possible,...)
  - 2nd option:
    "optimal result": answer is correct answer in infinite precision arithmetic, rounded to nearest value.
  - potential problem: performance
    - It seems possible to have optimal result with modest overhead!
      Ongoing implementation effort (IBM Haifa) to demonstrate performance.

Summary

- **Following enhancements are important for Java floating point performance:**
  - LooseNumerics and IdealizedNumerics
    with overide option
  - Standard packages (Matrix, Complex)
    with compiler technology for inlining
  - Compiler optimizations
  - Compiler optimizations
  - Compiler optimizations