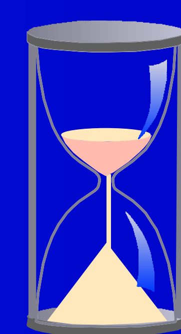
High Performance Numerical Computing in Java: Language and Compiler Issues

P.V.Artigas - M. Gupta - S.P.Midkiff - J.E.Moreira
IBM T.J. Watson Research Center



Java for Numeric Computing

- Simple, Object Oriented language
- Growing Number of Programmers
- Easier to Maintain and Debug
- BUT performance of current commercial environments is inadequate!

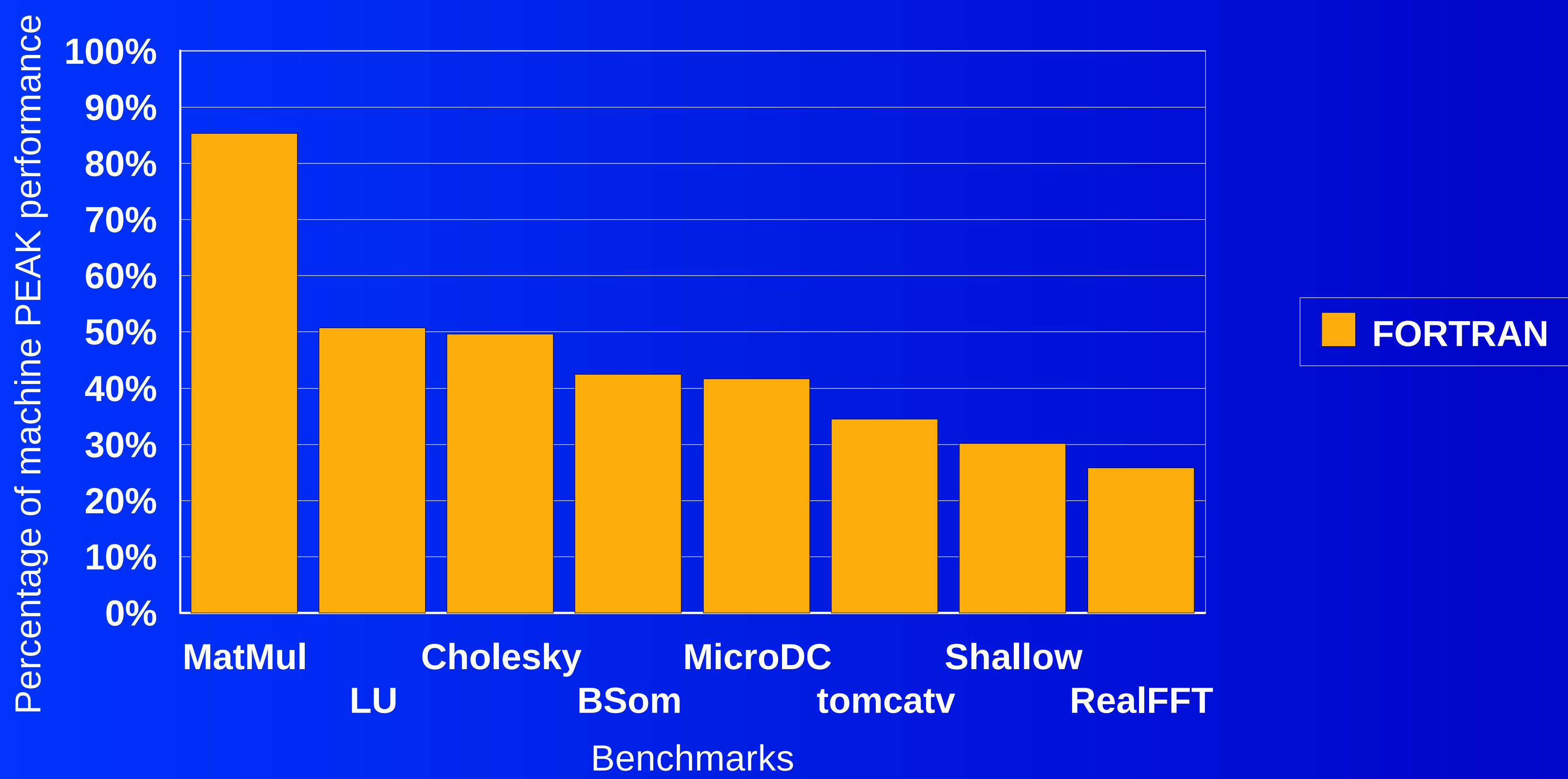
How did we benchmark?

- Want to compare equivalent FORTRAN and Java versions
- Developed the benchmarks using a third language that is automatically translated to FORTRAN and Java
- AIX port of IBM JDK 1.1.8 + JIT is currently the fastest JVM for our platform. IBM JDKs generally recognized as very fast in many platforms
- IBM XLF 6.1 FORTRAN compiler
- RS/6000 590 Workstation used, 67Mhz POWER II processor, 266 MFlops peak performance, 256Kb L1 data cache

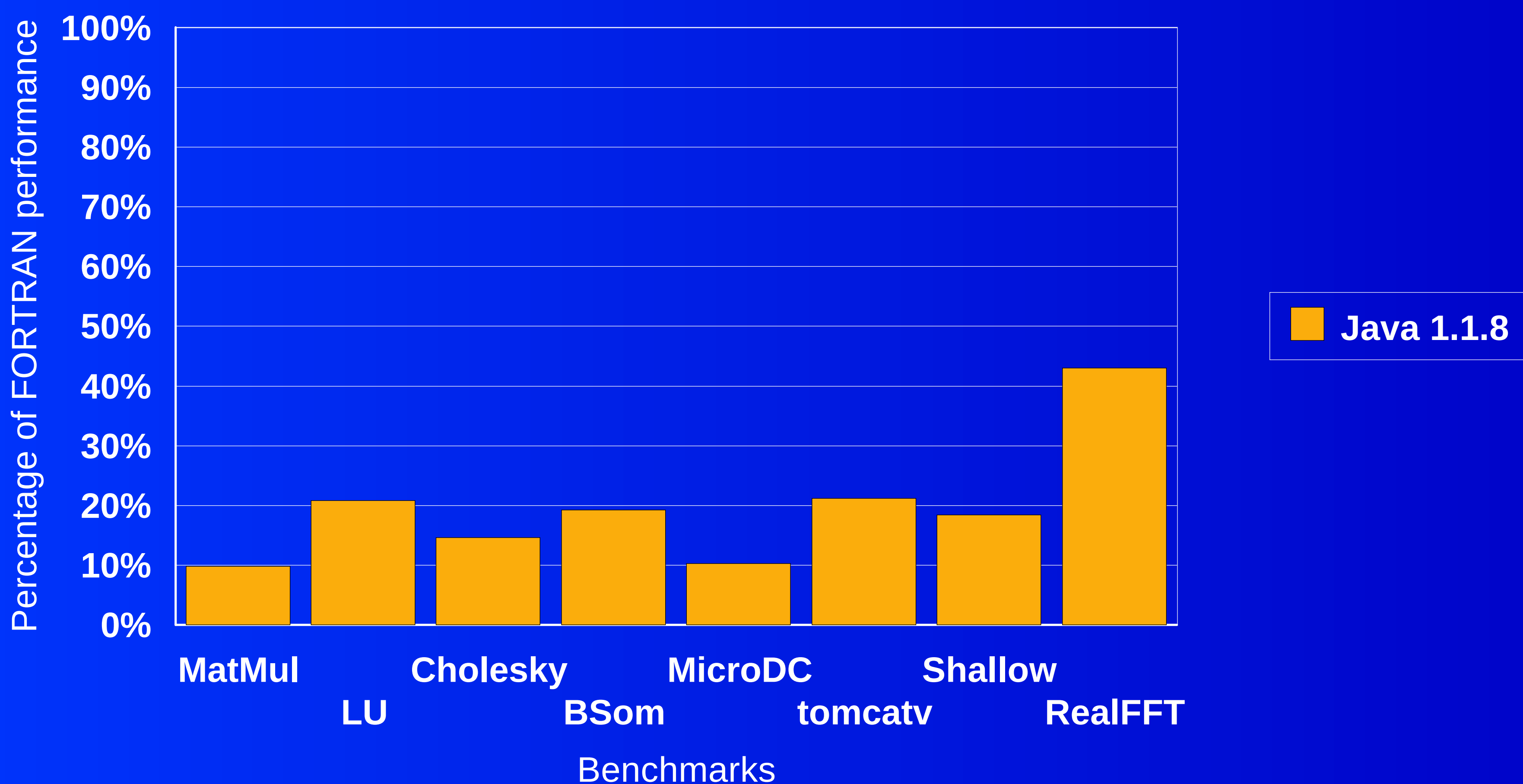
What did we benchmark?

- 8 Programs that represent most classes of dense numerical computing algorithms
- MATMUL - Blocked Matrix Multiply Kernel
- LU, CHOLESKY - Direct Equation Solvers
- BSOM - Data Mining Kernel
- MICRODC - Relaxation Solver
- TOMCATV - Mesh Generation, part of specFP95
- SHALLOW - Shallow water simulation
- REALFFT - 2D FFT, real samples

Do these benchmarks stress the hardware?



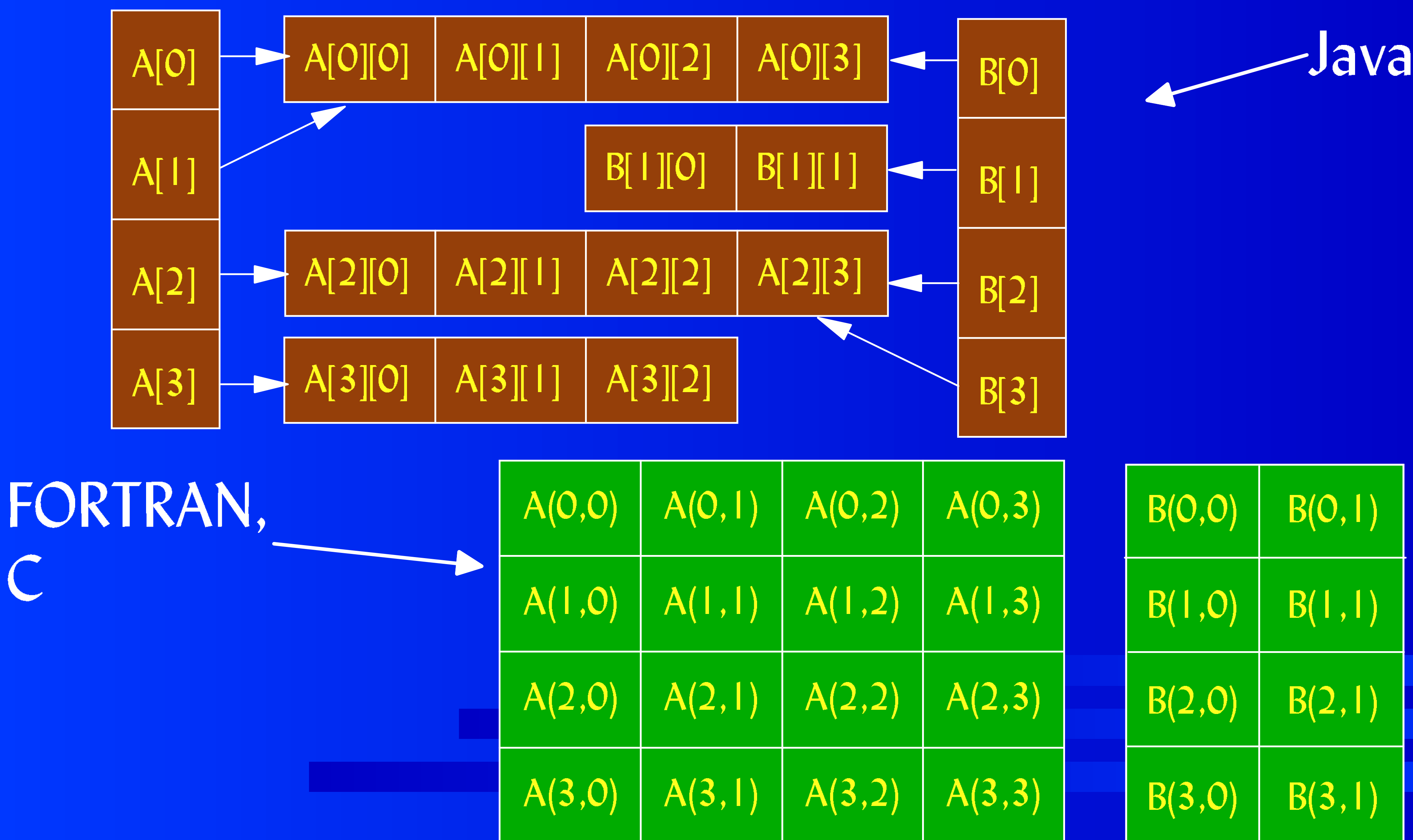
Current Performance of Java



Why Java performance so low?

- As shown in previous LCPC98 paper, bounds and null-pointer checking are expensive; when combined with precise exceptions they inhibit code reorder
- Good alias disambiguation important for optimization, difficult with Java array structure
- Java does not use available floating point hardware efficiently (`fma`s and 80bit FP)
- Creation of safe regions effective in enabling optimizations, but not easy to determine array bounds, alias information, and guarantee thread safety with Java arrays

Java versus Other language multidimensional arrays



How can we add true multidimensional arrays to Java?

- We do not want to violate the Java specification and spirit
- We do not want to violate the JVM specification (byte codes can not directly represent true multidimensional array indexing operations)
- We do not want to create a new language
- Solution: Use the language provided mechanisms to extension: sets of classes and packages!

The Array package for Java

- A set of classes representing true, rectangular, multidimensional arrays and associated functionality
- Also include the functionality expected by numerical programmers (Array sections, transposing, reshaping, etc.)
- Provide libraries expected by numerical programmers (for now, BLAS)
- Preserves safety and security features of Java (extensive bounds/consistency checks)

Array package design principles

- 100% Java implementation
- Most classes are final, to statically bind semantic to syntax
- One class per type and rank (**`doubleArray2D`**, **`ComplexArray3D`**) for good code generation and optimization
- Internal structure is not exposed, more ambitious optimizations are possible
- Transactional array semantics: operations complete successfully or fail before changing any data

Operator overloading would help

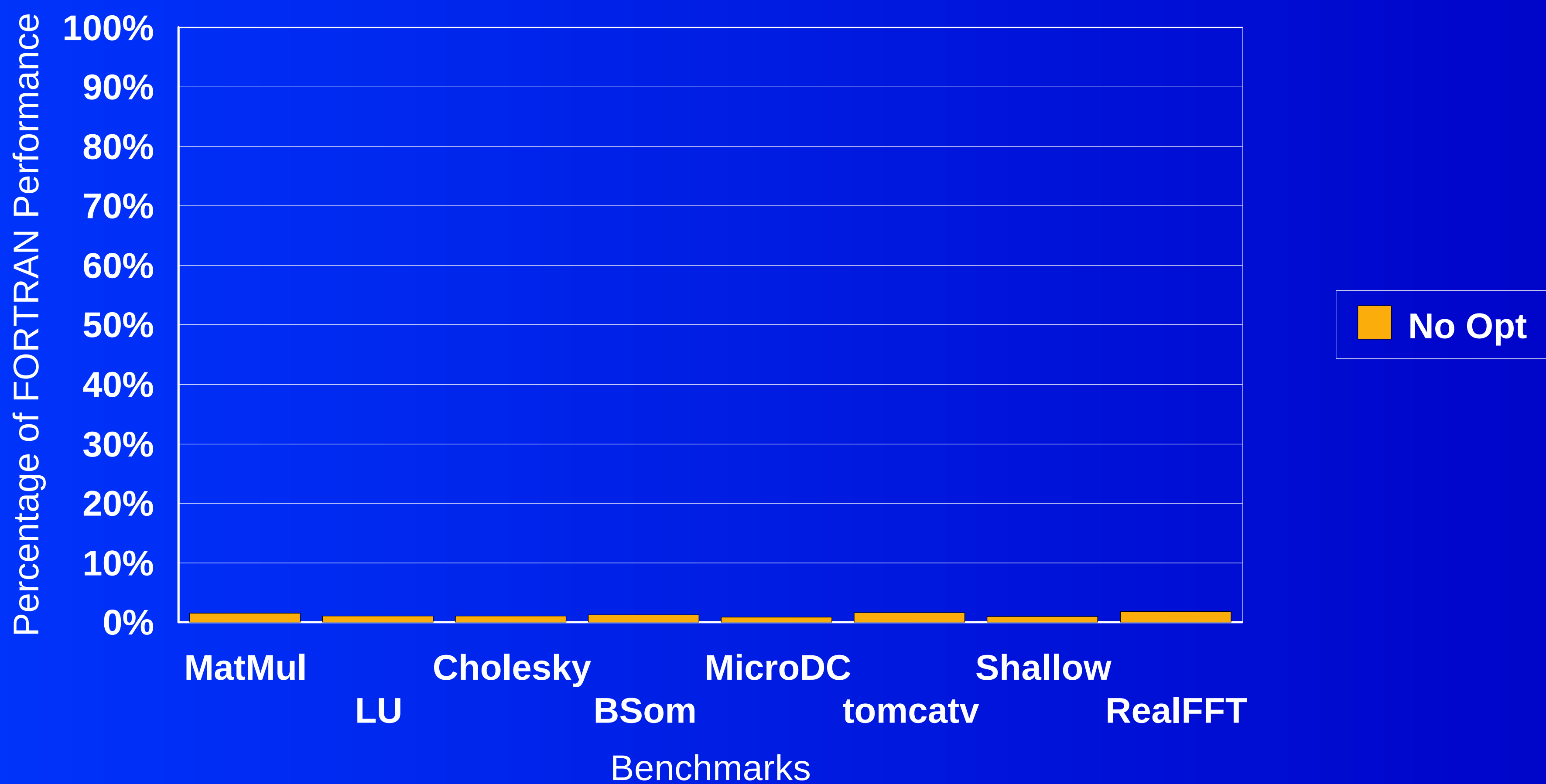
Example of a relaxation code:

```
doubleArray2D b = new doubleArray2D(h+1,w+1);
doubleArray2D a = new doubleArray2D(h+1,w+1);

for(j=1;j<=h-1;j++)
 for (i=1;i<=w-1;i++)
  b.set(j,i,0.25*(a.get(j,i+1)+a.get(j,i-1)+
                 a.get(j+1,i)+a.get(j-1,i)));
```

OR at the Java source level (same bytecode):

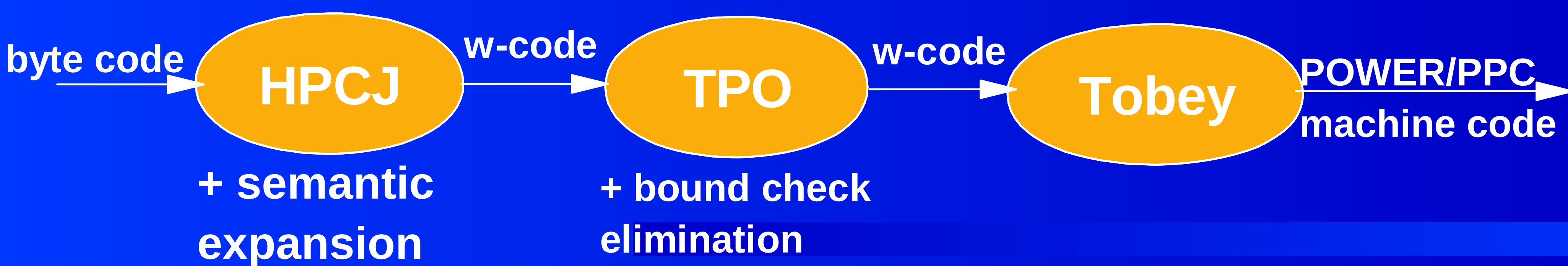
```
for(j=1;j<=h-1;j++)
 for (i=1;i<=w-1;i++)
  b[j,i] = 0.25*(a[j,i+1]+a[j,i-1]+a[j+1,i]+a[j-1,i]);
```

The Array package for Java performance



Our prototype research Java compiler

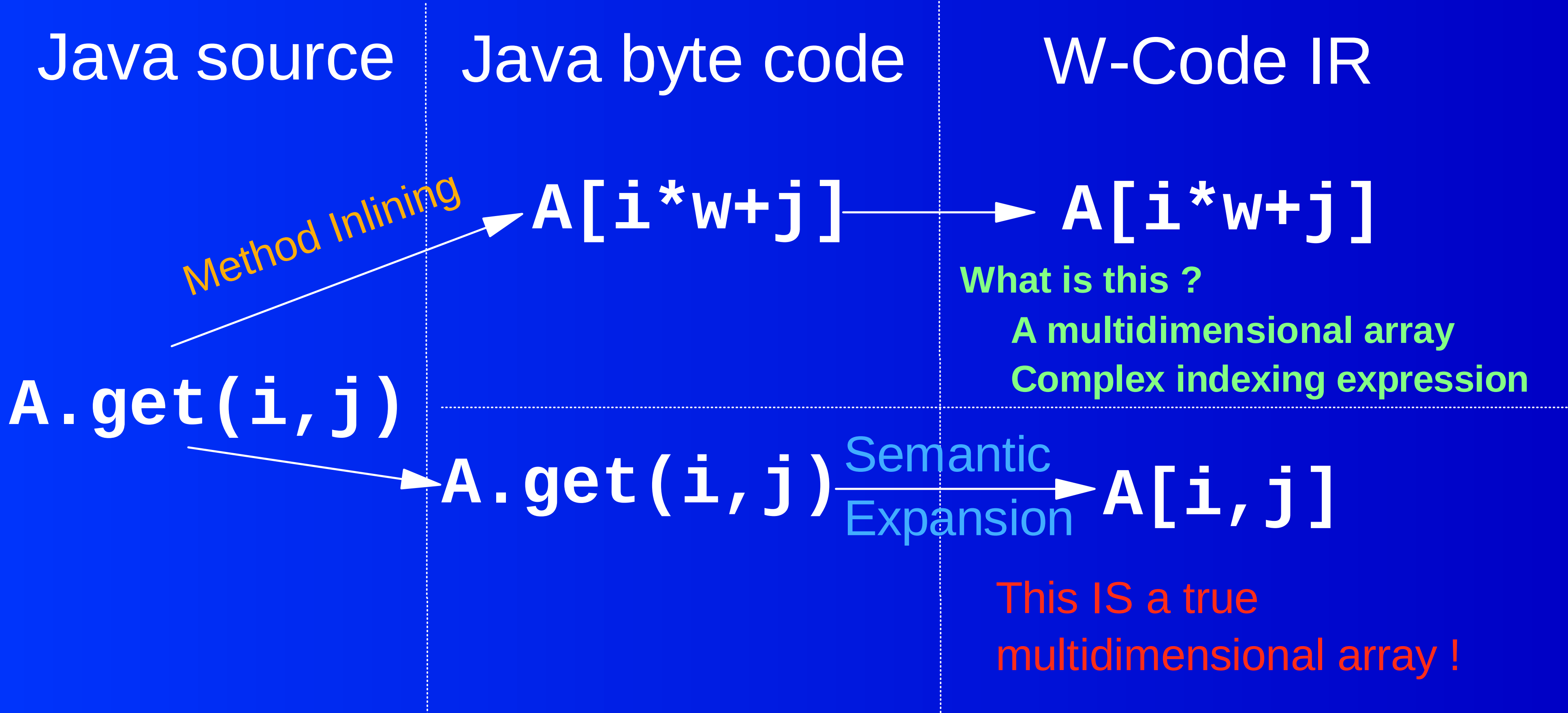
- Static Java compiler based on the IBM XL family of compilers
- Optimizations required for good Array package performance developed (codesign!)
- Use existing compiler infrastructure



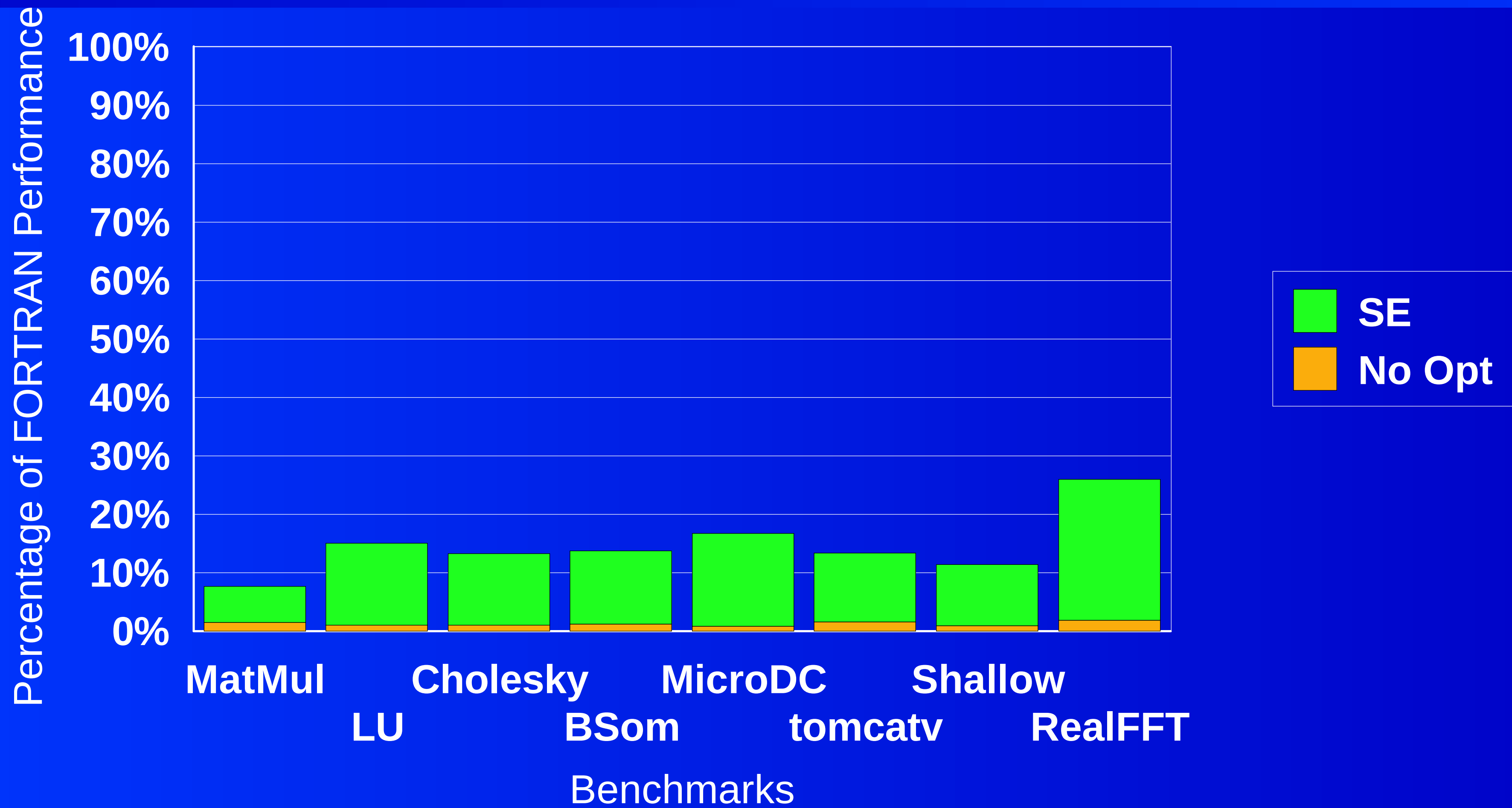
Semantic expansion of Array package methods

- The **`get()`** and **`set()`** accessor methods are semantically expanded by our HPCJ front end
- semantic expressed using the w-code IR, code expanded inline replaces method invocation
- w-code IR is able to express the semantics of true multidimensional arrays, no need to add bytecodes to the JVM
- Java precise exceptions model preserved, w-code expanded inline contains exception checks, as expected for arrays
- Refer to Java Grande 99 (Wu, *et. al*) for detailed discussion of semantic expansion

Semantic Expansion versus Regular Method Inlining



Performance of Array package with semantic expansion enabled



Exception Checks Limit Performance

- We need an optimization that reduces drastically the cost of exception checks
- We concentrate our efforts on loop regions, those are pervasive in numeric computing
- We try to reduce the dynamic exception check count
- Large exception free regions are needed to enable high order transformations, which leads to large speedups

The bounds checking optimization

- Array range analysis used to find ranges of array indexes referenced in loop regions
- Information used to create 2 versions of loop nests, one with exception checks, the other without, run time test will decide which will execute
- If the safe (exception free) regions dominate the execution time performance will be competitive

Bounds Checking Optimization, an example

MICRODC kernel (exception checks omitted in this slide)

```
for(j=1;j<=h-1;j++) /* 1 to h-1 */
  for (i=1;i<=w-1;i++) /* 1 to w-1 */
    b[j,i] = 0.25*(a[j,i+1] + a[j,i-1] +
                   a[j+1,i] + a[j-1,i])
```

Array section analysis summary:

Inner loop summary: **`a[j-1:j+1;0;w],b[j:j;1:w-1]`**

Outer loop summary: **`a[0:h;0:w],b[1:h-1;1:w-1]`**

Code generated

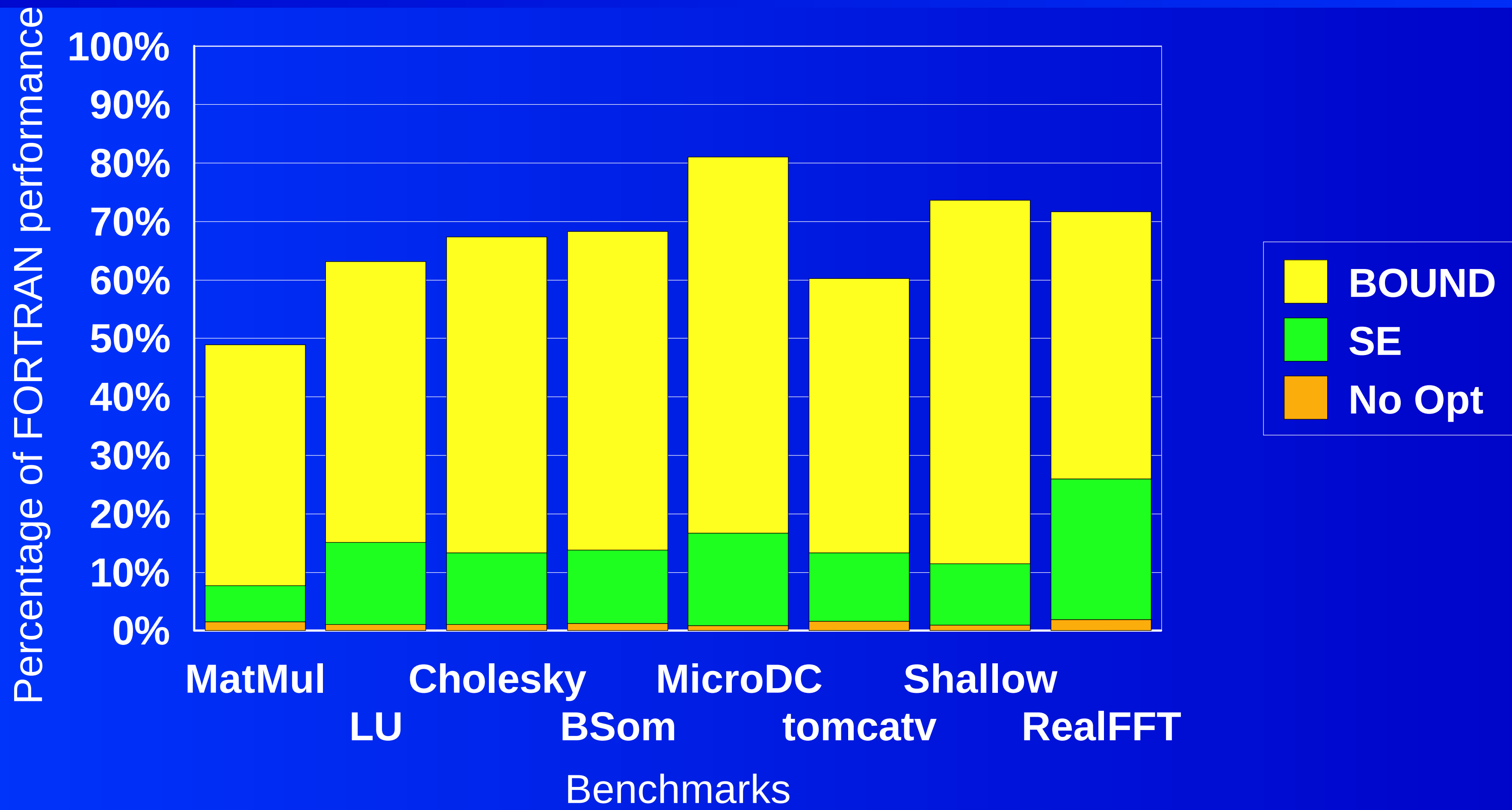
```
if ((a!=null)&&(b!=null)&&
   (h<a.size(0))&&(h-1<b.size(0))&&
   (w<a.size(1))&&(w-1<b.size(1)))
{

 for(j=1;j<=h-1;j++) /* 1 to h-1 */
  for (i=1;i<=w-1;i++) /* 1 to w-1 */
   b[j,i] = 0.25*(a[j,i+1] + a[j,i-1] + a[j+1,i] + a[j-1,i]);

} else {

 for(j=1;j<=h-1;j++) /* 1 to h-1 */
  for (i=1;i<=w-1;i++) /* 1 to w-1 */
   CHKn(b)[CHKb(j),CHKb(i)] = 0.25* (
    CHKn(a)[CHKb(j),CHKb(i+1)] + CHKn(a)[CHKb(j),CHKb(i-1)] +
    CHKn(a)[CHKb(j+1),CHKb(i)] + CHKn(a)[CHKb(j-1),CHKb(i)]
   );

}
```

run time test

safe region, no checks, reordering OK

unsafe region, checks, no reordering

Performance with bound check optimization



Is the bounds optimization responsible for all that speedup?

- NO!
- optimizations present in TPO and TOBEY now allowed in exception free regions
- In numeric computing the large exception free regions dominate execution time
- Performance is now competitive with other numeric languages

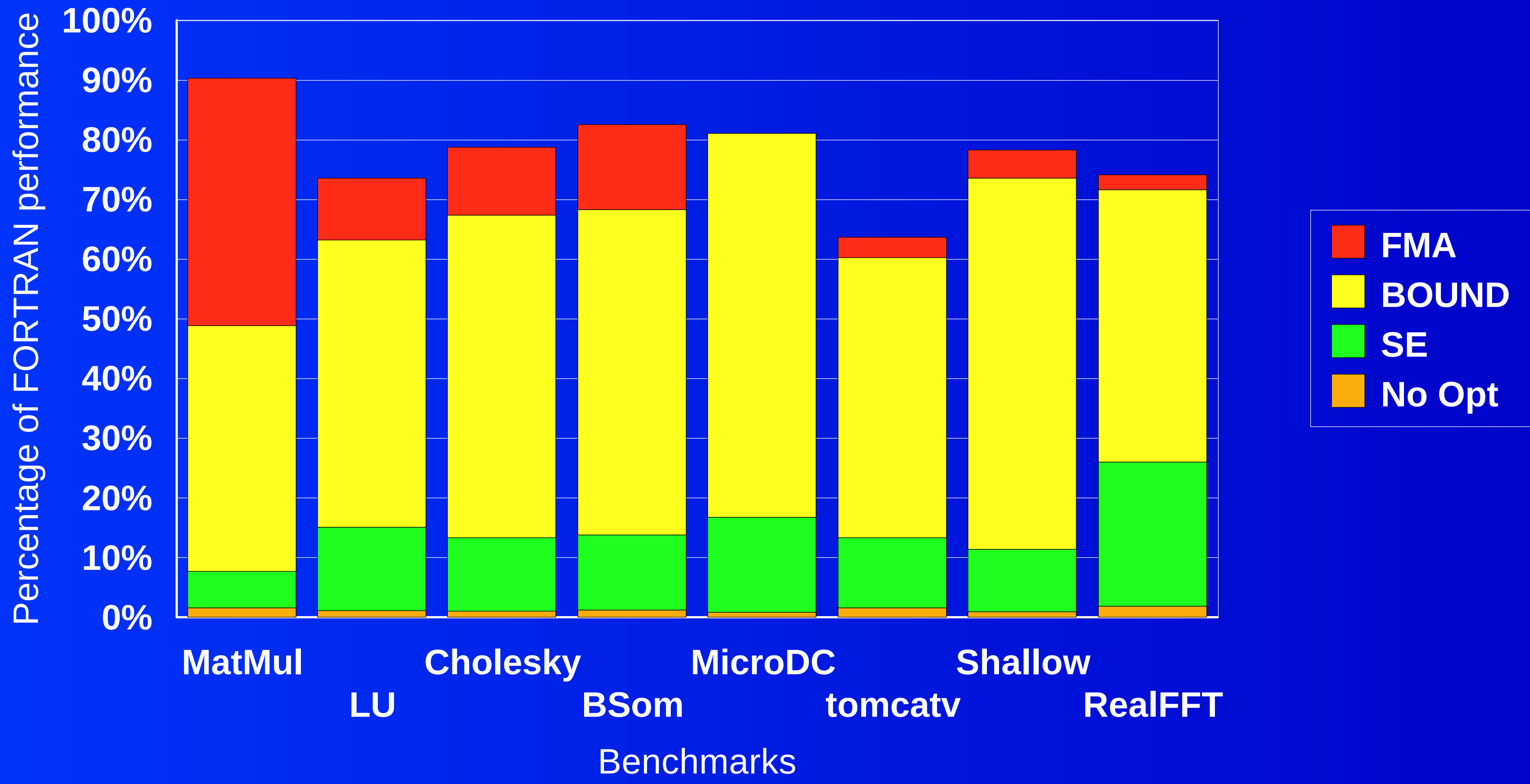
Array package advantages for bounds checking optimization

- Java arrays of arrays would require pointer chasing to recognize the complete array data structure
  OR
  The optimization would only be performed for the last axis
- Privatization of array data structures would be needed, given the multithreaded nature of Java, to allow the creation of large safe (exception free) regions
- None of these is a problem for Array package Arrays

What else is missing?

- **`fma`**s, Floating-point multiply-add, existing in POWER/PowerPC, MIPS (MIPS IV and above), PA-RISC (v1.1 and above) and IA-64 are not allowed
- **`fma`** computes a*b+c("infinite" precision) and round
- Extended precision is used in the intermediate product, final result is more precise, therefore disallowed
- Proposals to allow **`fma`**s in Java are being considered

What do `fmas` buy us?



The `fma` benefit

- All the benchmarks that are closer to peak performance require the **`fma`**s to reach near-to-FORTRAN performance
- In others the bottleneck is not the computation or **`fma`**s are not used at all
- Improper alias information limits memory bandwidth in lower performing benchmarks
- Also array organization information is not propagated all-the-way to the backend as in FORTRAN

Conclusion

- High performance numeric codes can be developed in Java, there is nothing fundamental that limits the Java performance for numeric computing!
- Language and compiler codesign approach provided good performance
- Usability features required to attract existing numeric programmers
- New programmers already used to Java

The role of language/compiler codesign

- For aggressive compiler optimization we need regions of code that are free of exceptions and aliases
- Designed Array package to facilitates bounds checking optimization and alias disambiguation
- Semantic expansion necessary to make Array package viable from performance perspective

Future Work

- Better alias disambiguation for Array package arrays in order to enable paralelization and other high order transformations
- If there is not enough compile-time alias information use Array package built in dependency test at run time and versioning

Any questions?