Java Performance and Compiler Optimizations

Ninja project: M Gupta, S Midkiff, J Moreira

Marc Snir

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

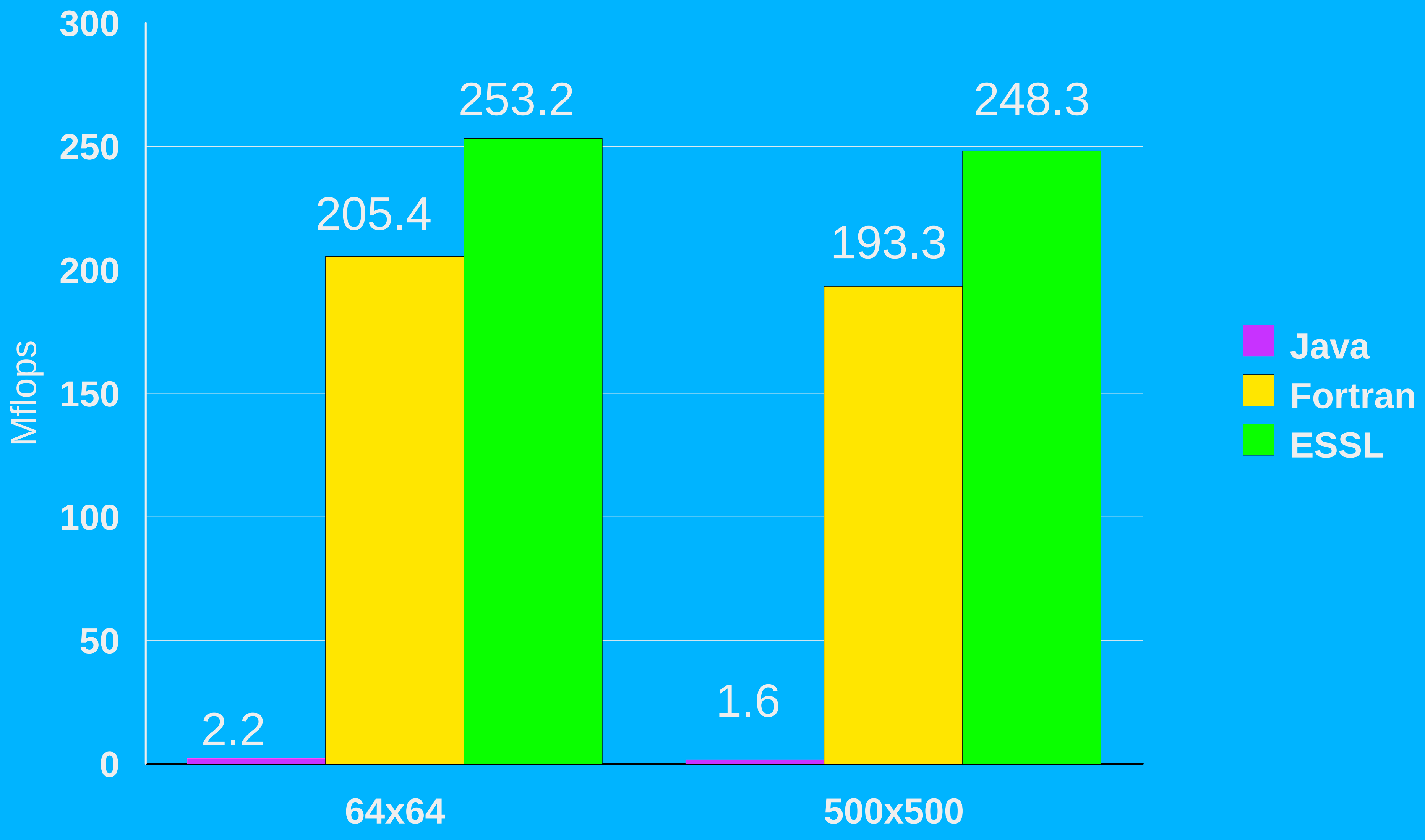
Nov 98

Simple matrix-multiply loop

```
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];
```

How is Java doing?

MATMUL results on RS/6000 590



- Both Java and Fortran are compiled code; most of the difference is in compiler optimization

Fortran optimizations for Java?

- **Not done because**
  - Has not been a priority of Java compiler developers (main reason!)
  - Java language design prevents it (see Java Grande proposals)
  - New compiler techniques are needed
  - ▸ The possibility of exceptions in Java prevent applying Fortran-style optimizations:
  - blocking, for better memory behavior,
  - multiple loop unrolling,
  - loop interchange,
  - scalar replacement.
  - ▸ Safe regions allow aggressive optimization within Java semantics.

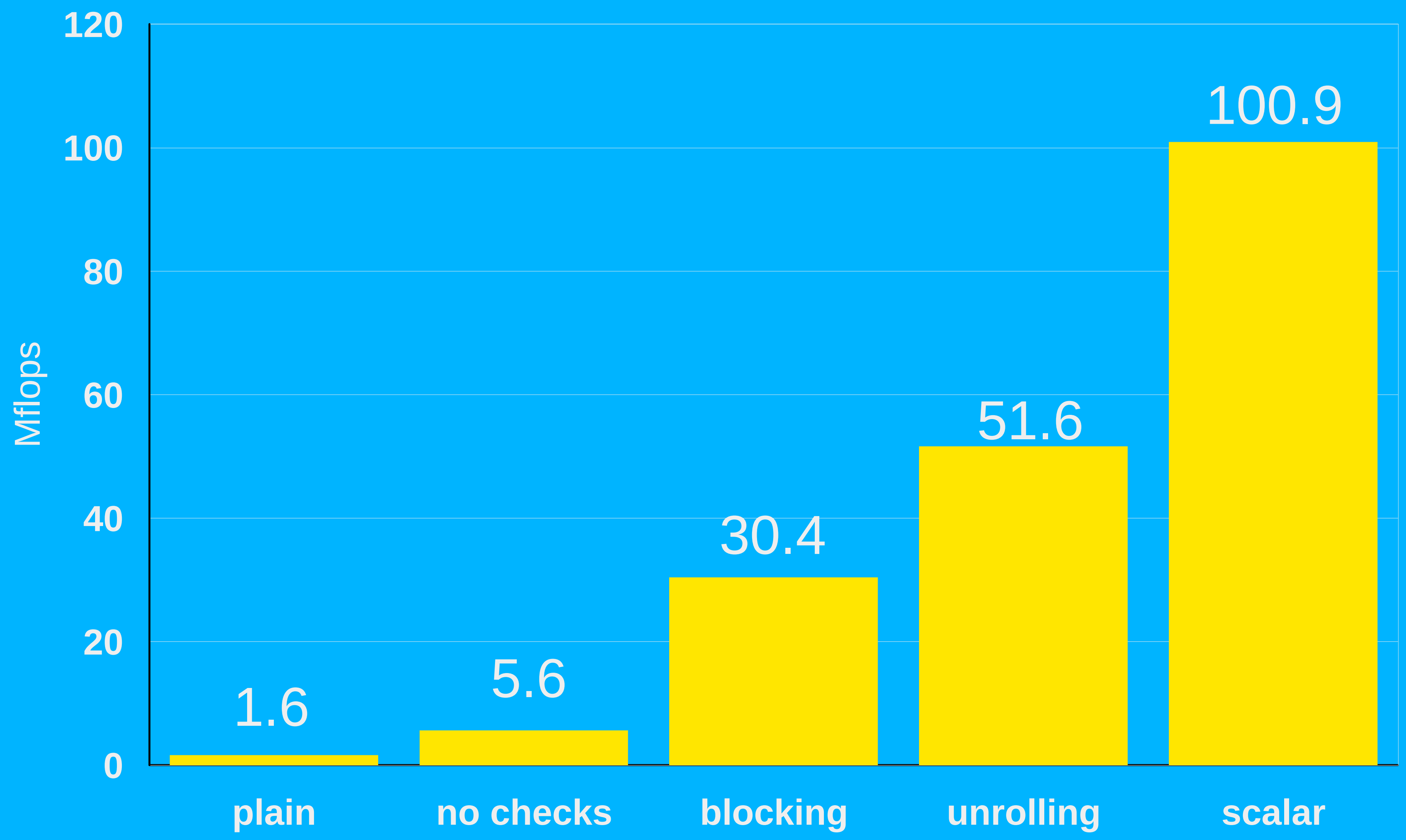
Safe regions through versioning

```
if ((m <= Crows) && (p <= Ccols) &&
    (m <= Arows) && (n <= Acols) &&
    (n <= Brows) && (p <= Bcols))
  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
  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];
```

safe region
index checks disabled
ops can be reordered

unsafe region
index checks enabled

500x500 MATMUL Optimized

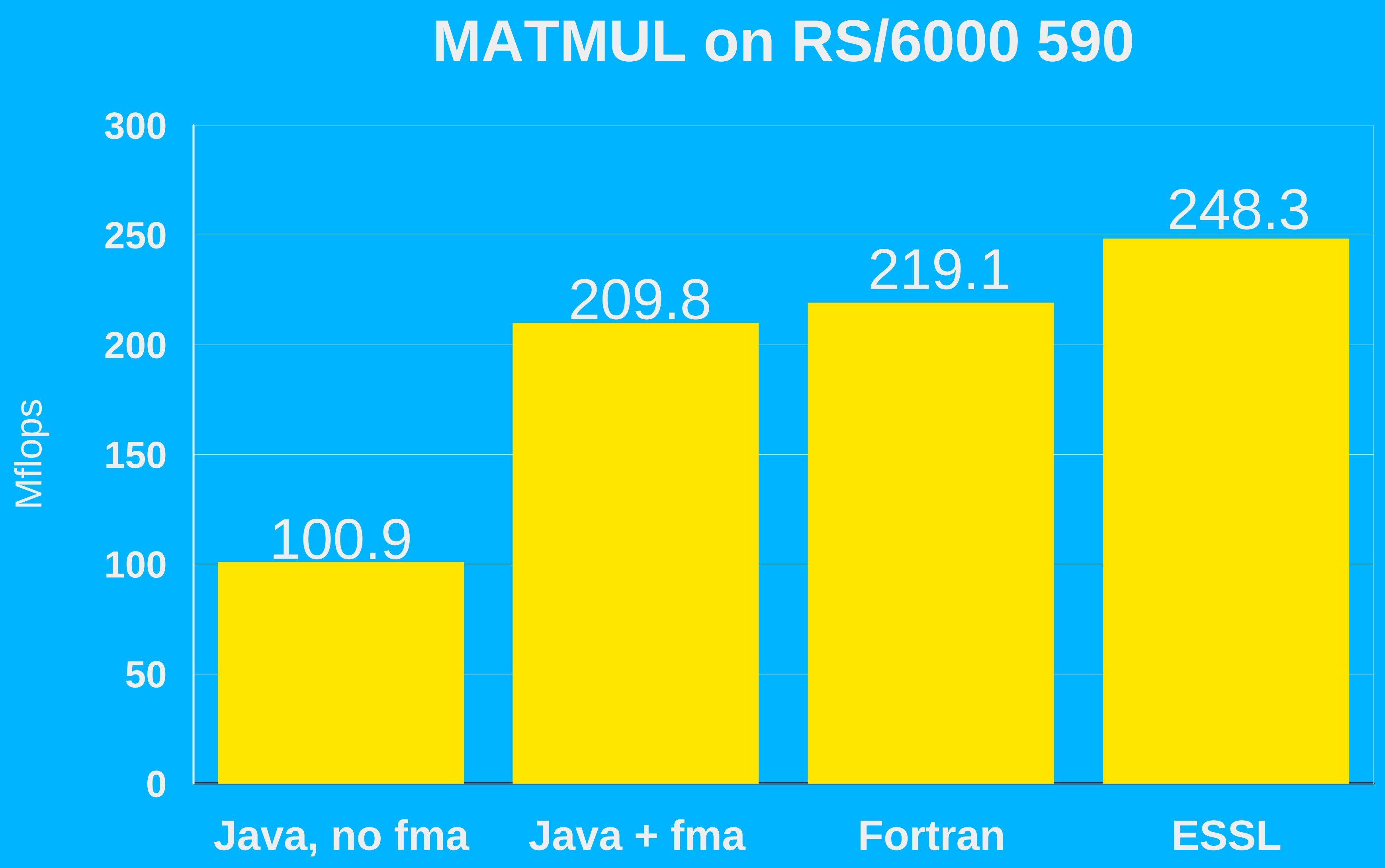


Marc Snir

Benefits of fused multiply-add

- **A fused multiply-add (fma) operation computes a*b+c with a single rounding**
- **Java cannot use fma today**
  - changes outcome (even if more precise)
    - In the POWER/PowerPC families, this cuts the peak performance in half!
- **fma will be legal in Java if Java Grande proposals are adopted**
  - also legal according to alternative proposals being discussed

Impact of allowing fma in Java

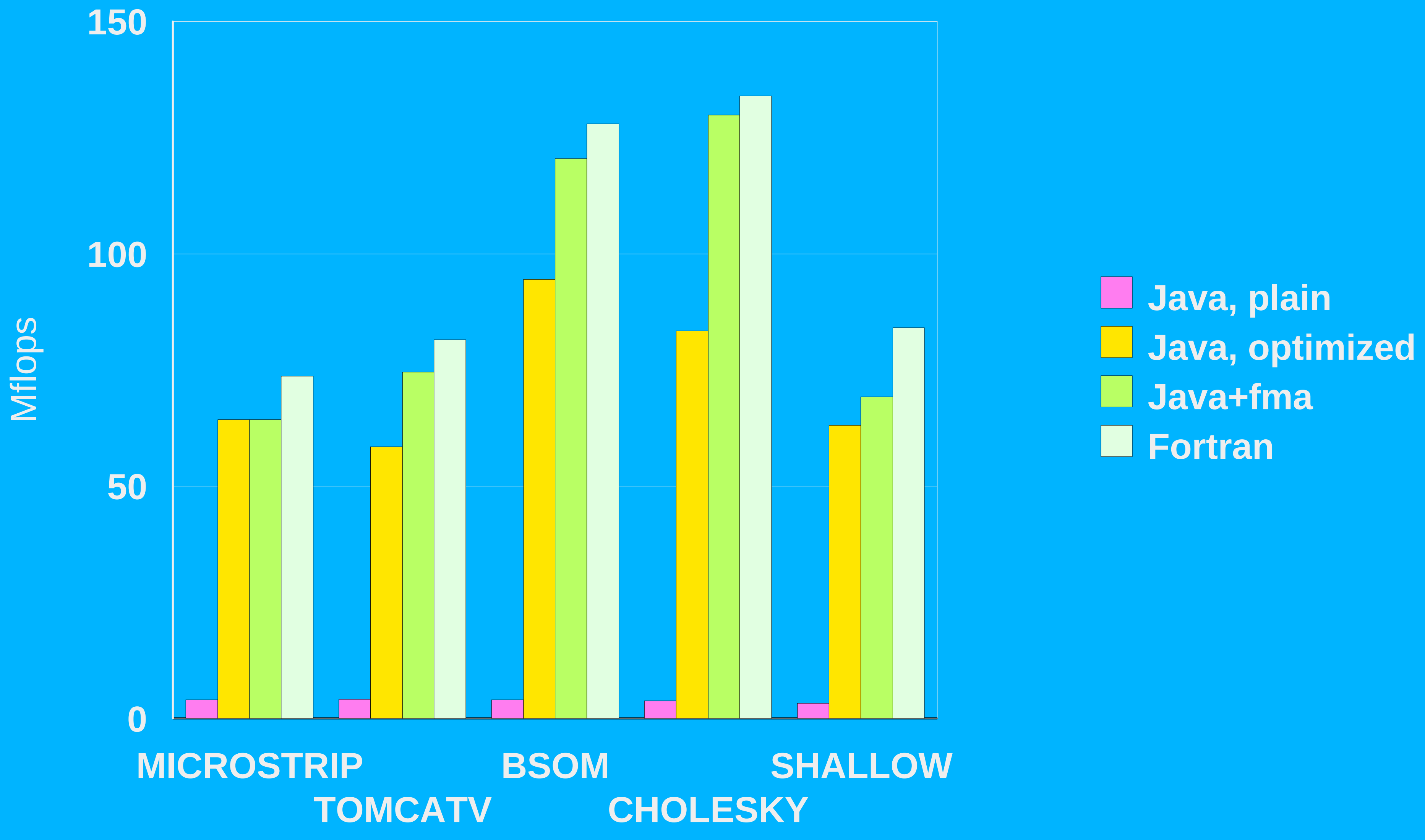


Marc Snir

Other benchmarks

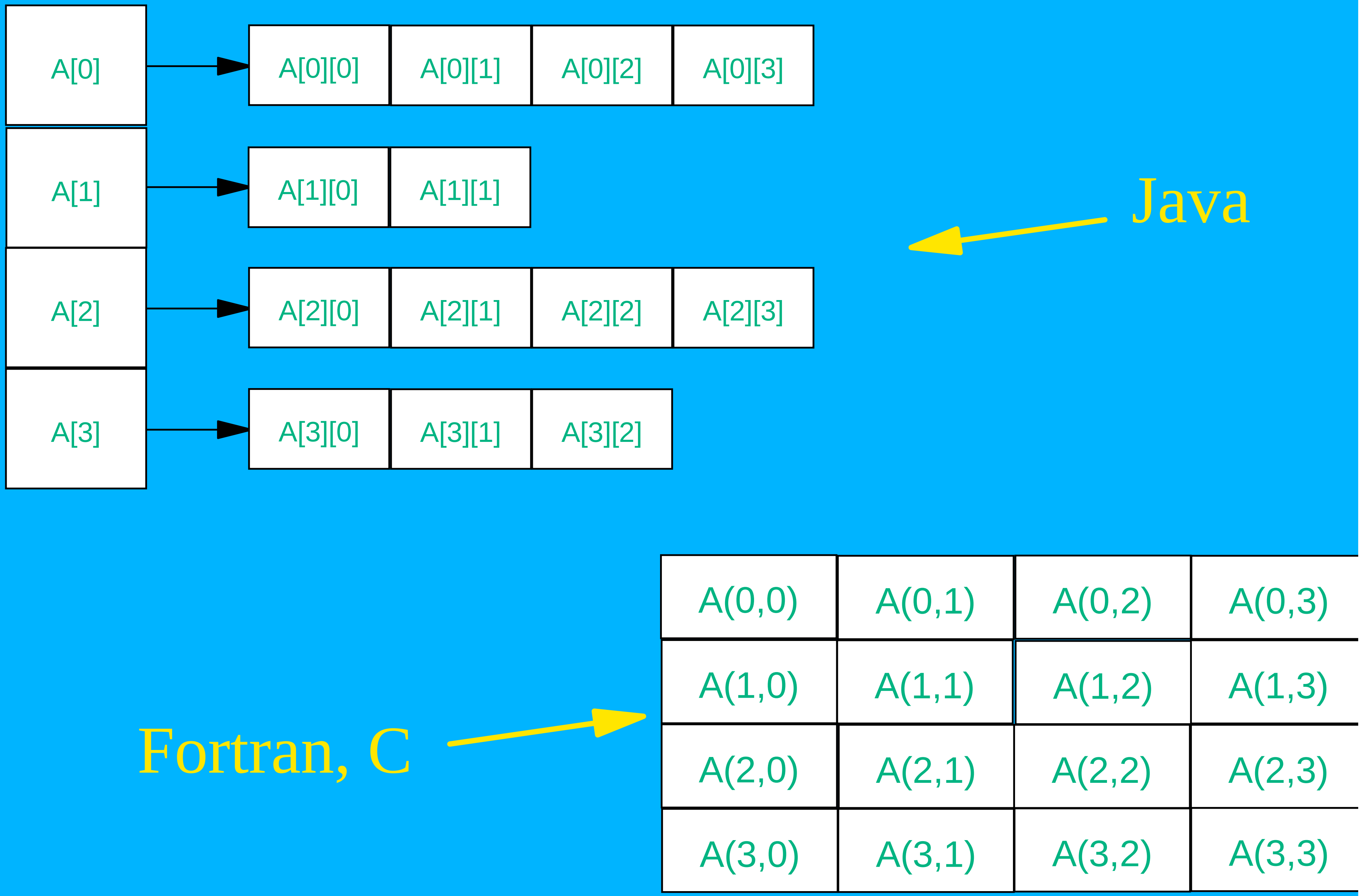
- MICROSTRIP: electrostatic potential computation (1000x1000 grid).
- TOMCATV: mesh generation, solver (513x513 mesh).
- BSOM: data mining neural-network training (16 nodes).
- CHOLESKY: Cholesky factorization (1000x1000 dense matrix).
- SHALLOW: shallow water simulation (256x256 grid).

Results on RS/6000 590



Marc Snir

Array layouts



A[0] A[0][0] A[0][1] A[0][2] A[0][3]

A[1] A[1][0] A[1][1]

A[2] A[2][0] A[2][1] A[2][2] A[2][3]

A[3] A[3][0] A[3][1] A[3][2]

Java

Fortran, C

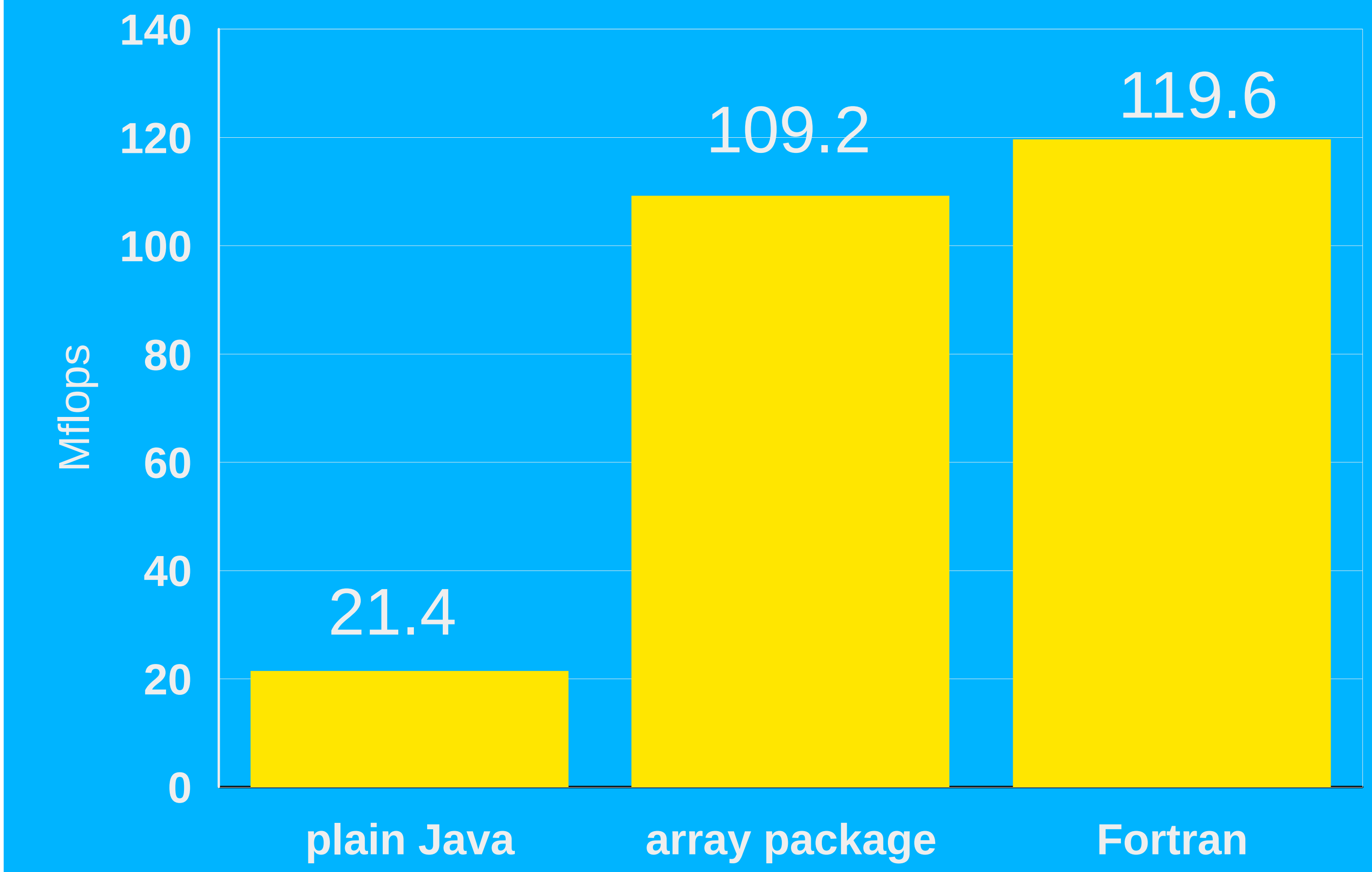
A(0,0)	A(0,1)	A(0,2)	A(0,3)
A(1,0)	A(1,1)	A(1,2)	A(1,3)
A(2,0)	A(2,1)	A(2,2)	A(2,3)
A(3,0)	A(3,1)	A(3,2)	A(3,3)

Benefits of rectangular arrays

- **Location of element A(i,j) can be computed with index arithmetic. Access to A[i][j] requires pointer chasing.**
- **Easier to disambiguate two rectangular arrays A and B than two rows A[i] and B[j].**
  - disambiguation required to enable the previously listed optimizations!
- **Out of bound index and null pointer checks can be eliminated more easily**
- **Privatization for thread safety is trivial for rectangular arrays, but requires copying vector of pointers for array of arrays.**
- **The array package:**
  - Collection of classes that implement arithmetic operations on multidimensional rectangular arrays.
  - Approach allows classical optimizations developed for Fortran to be applied.

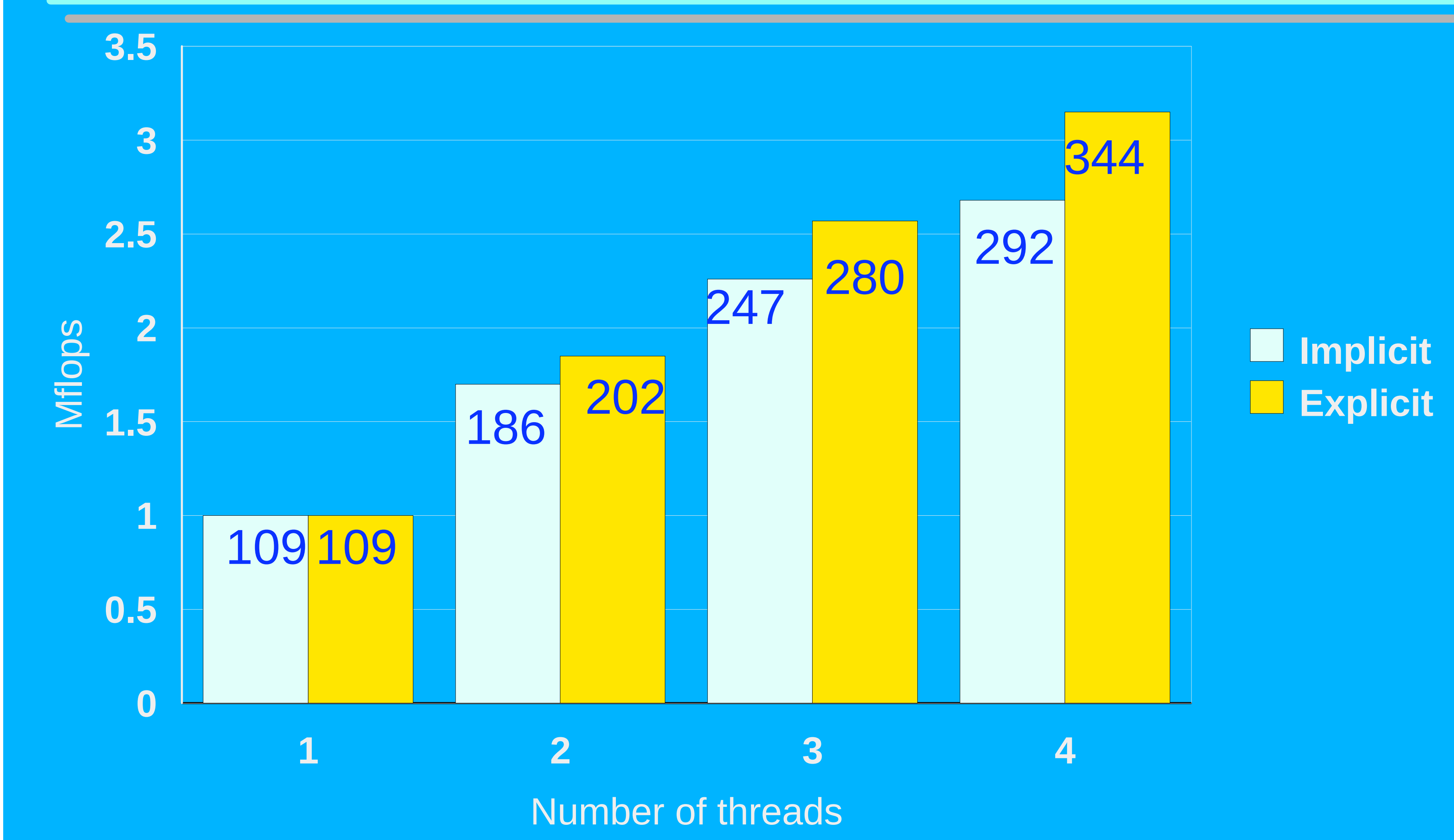
Marc Snir

Data mining code on one F50 node



- Sparse matrix operations.
- Computations performed through array arithmetic and BLAS class for multidimensional arrays.

Parallel data mining



- Explicit: code uses Java threads
- Implicit: parallelism inside array package

Marc Snir

Complex numbers & semantic inlining

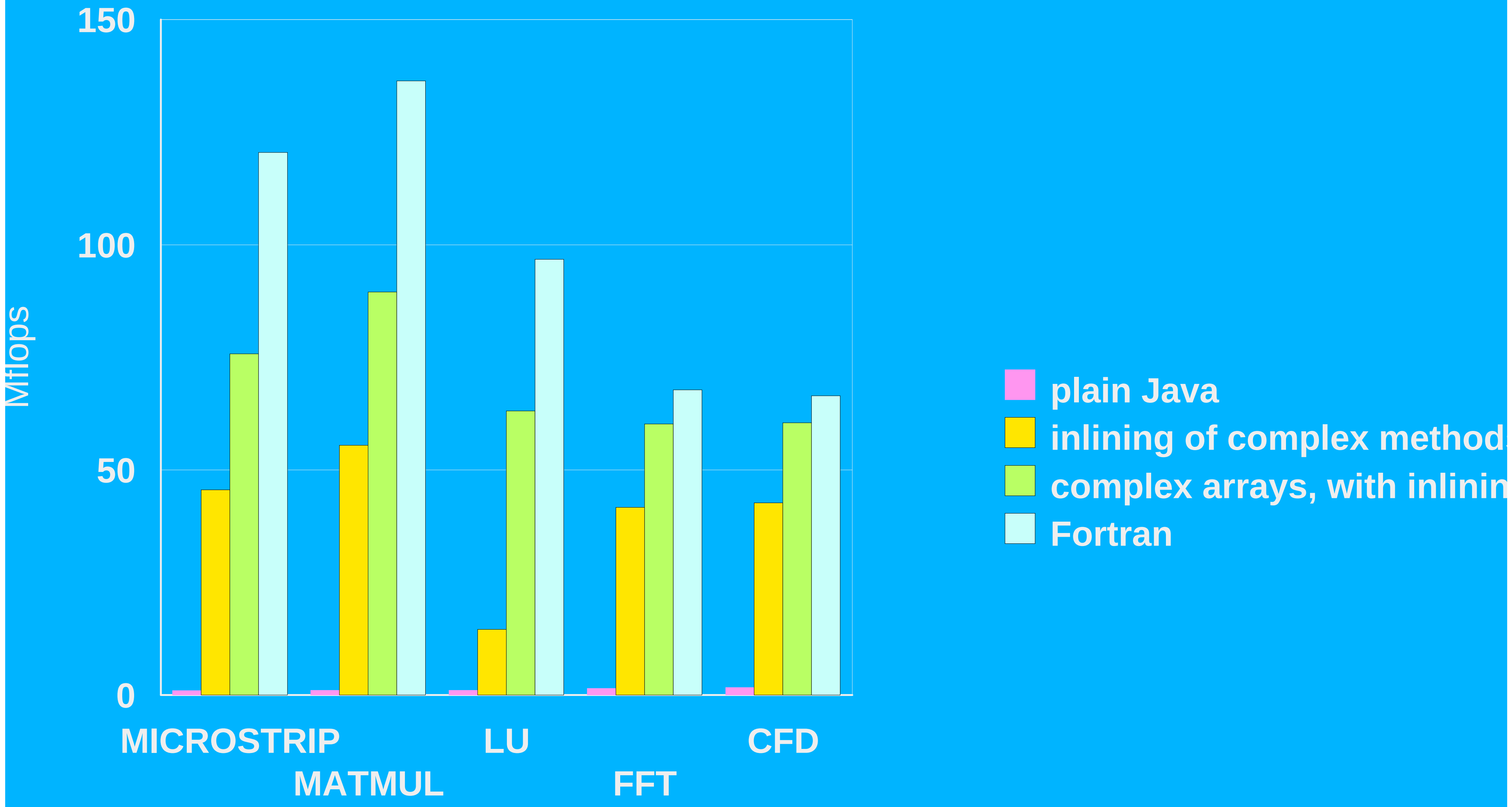
- **Example: dot product**

```
      Complex[] a,b;
Complex s;
for (int i=0; i<n; i++)
  s.assign(s.plus(a[i].times(b[i])));
```

  - generates 2n Complex objects that only hold intermediate results!

- **Semantic inlining:**
  - Compiler recognizes ops on standard classes as primitives.
  - Compiler generates code that implements semantics of classes, disregards bytecodes for methods.
  - Mechanism to extend Java and the JVM "under the covers". No need to add bytecodes.
  - Can implement "value objects" without any language or JVM changes.

Results on RS/6000 590


Mflops
MICROSTRIP
MATMUL
LU
FFT
CFD
plain Java
inlining of complex method
complex arrays, with inlinin
Fortran