Optimizing and Parallelizing Java

Marc Snir Manish Gupta Sam Midkiff José Moreira
IBM T. J. Watson Research Center
P.O. Box 218
Yorktown Heights NY 10598

What about performance?

MATMUL benchmark description:

- **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
- $m = n = p = 64$ (fits in cache of POWER2).

MATMUL results on the RS/6000 590

Why is Fortran so much faster?

- At 205 Mflops, Fortran is 50% faster that C, and ~ 100 times faster than Java!

- The IBM XLF 4.1 compiler uses several high-order transformations:
  - blocking, for better memory behavior,
  - multiple loop unrolling,
  - loop interchange,
  - scalar replacement,
  - loop fusion.

- This technology is moving to other languages as well, with the IBM Toronto Portable Optimizer (TPO).

- The same transformations can be applied to safe regions of Java! Normally, exceptions prevent reordering.

Creating a safe region

- First step: localize array for thread safety. Much simpler with Matrix objects.

matmul(A',B',C',m,n,p) {
  double[][] A = **new double**[A'.length][]
  $A_{\text{rows}} = A.\text{length}$
  $A_{\text{cols}} = \infty$
  for $(i = 0; i < A.\text{length}; i{+}{+})$ {
    $A[i] = A'[i]$
    $A_{\text{cols}} = \min(A_{\text{cols}}, A[i].\text{length})$
  }
  ...
}

- Do the same for B and C.

IBM

Creating a safe region (continued)

- Second step: avoid index exceptions.

if $((m < C_{\text{rows}}) \wedge (p < C_{\text{cols}}) \wedge (m < A_{\text{rows}}) \wedge (n < A_{\text{cols}}) \wedge \ldots)$ $\{$
  for $(i = 0; i < m; i{+}{+})$
    for $(j = 0; j < p; j{+}{+})$
      for $(k = 0; k < n; k{+}{+})$
        $C[i][j] \mathrel{+}= A[i][k] * B[k][j]$
$\}$ **else** $\{$
  for $(i = 0; i < m; i{+}{+})$
    for $(j = 0; j < p; j{+}{+})$
      for $(k = 0; k < n; k{+}{+})$
        $C[i][j] \mathrel{+}= A[i][k] * B[k][j]$
$\}$

- The red version is safe! Bounds checking is not necessary and reordering is possible!

Optimizing the safe region

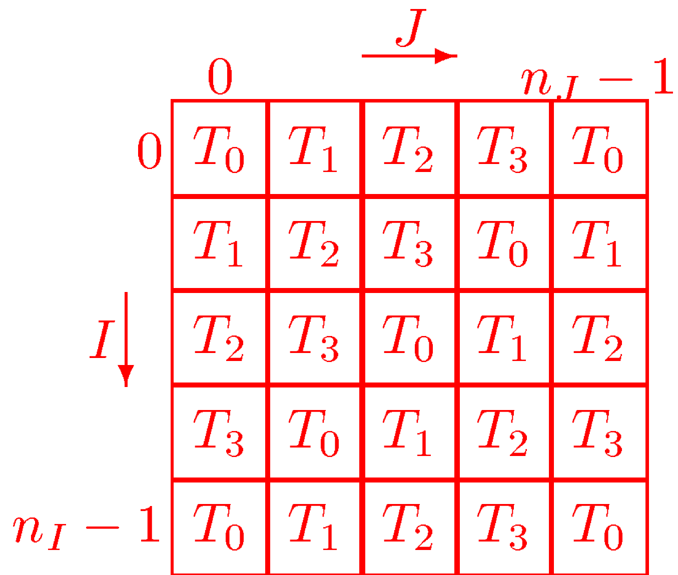
- blocking: for better cache reuse.
- unrolling: of i and j loops for better register reuse and to exploit functional-unit parallelism. Note that k loop cannot be parallelized in Java!
- scalar replacement: requires pointer disambiguation. Can be done at run-time, much easier with Matrix objects.
- fma: uses extended-precision fused-multiply-add instruction. Not legal yet.

In the next plot, library has all the optimizations of the fma version except for bounds checking. This is the best hand implementation with all checks explicit.

500×500 MATMUL

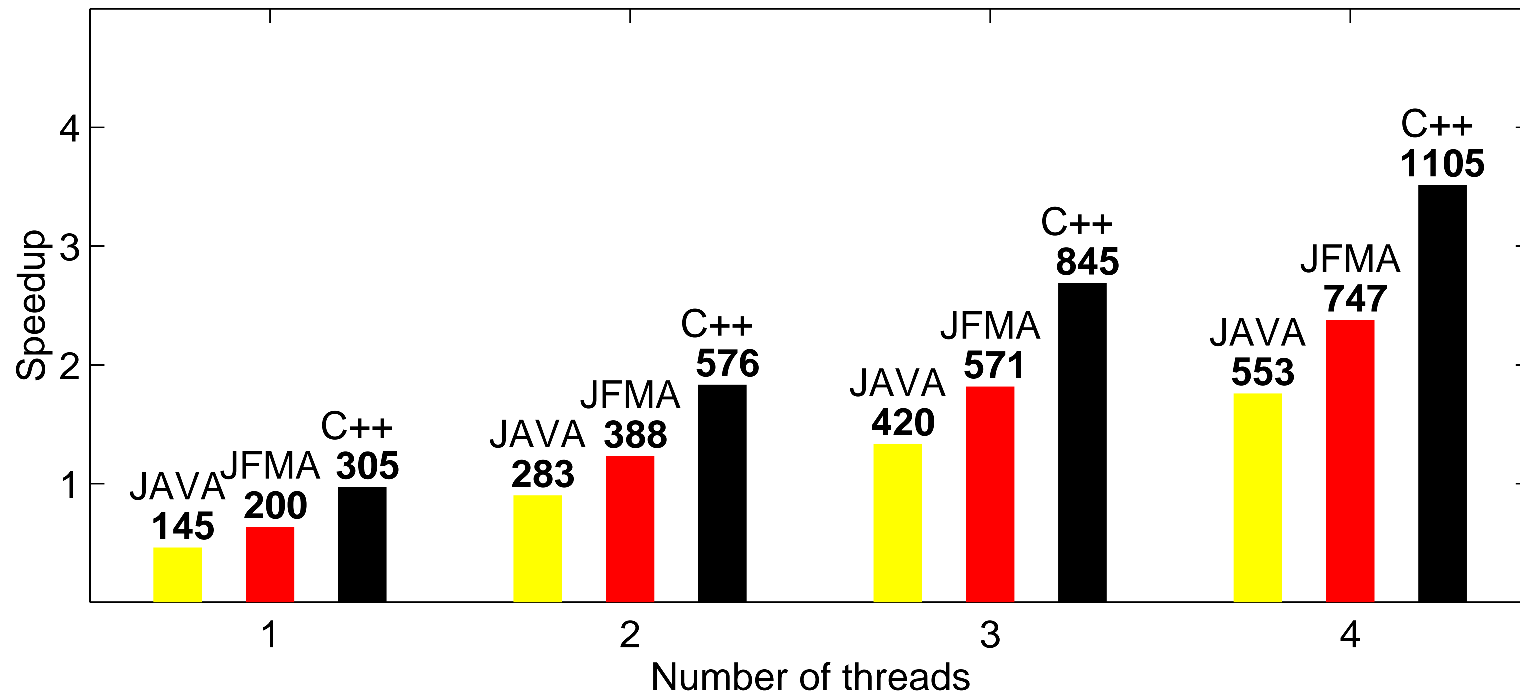
Parallelizing MATMUL

- The safe region can be multithreaded! Exploit parallelism on I and J block loops.
- b_I,b_J, and b_K are the block sizes along I, J, and K. t is the thread id, n_T is the number of threads.




```
for (b = t; b < n_I n_J; b+=n_T) {
   I = (b/n_I) × b_I
   J = (b mod n_I) × b_J
   for (K = 0; K < n; K+=b_K) {
      for (i = I; i < min(m, I + b_I); i++)
         for (j = J; j < min(p, J + b_J); j++)
            for (k = K; k < min(n, K + b_K); k++)
               C[i][j]+=A[i][k] * B[k][j]
   }
}
```

Parallelizing MATMUL – results



Parallel ESSL: 1183 Mflops.