Data-centric Multi-level Blocking

Induprakas Kodukula

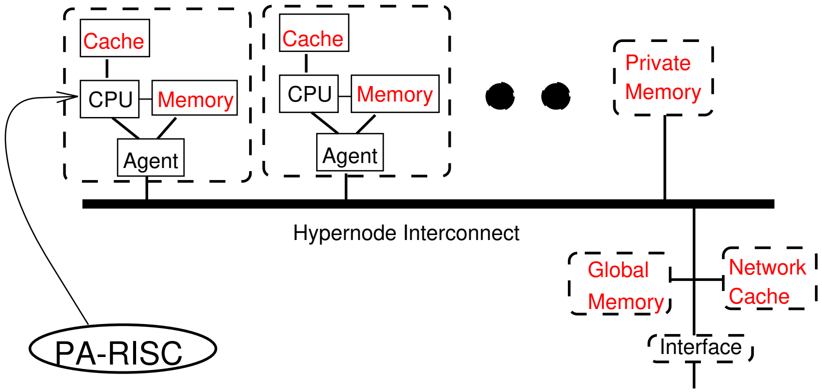
Nawaaz Ahmed

Keshav Pingali

This research was supported by an NSF Presidential Young Investigator award CCR-8958543, NSF grant CCR-9008526, ONR grant N00014-93-1-0103, and a grant from Hewlett-Packard Corporation.

Hardware: Clusters of SMPs

Convex Exemplar Architecture:



Cache	10ns
Local Memory	500ns
Remote Cluster	2us

Hypernode

Latency increases by a factor of 5-10 from level to level

Locality of reference is crucial for performance

Exploiting locality in memory hierarchies

- Library code: LAPACK
- Compiler state of the art: *Loop tiling*
- “Data-centric” Approach: Data shackles
- Performance
- Multilevel blocking
- Extensions and Ongoing work

What is locality?

```
do I = 1, N
  do J = 1, N
    .... C[J] ....
    .... C[J-2].....
```

	Temporal	Spatial
Self	C[J]	C[J]
Cross	C[J], C[J-2]	C[J], C[J-2]

Hand-written libraries

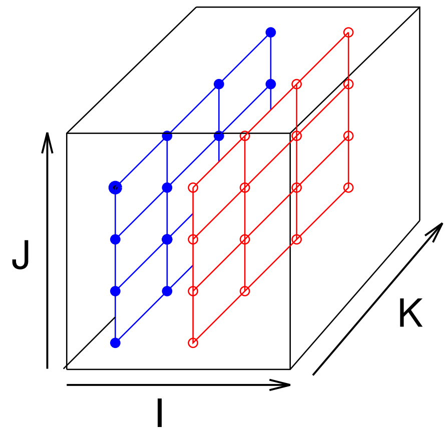
LAPACK

- Restructure at high level to expose matrix multiply operations
- Exploit locality in matrix multiply ($O(n^3)$ work on $O(n^2)$ data)
- Routines
  - Matrix-multiply
  - Cholesky
  - Gaussian Elimination
  - Gaussian Elimination w/ partial pivoting
  - QR Factorization

Designed for a two level memory hierarchy

Restructuring compiler technology: Tiling

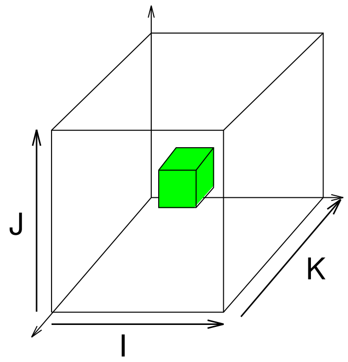
```
do I = 1, N
  do J = 1, N
    do K = 1, N
      C[I,J] = C[I,J] + A[I,K] * B[K,J]
```



Perfectly nested loops

- All statements in innermost loop
- Loop execution mapped to points in integer lattice
- Key transformation: Tiling
- Enabling transformations: Interchange, skewing, etc.

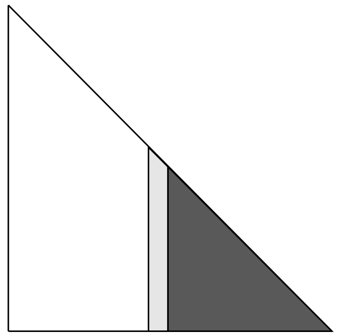
Tiling - exploit locality along multiple directions




```
do I = 1, N/25
 do J = 1, N/25
  do K = 1, N/25
   do It = I*25, I*25+24
    do Jt = J*25, J*25+24
     do Kt = K*25, K*25+24
      S1: C[It,Jt]=C[It,Jt]+A[It,Kt]*B[Kt,Jt]
```

Limitations of Tiling - #1

```
do K = 1, N
   A[K,K] = sqrt (A[K,K])
   do I = K+1, N
      A[I,K] = A[I,K] / A[K,K]
   do J = K+1, N
      do L  = K+1, J
         A[J,L] = A[J,L] - A[J,K] * A[L,K]
```



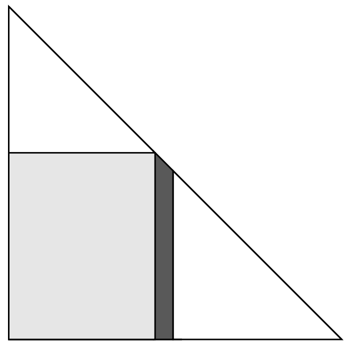
Right-looking Cholesky

- Make loop perfectly nested: many choices
- Correct choice:

1. Fuse I and J loops
2. Sink all statements
3. Tile

Limitations of Tiling - #2

```
do K = 1, N
   do L = K+1, N
      do J = 1, K-1
         A[L,K] = A[L,K] - A[L,J] * A[K,J]
   A[K,K] = sqrt (A[K,K])
   do I = K+1, N
      A[I,K] = A[I,K] / A[K,K]
```



Left-looking Cholesky

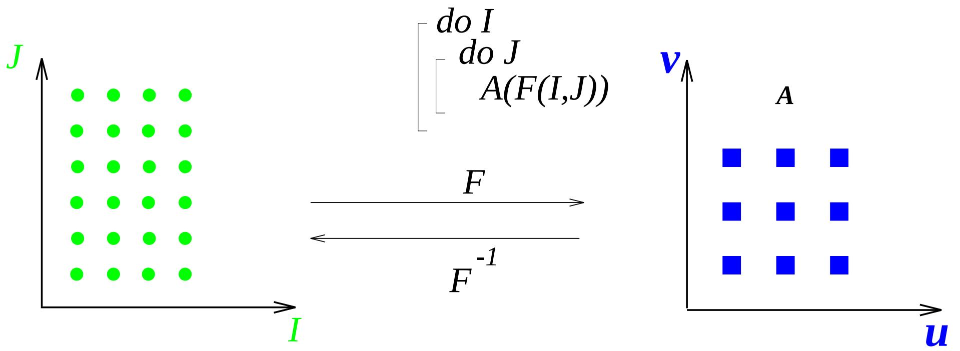
- Correct choice:
  1. S2 must be sunk into I loop
  2. Fuse resulting loops
  3. Tile
- Different from the sequence for previous version!

Summary of Tiling

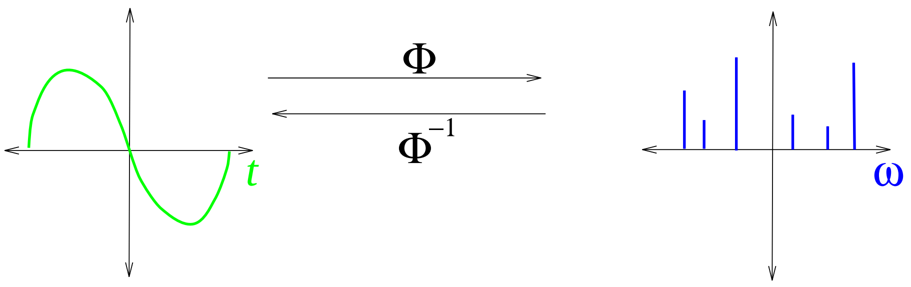
- Works very well for perfectly nested loops
- Majority of codes imperfectly nested
- Many choices to make imperfectly nested codes perfectly nested
- Tiling requires code-specific enabling loop transformations
- No systematic scheme to deduce these loop transformations

Data-centric approach

- Iteration space indirect measure of locality
- Drive transformations by concentrating on data



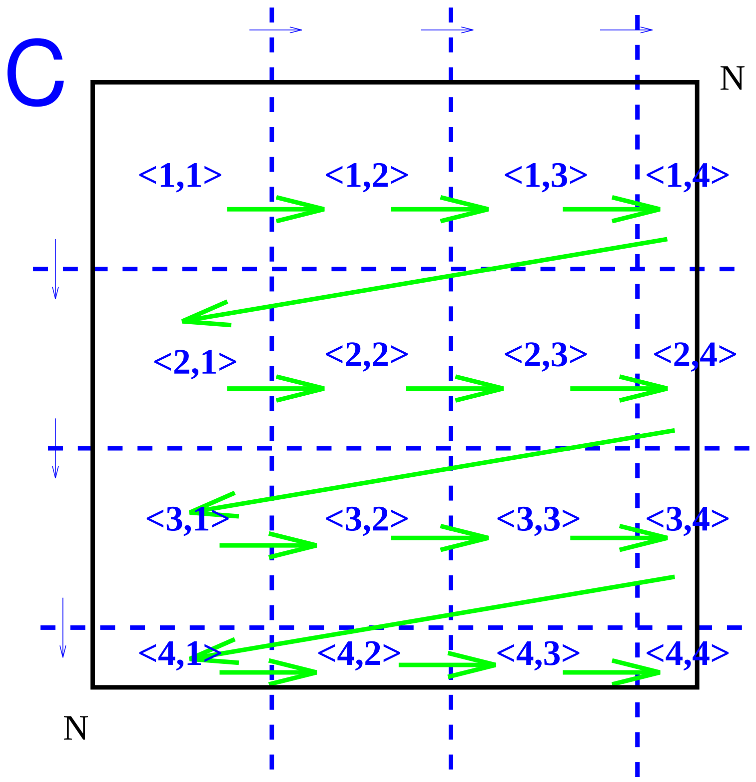
- Signal processing analogy: Fourier Transforms



Data shackle

- Choose a data object, chop it into “blocks”
- Specify an order to touch the blocks
- Shackle statement instances to blocks using Data-centric reference

```
do I = 1, N
 do J = 1, N
  do K = 1, N
   C[I,J]= C[I,J]+A[I,K]*B[K,J]
```



Blocked code - “Localization Constraints”

```
do C1 = 1, 4
  do C2 = 1, 4
    do I = 1, N
      do J = 1, N
        do K = 1, N
          if ((25*C1-24 <= I <= 25*C1) &&
              (25*C2-24 <= J <= 25*C2))
            C[I,J] = C[I,J] + A[I,K] * B[K,J]
```

Localization constraints are affine functions of surrounding loop indices

Blocked code with folded Localization Constraints

- Simplify them using a standard polyhedral analysis tool (PIP/Omega)

```
do C1 = 1, 4
  do C2 = 1, 4
    do I = 25*C1-24, min(25*C1,N)
      do J = 25*C2-24, min(25*C1, N)
        do K = 1, N
          C[I,J] = C[I,J] + A[I,K] * B[K,J]
```

Folding localization constraints into loop bounds is automatic

Shackling and locality, a.k.a Why is shackling a good idea?

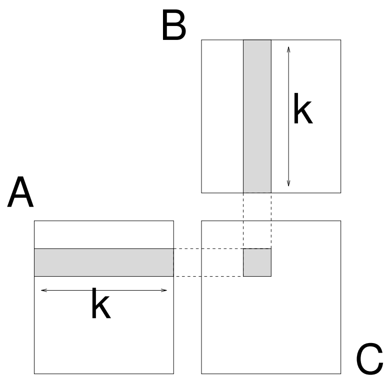
If block of data remains in cache, for *data-centric* references

- Perfect self temporal locality
- Perfect group temporal locality

- Perfect self spatial locality
- Perfect group spatial locality

What about *non* data-centric references?

Composition of shackles #1




```
do C1 = 1, 4
  do C2 = 1, 4
    do I = 25*C1-24, min(25*C1,N)
      do J = 25*C2-24, min(25*C1, N)
        do K = 1, N
          C[I,J] = C[I,J] + A[I,K] * B[K,J]
```

- K is unconstrained $\Longrightarrow$ *Poor locality for references to A and B*
- Solution: Use an additional shackle to constrain K

Composition of shackles #2

- Outer shackle over C with C[I,J]
- Inner shackle over A with A[I,K]

```
Over all blocks of C do
  Over all blocks of A do
    do I = 1, N
      do J = 1, N
        do K = 1, N
          if ((C[I,J] in block of C) &&
              (A[I,K] in block of A))
            C[I,J] = C[I,J] + A[I,K] * B[K,J]
```

- Notation for composition: Red$\otimes$Cyan
- I, J, K are all constrained, further shackling not useful

Optimized composite blocked code

```
do C1 = 1, 4
  do C2 = 1, 4
    do A2 = 1, 4
      do I = 25*C1-24, min(25*C1,N)
        do J = 25*C2-24, min(25*C2,N)
          do K = 25*A2-24, min(25*A2,N)
            C[I,J] = C[I,J] + A[I,K] * B[K,J]
```

Fully blocked Cholesky factorization

```
do K = 1,N
  A[K,K] = sqrt( A[K,K])
  do I = K+1, N
    A[I,K] =  A[I,K] / A[K,K]
  do J = K+1, N
    do L = K+1, J
      A[J,L] = A[J,L] -  A[J,K] * A[L,K]
```

- Red $\otimes$Cyan- fully blocked Cholesky in LAPACK
- Cyan $\otimes$Red- another fully blocked Cholesky
- Can obtain both versions starting from same input code, tiling cannot
- Can obtain both versions from all 6 versions of Cholesky as well, tiling cannot

Legality

- Not all shackles are legal

- S = dynamic statement instance source of a dependence
- D = dynamic statement instance destination of a dependence
- $B(s)$ = Composite block-coordinate that executes dynamic instance s

Shackling is legal iff

- $B(S) \leq B(D)$

- Formulation as integer system of constraints
- Automatic verification using existing tools (Omega/PIP)

Composite Shackling vs. Tiling

	Tiling	Composite Shackling
Matrix Multiply	$\surd$	$\surd$
Cholesky Factorization	$\times$	$\surd$
Gaussian Elimination	$\times$	$\surd$
GE w/ partial pivoting	$\times$	Updates blocked, not row permutations
QR Factorization	$\times$	Only updates blocked

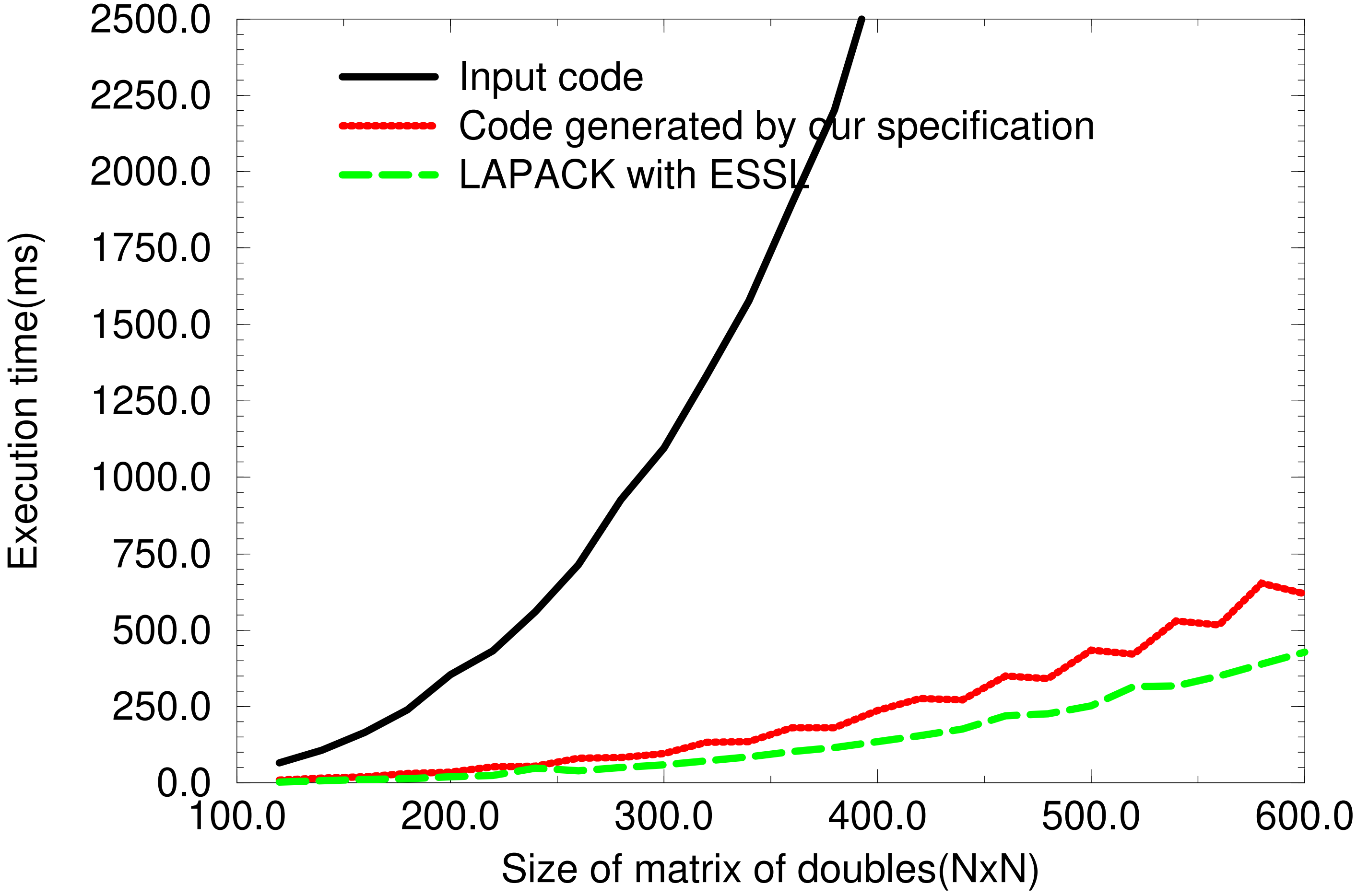
Experimental setup

- Designed an annotation scheme for expressing shackles
- Fortran source-to-source translator
- Pipe Fortran output through vendor's compiler

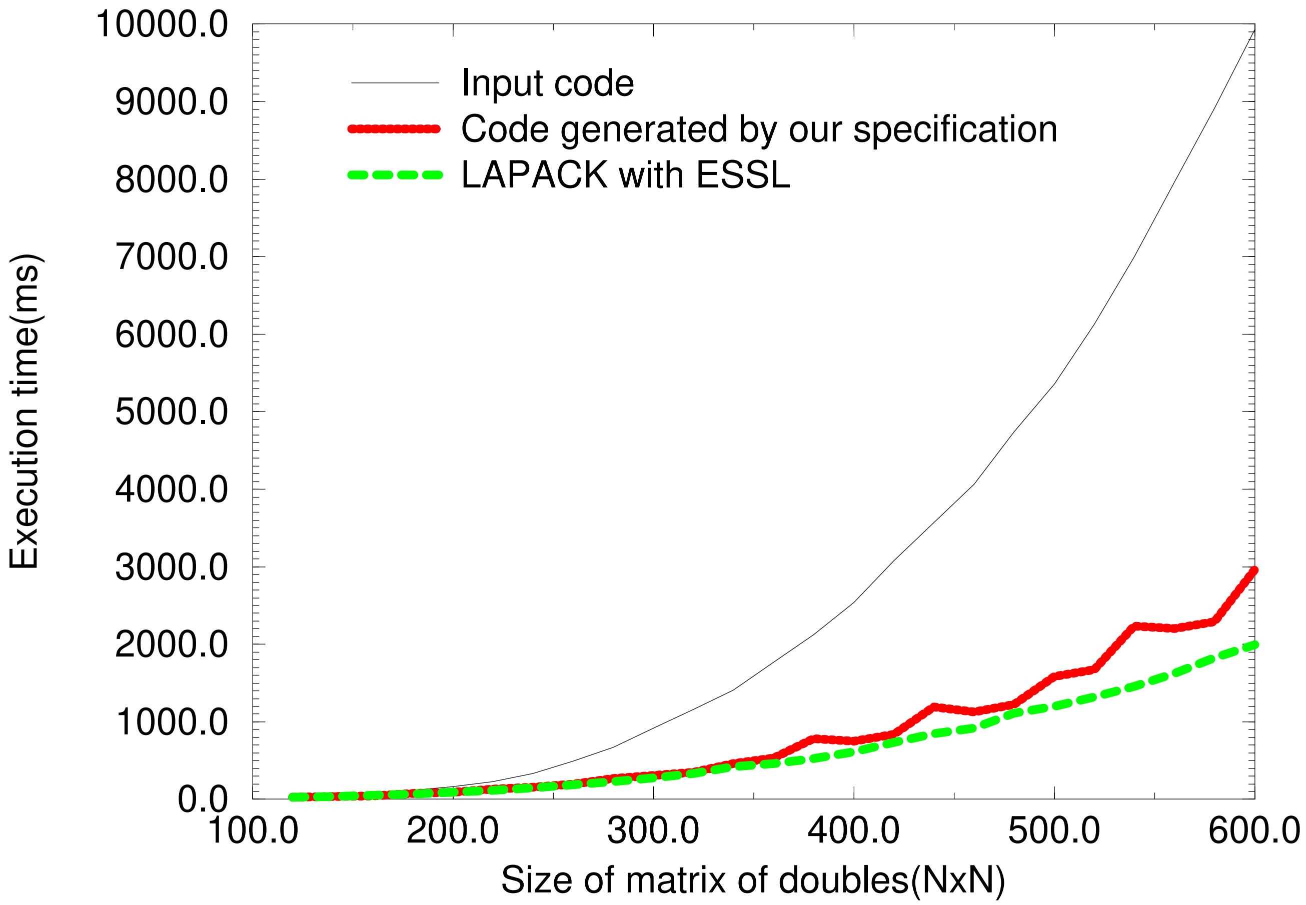
Hardware Platform:

- Single thin node of the SP-2 ($\approx$ RS/6000, model 390)
- Peak performance - 266 MFlops, Clock speed 66.7 MHz
- 128 MBytes memory, 64 KBytes cache (4-way assoc)
- Xlf version 4.1.0.0, -O3
- Compare with routines from ESSL (hand-tuned library routines)

Cholesky factorization on the RS/6000 - timings

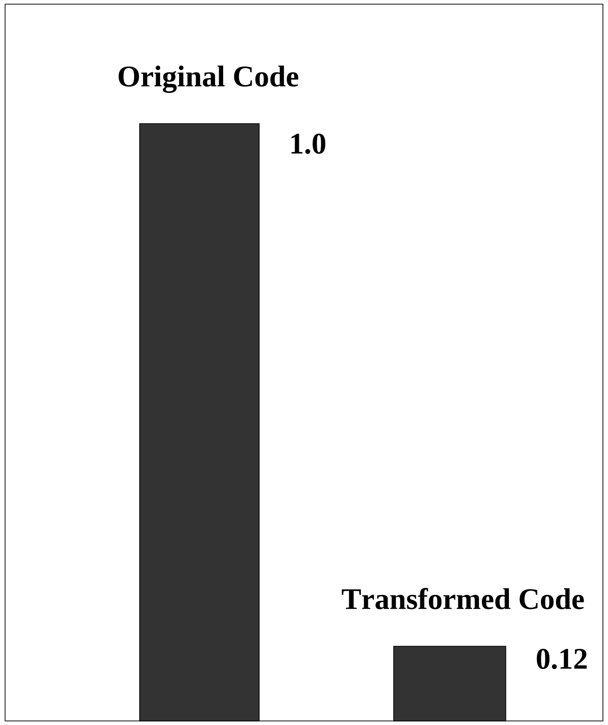


QR factorization on the RS/6000 - timings



Data-centric transformations different from blocking

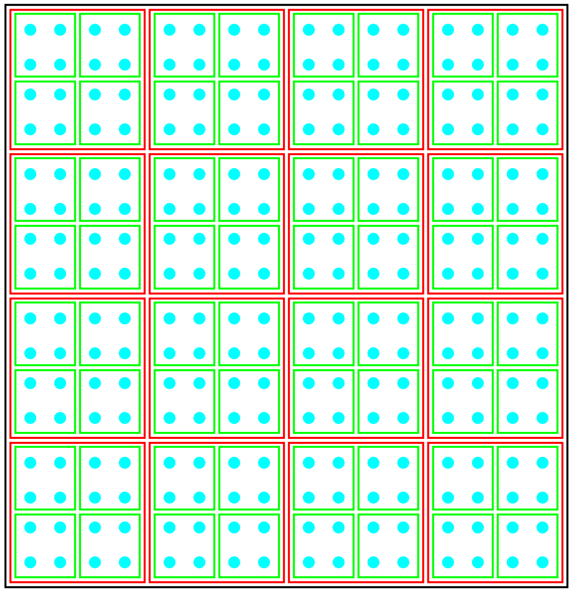
```
do I = 1, N
   do J = 1, N
      B(I,J) = A(I,J-1) + A(I,J+1)
do J = 1, N
   do I = 1, N
      C(J,I) = A(J,I-1) + A(J,I+1)
```



Normalized execution time

- Output from a Fortran 90 scalarizer
- Used by McKinley et al (TOPLAS, 7/96)

Multiple levels of memory hierarchy



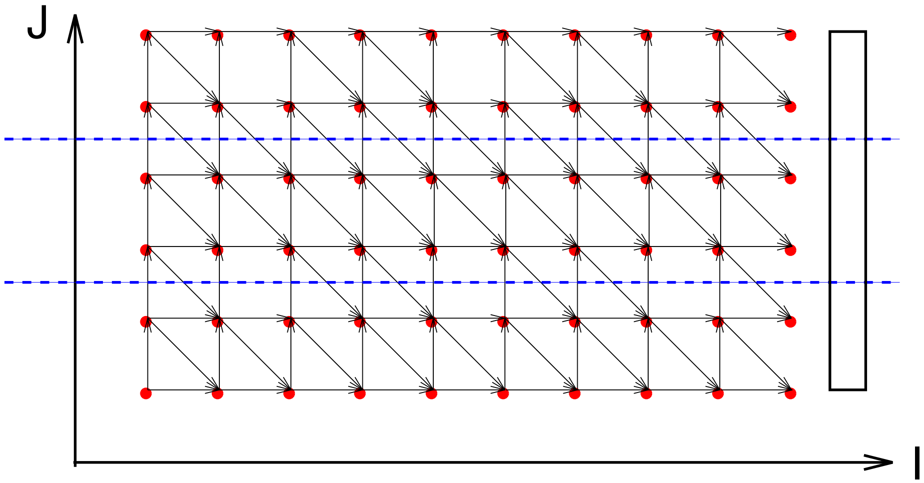
- Repeated composition
- Small block sizes for smaller levels of memory hierarchy
- Have used for register tiling on the SGI
- Currently looking at
  - Out-of-core computations
  - Remote processor memory on SMPs

Ongoing work #1

- All work assigned to a block can't be done at one shot

```
do I = 1, M
  do J = 1, N
    A[J] = (A[J-1]+A[J]+A[J+1])/3
```

Multiple passes over each block required in general



Ongoing work and Open Problems

- Automating shackling
  - Determine specifications of a shackle automatically
  - Detailed memory model
- Utilizing application-specific information?
  - Matrix product associativity in QR?
  - Start with MATLAB-like specification?
- Applications other than blocking?
  - Parallelization
  - Prefetching in DSM systems