CS 267
More on Communication-optimal Matmul (and beyond)

James Demmel
www.cs.berkeley.edu/~demmel

Outline

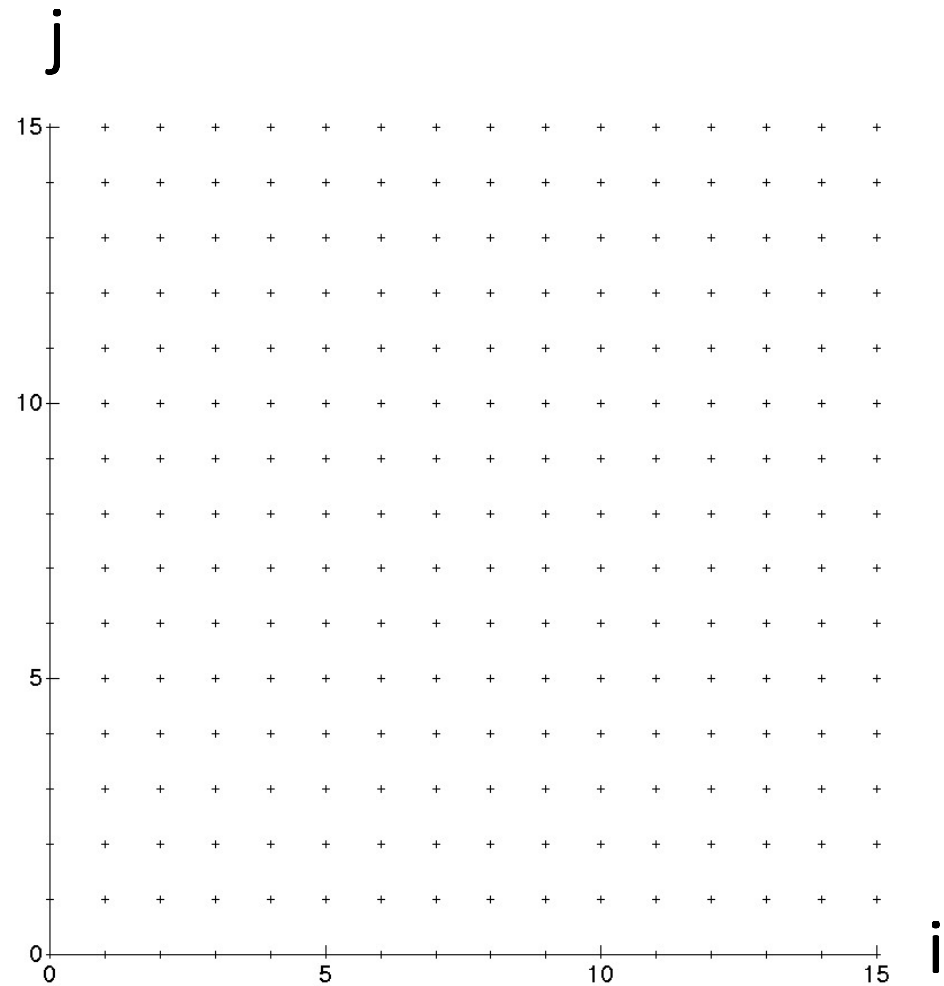
- Communication = moving data
  - Between main memory and cache
  - Between processors over a network
  - Most expensive operation (in time or energy)
- Goal: Provably minimize communication for algorithms that look like nested loops accessing arrays
  - Includes matmul, linear algebra (dense and sparse), n-body, convolutional neural nets (CNNs), …
- Simple case: n-body (sequential, with main memory and cache)
  - Communication lower bound and optimal algorithm
- Extension to Matmul
- Extension to algorithms that look like nested loops accessing arrays, like CNNs (and open questions)

Data access for n-body

- A() = array of structures
  - A(i) contains position, charge on particle i
- Usual n-body
  - for i = 1:n, for j = 1:n except i, F(i) = F(i) + force(A(i),A(j))
- Simplify to make counting easier
  - Let B() = array of disjoint set of particles
  - for i = 1:n, for j = 1:n, e = e + potential(A(i),B(j))
- Simplify more
  - for i = 1:n, for j = 1:n, access A(i) and B(j)

Data access for n-body

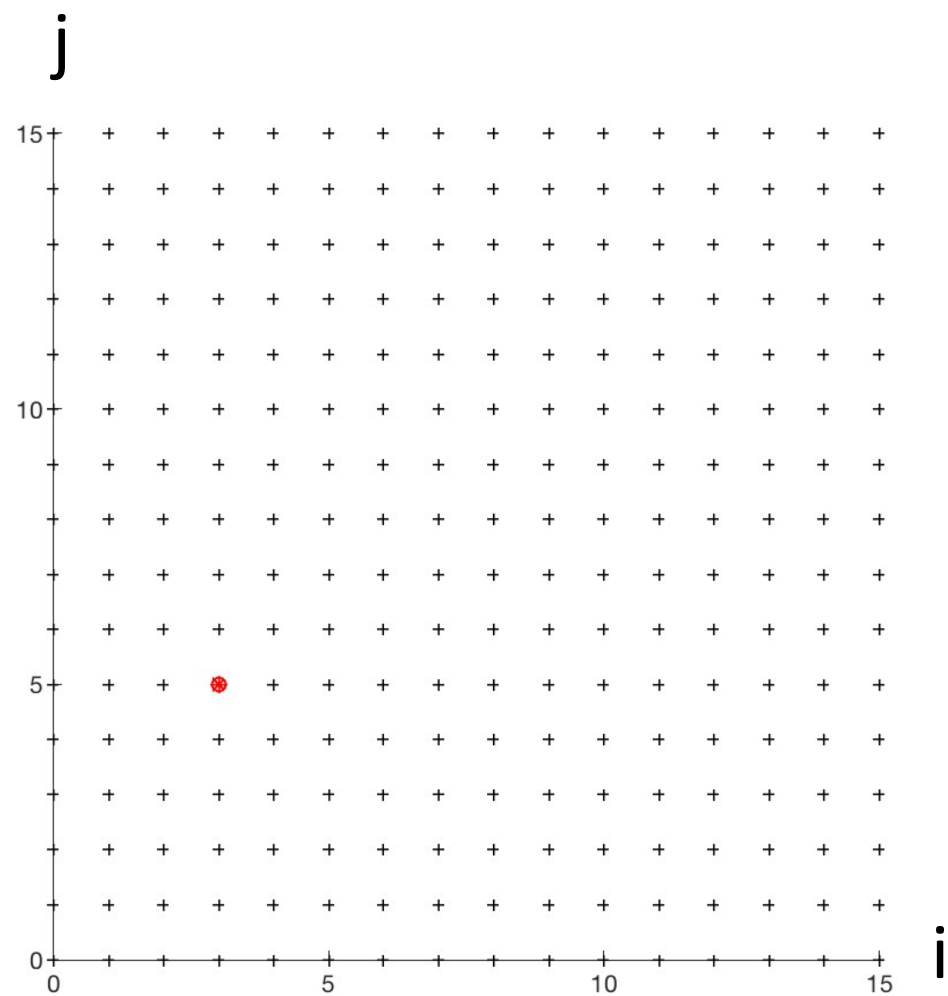
```
for i = 0:n
  for j = 0:n
    access A(i), B(j)
```



Data access for n-body

```
for i = 0:n
   for j = 0:n
      access A(i), B(j)
```

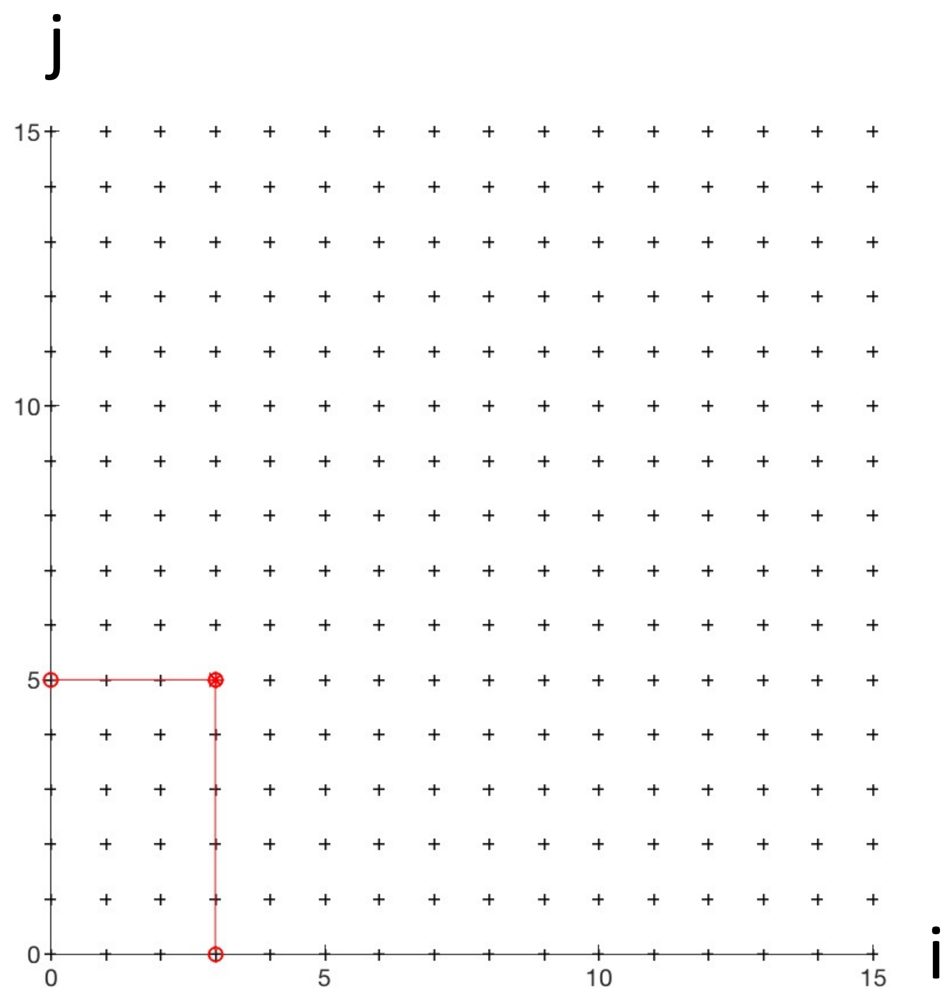
Ex: execute loop for
   i = 3, j = 5



Data access for n-body

for i = 0:n
  for j = 0:n
    access A(i), B(j)

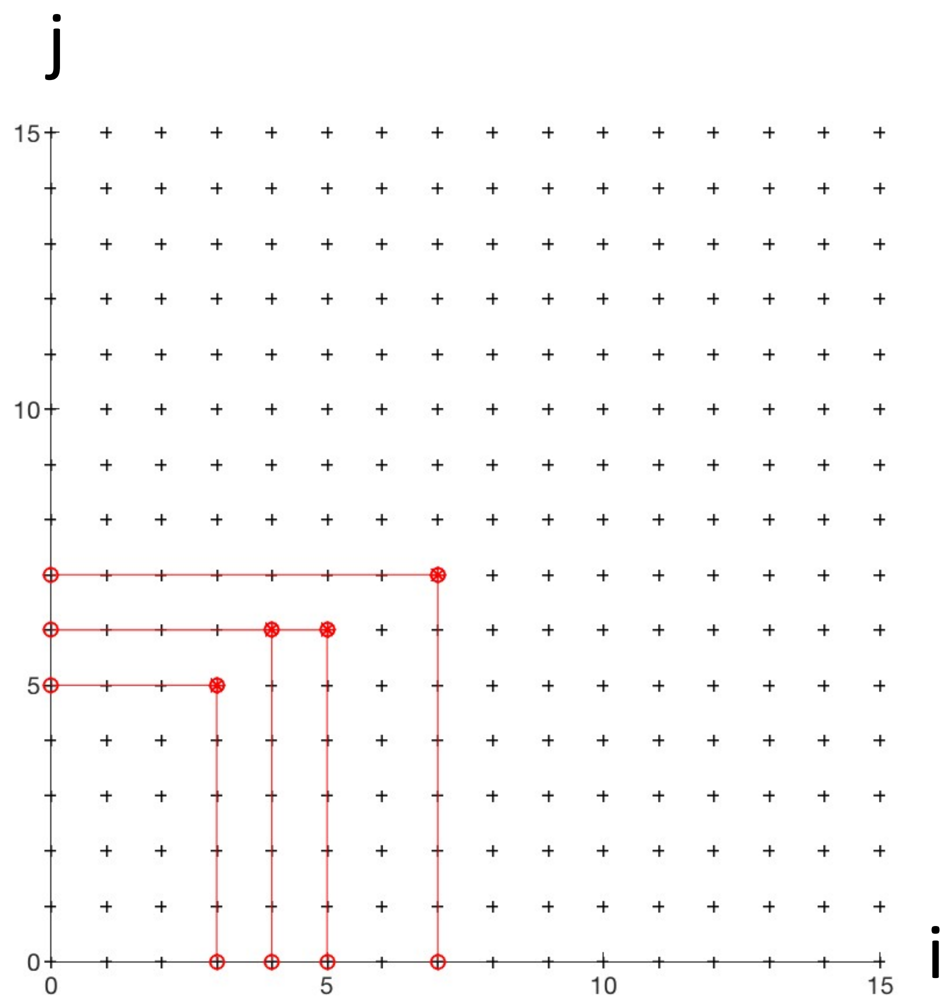
Ex: execute loop for
  i = 3, j = 5
access A(3), B(5)



Data access for n-body

for i = 0:n
  for j = 0:n
    access A(i), B(j)

Ex: execute loop for
  multiple pairs (i,j),
  access multiple
  A(i), B(j)

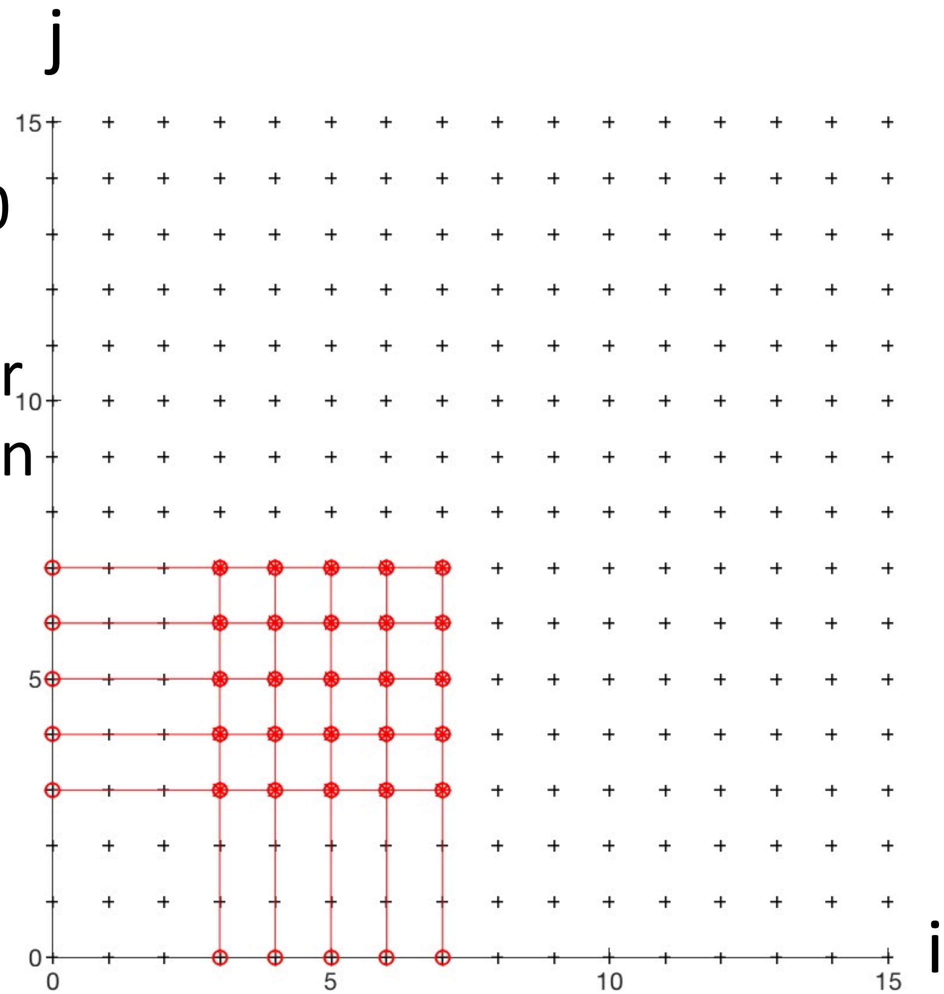


Data access for n-body

If we can only access 10 entries of A(i) and B(j), what is the max number of loop iterations we can do?

25 = 5 x 5

If we can access
M = cache size
entries, then we can do
$(M/2)^2 = M^2/4$
loop iterations



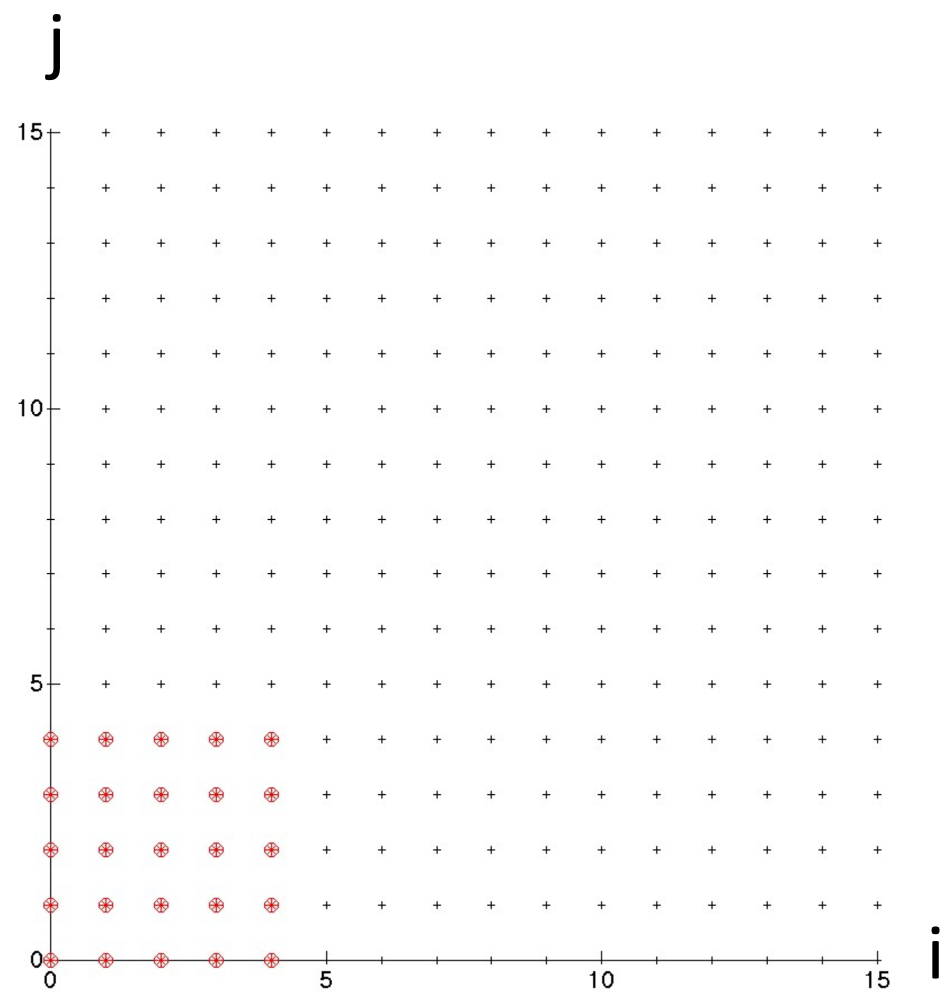
Communication lower bound for n-body (intuition)

- for i=1:n, for j=1:n, access A(i), B(j)
- With a cache of size M full of data, can only perform $M^2/4$ loop iterations
- To perform all n^2 loop iterations, need to (re)fill cache $n^2/(M^2/4) = 4(n/M)^2$ times
- Filling cache costs M reads from slow memory
- Need to do at least $4(n/M)^2 * M = 4n^2 / M$ reads
  - Can improve constant slightly
  - Write as $\Omega(n^2/M) = \Omega(\#\text{loop iterations} / M)$

Optimal tiling for usual n-body

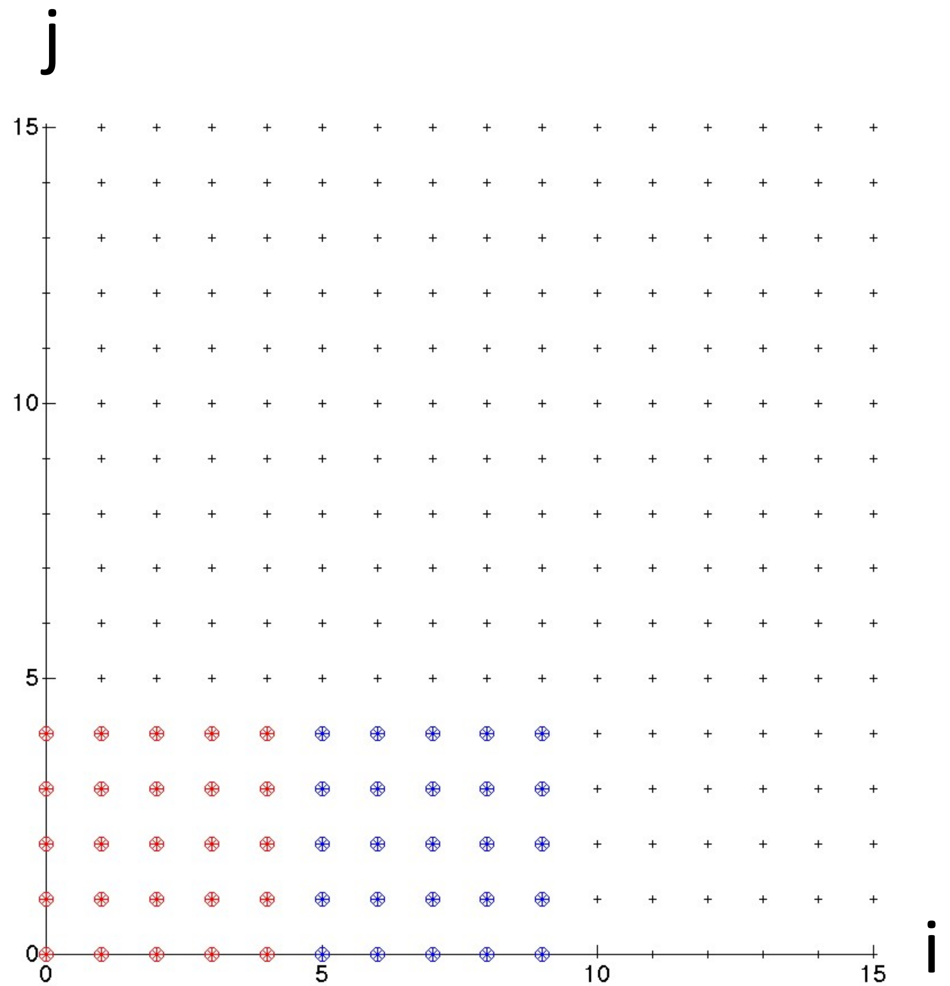
for i = 0:n
  for j = 0:n
    access A(i), B(j)

Tiling (M=10)
  Read 5 entries of A:
    A([0,1,2,3,4])
  Read 5 entries of B:
    B([0,1,2,3,4])
  Perform 5^2 = 25
    loop iterations



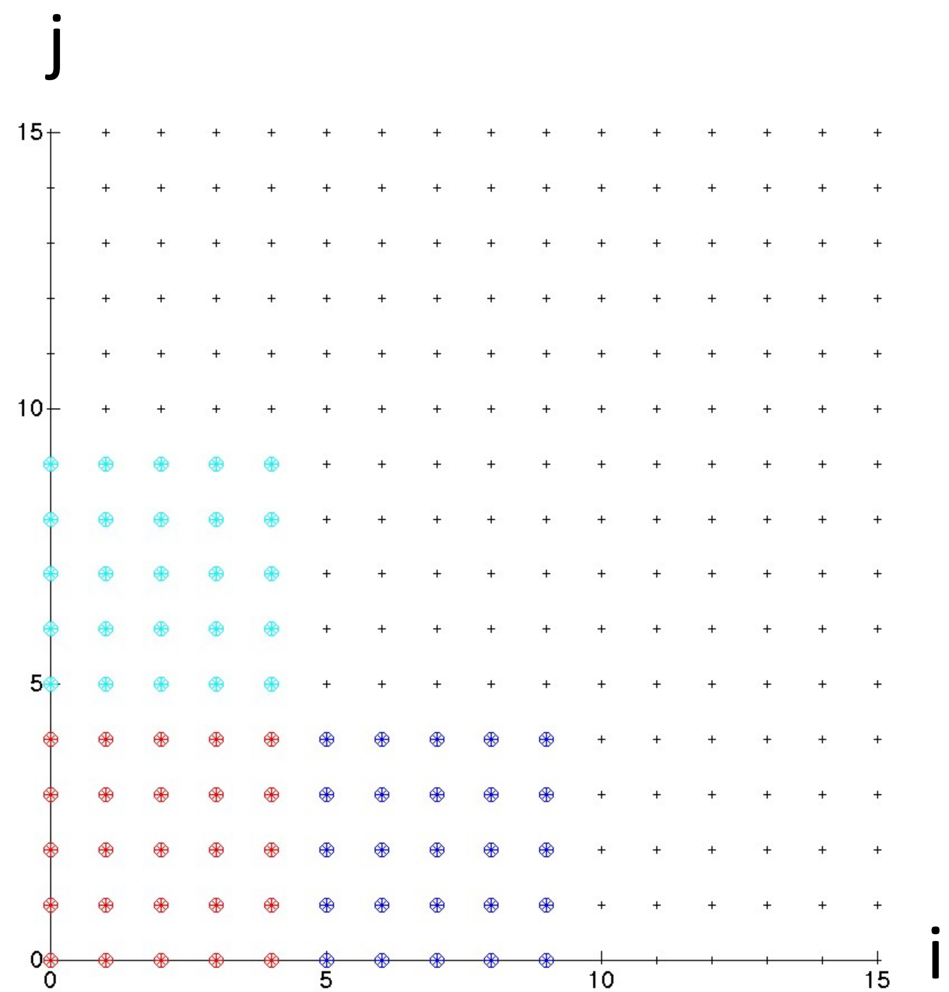
Optimal tiling for usual n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(j)
```



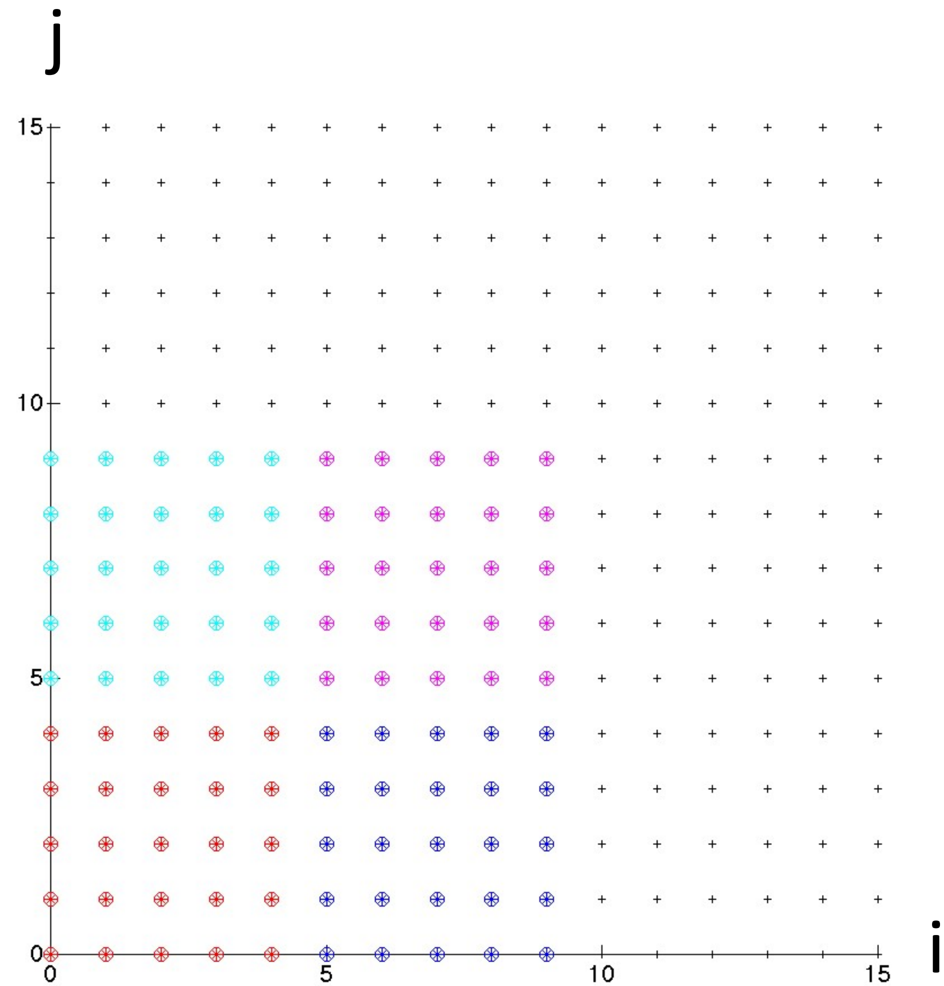
Optimal tiling for usual n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(j)
```



Optimal tiling for usual n-body

```
for i = 0:n
   for j = 0:n
      access A(i), B(j)
```



Generalizing to other algorithms

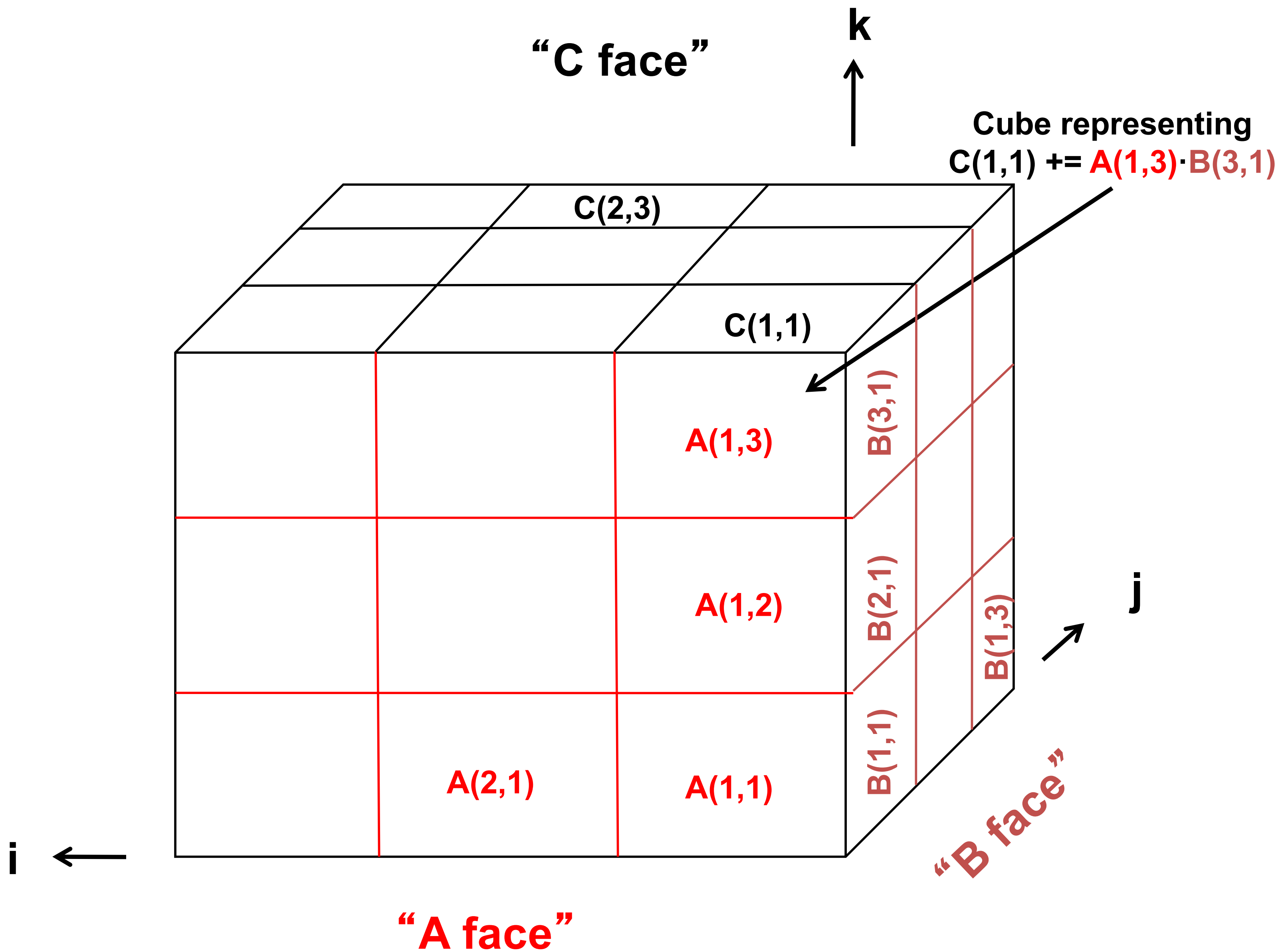
- Many algorithms look like nested loops accessing arrays
  - Linear Algebra (dense and sparse)
  - Grids (structured and unstructured)
  - Convolutional Neural Nets (CNNs) …
- Matmul: C = A*B
  - for i=1:n, for j=1:n, for k=1:n

    C(i,j) = C(i,j) + A(i,k) * B(k,j)

Proof of Communication Lower Bound on C = A·B (1/4)

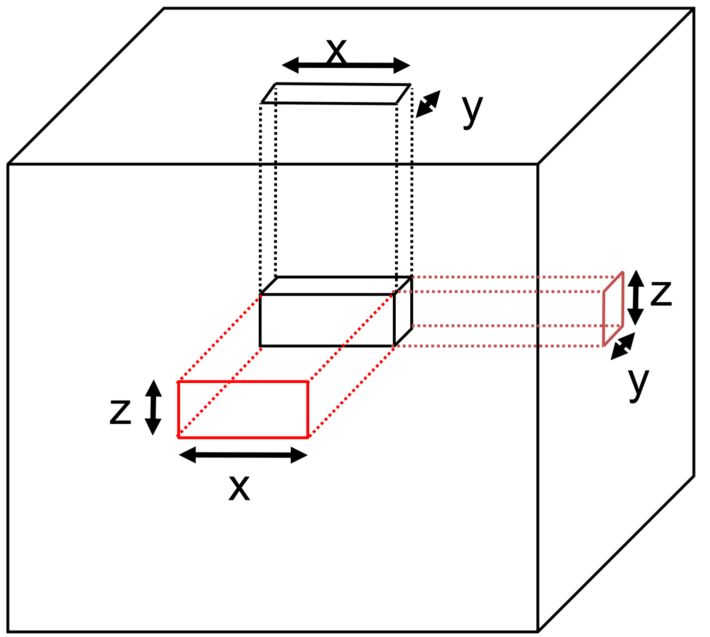
- Analogous to n-body:
  - Only M entries of A, B and C are available in cache
  - Find an upper bound F on the number of different iterations C(i,j) = C(i,j) + A(i,k)*B(k,j) we can perform
  - Need to refill cache n^3/F times to complete algorithm
  - Need to read/write at least M n^3/ F words to/from cache
- Like n-body, represent iterations and data geometrically

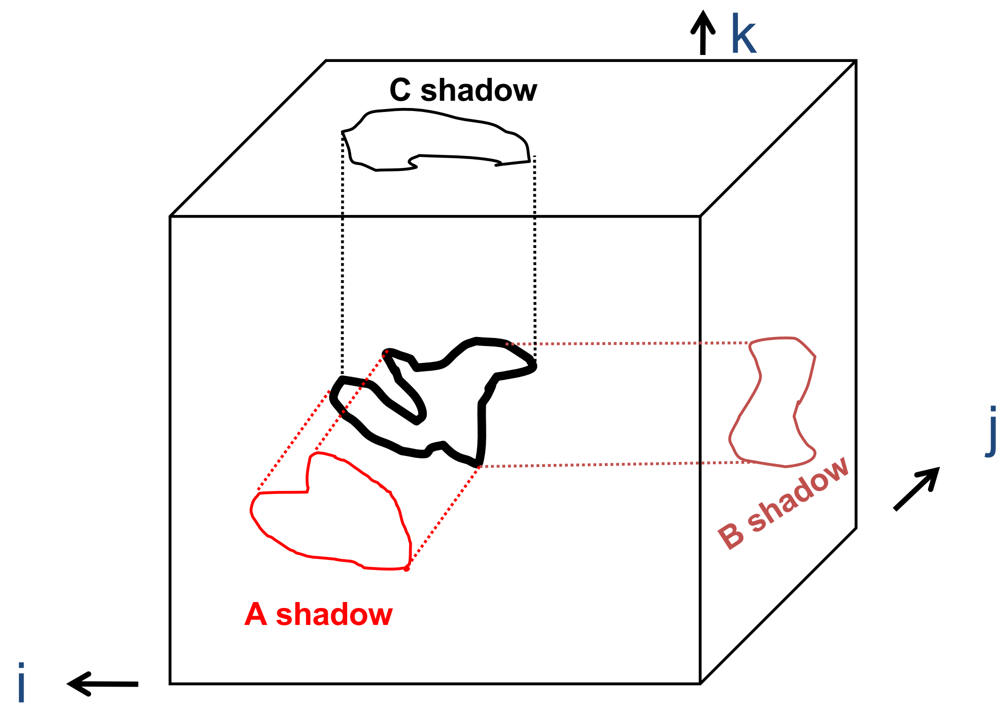
Proof of Communication Lower Bound on C = A·B (2/4)



- If we have at most M "A squares", "B squares", and "C squares" on faces, how many cubes can we have?

Proof of Communication Lower Bound on C = A·B (3/4)





 # cubes in black box with side lengths x, y and z
= Volume of black box
= x·y·z
= (xz · zy · yx)$^{1/2}$
= (#A□s · #B□s · #C□s)$^{1/2}$

(i,k) is in A shadow if (i,j,k) in 3D set
(j,k) is in B shadow if (i,j,k) in 3D set
(i,j) is in C shadow if (i,j,k) in 3D set

Thm (Loomis & Whitney, 1949)
 # cubes in 3D set = Volume of 3D set
≤ (area(A shadow) · area(B shadow) · area(C shadow)) $^{1/2}$

Proof of Communication Lower Bound on $C = A \cdot B$ (4/4)

- # loop iterations doable with M words of data = #cubes

  $\leq$ (area(A shadow) · area(B shadow) ·area(C shadow)) $^{1/2}$

  $\leq (M \cdot M \cdot M)^{1/2} = M^{3/2} = F$

- Need to read/write at least $M\, n^3 / F = \Omega(n^3/M^{1/2}) = \Omega(\#\text{loop iterations} / M^{1/2})$ words to/from cache

Recall optimal Matmul Algorithm

- Analogous to n-body:
  - What is the largest set of C(i,j)+=A(i,k)*B(k,j) we can perform given M entries A(i,k), B(k,j), C(i,j)?
  - What is the largest set of (i,j,k) we can have, given a bound M on the number of (i,k), (k,j), (i,j)?
  - What is the shape of the largest 3D volume we can have, given a bound M on the area of its shadows in 3 directions?
  - Answer: A cube, with edge length $O(M^{1/2})$, volume $O(M^{3/2})$
  - Optimal "blocked" Algorithm: 6 nested loops, 3 innermost loops do b x b matmul with $b = O(M^{1/2})$

Proof of Communication Lower Bound on C = A·B (4/4)

- # loop iterations doable with M words of data = #cubes

  $\leq$ (area(A shadow) · area(B shadow) ·area(C shadow)) $^{1/2}$

  $\leq (M \cdot M \cdot M)^{1/2} = M^{3/2} = F$

- Need to read/write at least $M\, n^3 / F = \Omega(n^3/M^{1/2}) = \Omega(\#\text{loop iterations} / M^{1/2})$ words to/from cache

- Parallel Case: apply reasoning to one processor out of P
  - "Fast memory" = local processor, "Slow memory" = other procs
  - Goal: lower bound # "reads/writes" = # words moved between one processor and others
  - # loop iterations = n^3 / P (load balanced)
  - $M = 3n^2 / P$ (each processor gets equal fraction of data)
  - # "reads/writes" $\geq M \cdot (n^3 /P) / (M)^{3/2} = \Omega(n^2 / P^{1/2})$

Recursive Matrix Multiplication (RMM) (1/2)

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = A \cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad \begin{bmatrix} A_{11} * B_{11} + A_{12} * B_{21} & A_{11} * B_{12} + A_{12} * B_{22} \\ A_{21} * B_{11} + A_{22} * B_{21} & A_{21} * B_{12} + A_{22} * B_{22} \end{bmatrix}$$

- Eventually, the matrices will fit in cache
- Don’t need to optimized for size ☺
- But call overhead is high ☹

Recursive Matrix Multiplication (2/2)

```
func C = RMM (A, B, n)
   if n=1, C = A * B, else
     {  C11 = RMM (A11 , B11 , n/2) + RMM (A12 , B21 , n/2)
        C12 = RMM (A11 , B12 , n/2) + RMM (A12 , B22 , n/2)
        C21 = RMM (A21 , B11 , n/2) + RMM (A22 , B21 , n/2)
        C22 = RMM (A21 , B12 , n/2) + RMM (A22 , B22 , n/2)  }
   return
```

$A(n)$ = # arithmetic operations in RMM(. , . , n)

$= 8 \cdot A(n/2) + 4(n/2)^2$ if $n > 1$, else 1

$= 2n^3 - n^2$ … same operations as usual, in different order

$W(n)$ = # words moved between fast, slow memory by RMM(. , . , n)

$= 8 \cdot W(n/2) + 4 \cdot 3(n/2)^2$ if $3n^2 > M_{fast}$, else $3n^2$

$= O(n^3 / (M_{fast})^{1/2} + n^2)$ … same as blocked matmul

Don't need to know M_{fast} for this to work!

Strassen's Matrix Multiply

- The traditional algorithm (with or without tiling) has $O(n^3)$ flops
- Strassen discovered an algorithm with asymptotically lower flops
  - $O(n^{2.81})$
- Consider a 2x2 matrix multiply, normally takes 8 multiplies, 4 adds
  - Strassen does it with 7 multiplies and 18 adds

Let $C = \begin{pmatrix} C11 & C12 \\ C21 & C22 \end{pmatrix} = \begin{pmatrix} A11 & A12 \\ A21 & A22 \end{pmatrix} \begin{pmatrix} B11 & B12 \\ B21 & B22 \end{pmatrix}$

Let P1 = (A12 - A22) * (B21 + B22)

P2 = (A11 + A22) * (B11 + B22)

P3 = (A11 - A21) * (B11 + B12)

P4 = (A11 + A12) * B22

P5 = A11 * (B12 - B22)

P6 = A22 * (B21 - B11)

P7 = (A21 + A22) * B11

Then C11 = P1 + P2 - P4 + P6

C12 = P4 + P5

C21 = P6 + P7

C22 = P2 - P3 + P5 - P7

Extends to nxn by divide&conquer

Strassen (continued)

T(n) = Cost of multiplying nxn matrices

$$= 7^*T(n/2) + 18^*(n/2)^2$$

$$= O(n^{\log_2 7})$$

$$= O(n^{2.81})$$

- Asymptotically faster
  - Several times faster for large n in practice
  - Cross-over depends on machine
  - "Tuning Strassen's Matrix Multiplication for Memory Efficiency", M. S. Thottethodi, S. Chatterjee, and A. Lebeck, in Proceedings of Supercomputing '98

- Possible to extend communication lower bound to Strassen
  - #words moved between fast and slow memory = $\Omega(n^{\log 2\ 7} / M^{(\log 2\ 7)/2 - 1}) \sim \Omega(n^{2.81} / M^{0.4})$ (Ballard, D., Holtz, Schwartz, 2011, **SPAA Best Paper Prize**)
  - Attainable too, more on parallel version later

Other Fast Matrix Multiplication Algorithms

- World's record was $O(n^{2.37548...})$
  - Coppersmith & Winograd, 1987
- New Record! 2.37*548* reduced to 2.37*293*
  - Virginia Vassilevska Williams, UC Berkeley & Stanford, 2011
- Newer Record! 2.372*93* reduced to 2.372*86*
  - Francois Le Gall, 2014
- Lower bound on #words moved can be extended to (some) of these algorithms (2015 thesis of Jacob Scott)
- Possibility of $O(n^{2+\varepsilon})$ algorithm!
  - Cohn, Umans, Kleinberg, 2003
- Can show they all can be made numerically stable
  - Demmel, Dumitriu, Holtz, Kleinberg, 2007
- Can do rest of linear algebra (solve $Ax=b$, $Ax=\lambda x$, etc) as fast , and numerically stably
  - Demmel, Dumitriu, Holtz, 2008
- Fast methods (besides Strassen) may need unrealistically large n

Approach to generalizing lower bounds

- Matmul

  for i=1:n, for j=1:n, for k=1:n,

  C(i,j)+=A(i,k)*B(k,j)

=> for (i,j,k) in S = subset of Z^3

  Access locations indexed by (i,j), (i,k), (k,j)

- General case

  for i1=1:n, for i2 = i1:m, … for ik = i3:i4

  C(i1+2*i3-i7) = func(A(i2+3*i4,i1,i2,i1+i2,…),B(pnt(3*i4)),…)

  D(something else) = func(something else), …

=> for (i1,i2,…,ik) in S = subset of Z^k

  Access locations indexed by “projections”, eg

  ϕ_C (i1,i2,…,ik) = (i1+2*i3-i7)

  ϕ_A (i1,i2,…,ik) = (i2+3*i4,i1,i2,i1+i2,…), …

- Goal: Communication lower bounds, optimal algorithms for *any* program that looks like this

General Communication Lower Bound

- Thm: Given a program with array refs given by projections ϕ_j, then there is an $s_{HBL} \geq 1$ such that

  #words_moved = Ω (#iterations/M^{sHBL-1})

  where s_{HBL} is the the value of a linear program:

  minimize $s_{HBL} = \Sigma_j \, e_j$ subject to

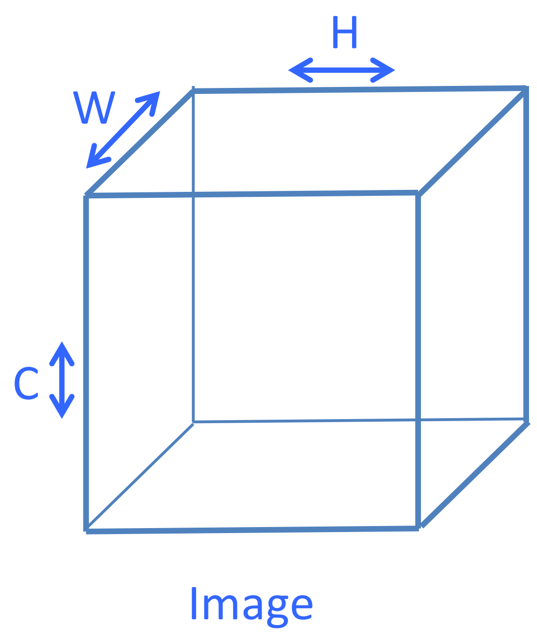
  $\text{rank}(H) \leq \Sigma_j \, e_j * \text{rank}(\phi_j(H))$ for all subgroups $H < Z^k$

- Proof depends on recent result in pure mathematics by Christ/Tao/Carbery/Bennett
  - Generalization of Hölder-Brascamp-Lieb (HBL) inequality to Abelian groups
  - HBL generalizes Cauchy-Schwartz, Loomis-Whitney, …

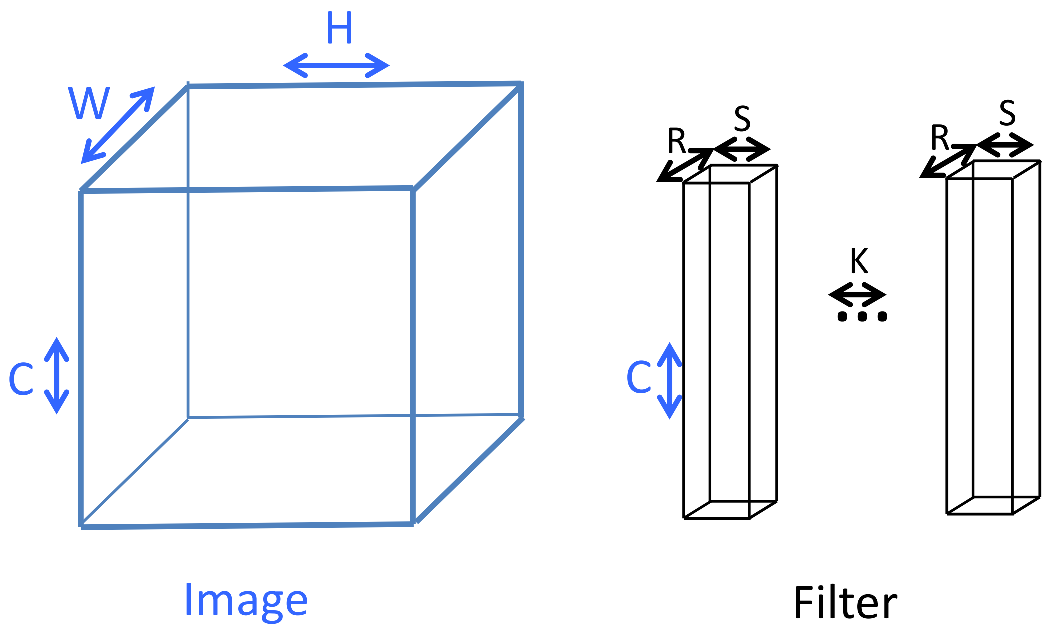
Is this bound attainable?

- Thm: We can always construct an optimal tiling, that attains the lower bound
- Assumptions/caveats/open questions
  - Attains lower bound Ω (#iterations/M^{sHBL-1}) in O() sense
  - Depends on loop dependencies
    - Not all tilings may compute the right answer
    - Best case: no dependencies, or just reductions (like matmul)
  - Assumes loop bounds are large enough to fit tile
    - Ex: same lower bound for matmul applies to matrix-vector-multiply, but not attainable

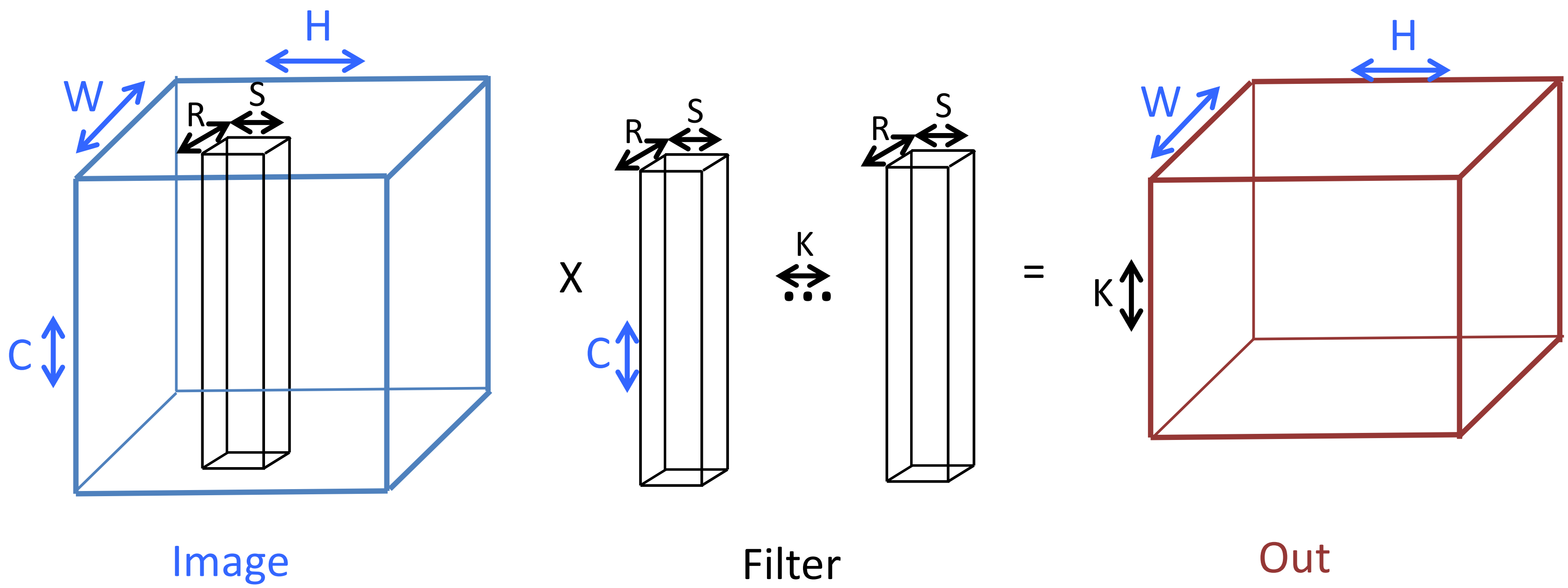
What CNNs compute



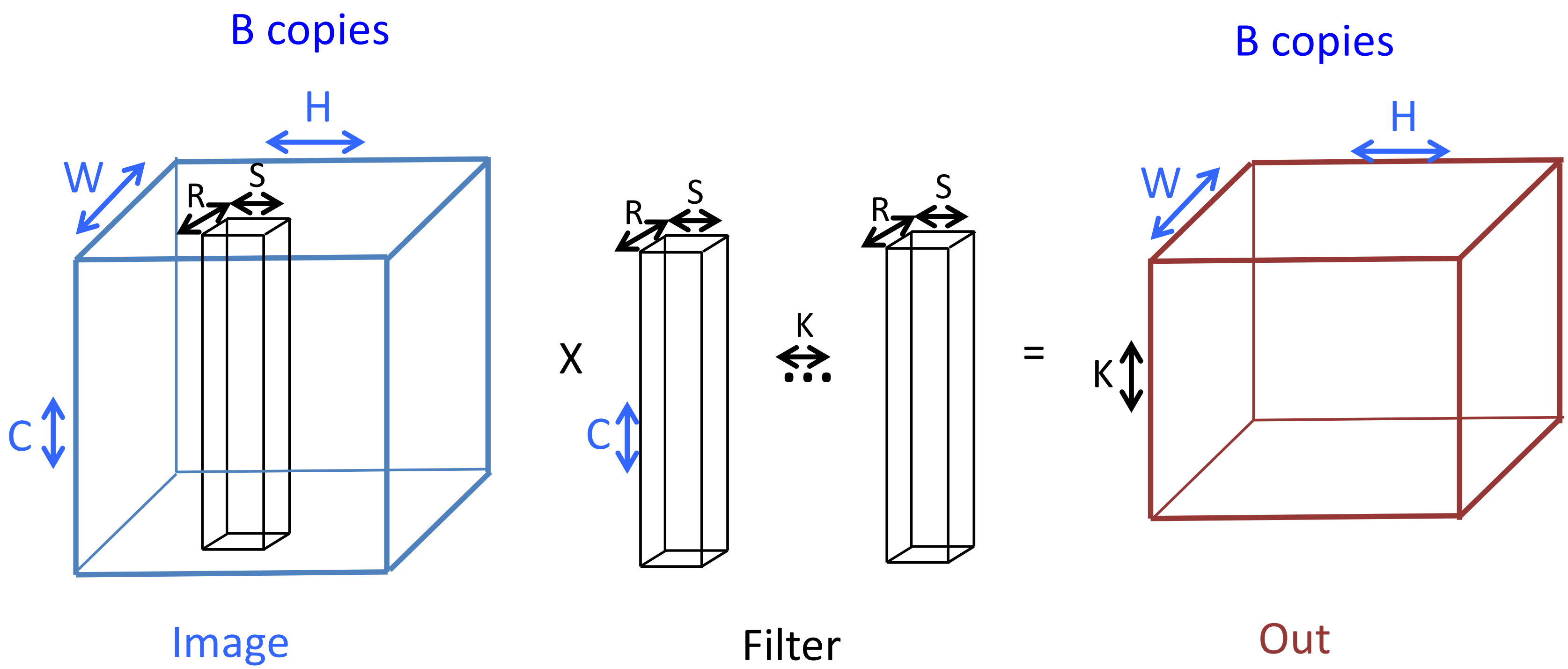
What CNNs compute



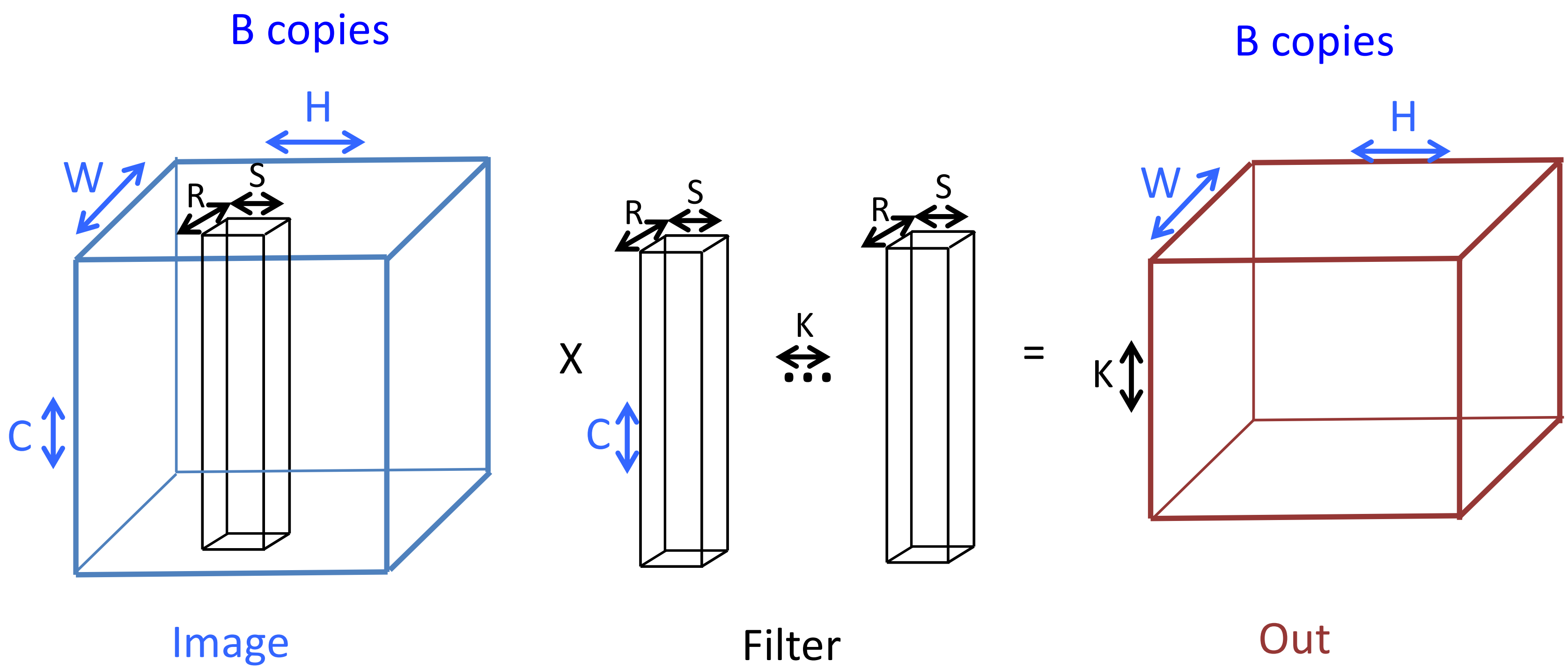
What CNNs compute



What CNNs compute

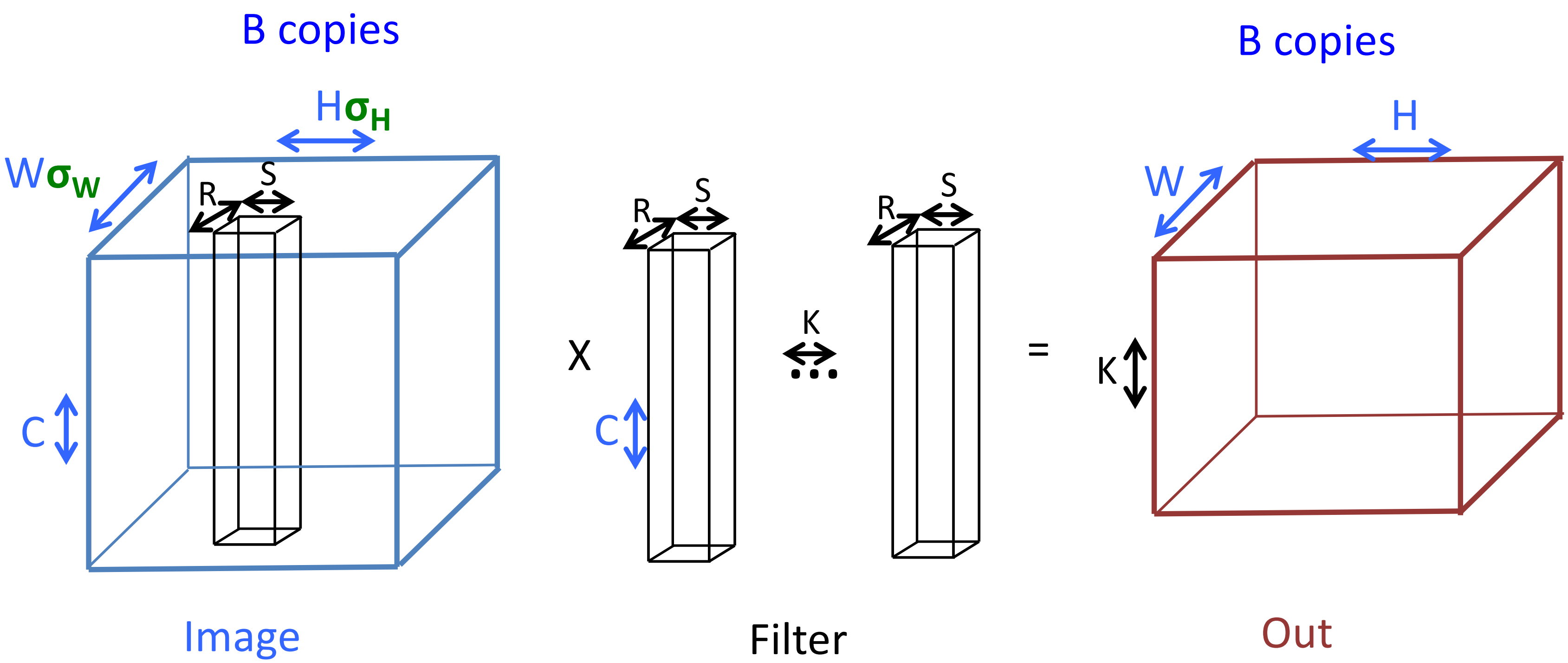


What CNNs compute



for k=1:K, for h=1:H, for w=1:W, for r=1:R,
for s=1:S, for c=1:C, for b=1:B
Out(k, h, w, b) += Image(r+w, s+h, c, b) * Filter(k, r, s, c)

What CNNs compute



for k=1:K, for h=1:H, for w=1:W, for r=1:R,
for s=1:S, for c=1:C, for b=1:B

$$\text{Out}(k, h, w, b) \mathrel{+}= \text{Image}(r+\sigma_W w, s+\sigma_H h, c, b) * \text{Filter}(k, r, s, c)$$

Communication Lower Bound for CNNs

- Let N = #iterations = KHWRSCB, M = cache size
- #words moved = Ω(max(... 5 terms

  BKHW, ... size of Out

  $\sigma_H \sigma_W$BCWH, ... size of Image

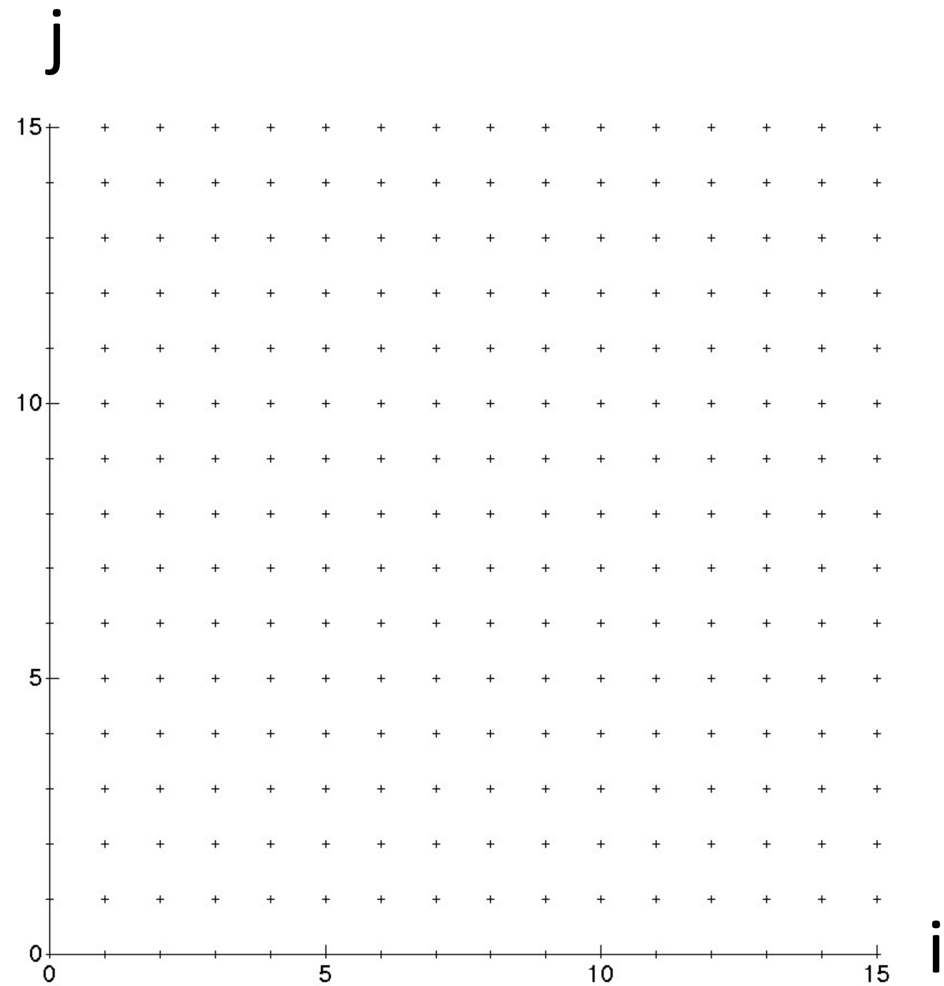
  CKRS, ... size of Filter

  N/M, ... lower bound from n-body

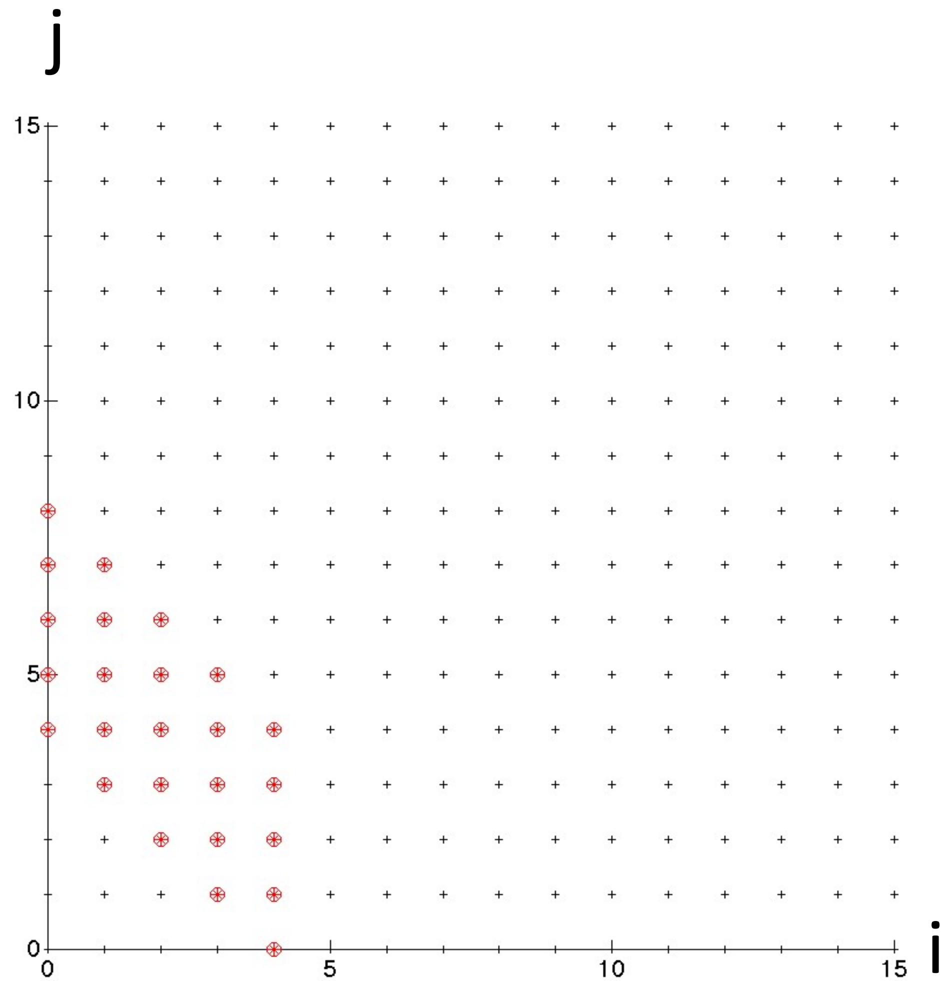
  $N/(M^{1/2} (RS/(\sigma_H \sigma_W))^{1/2})$ ... new lower bound)
- New lower bound
  - Beats matmul by factor $(RS/(\sigma_H \sigma_W))^{1/2}$
  - Applies in common case when data does not fit in cache, but one RxS filter does
  - Tile needed to attain N/M too big to fit in loop bounds
- Attainable (many cases)

Optimal tiling for “slanted” n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)
```



Optimal tiling for “slanted” n-body

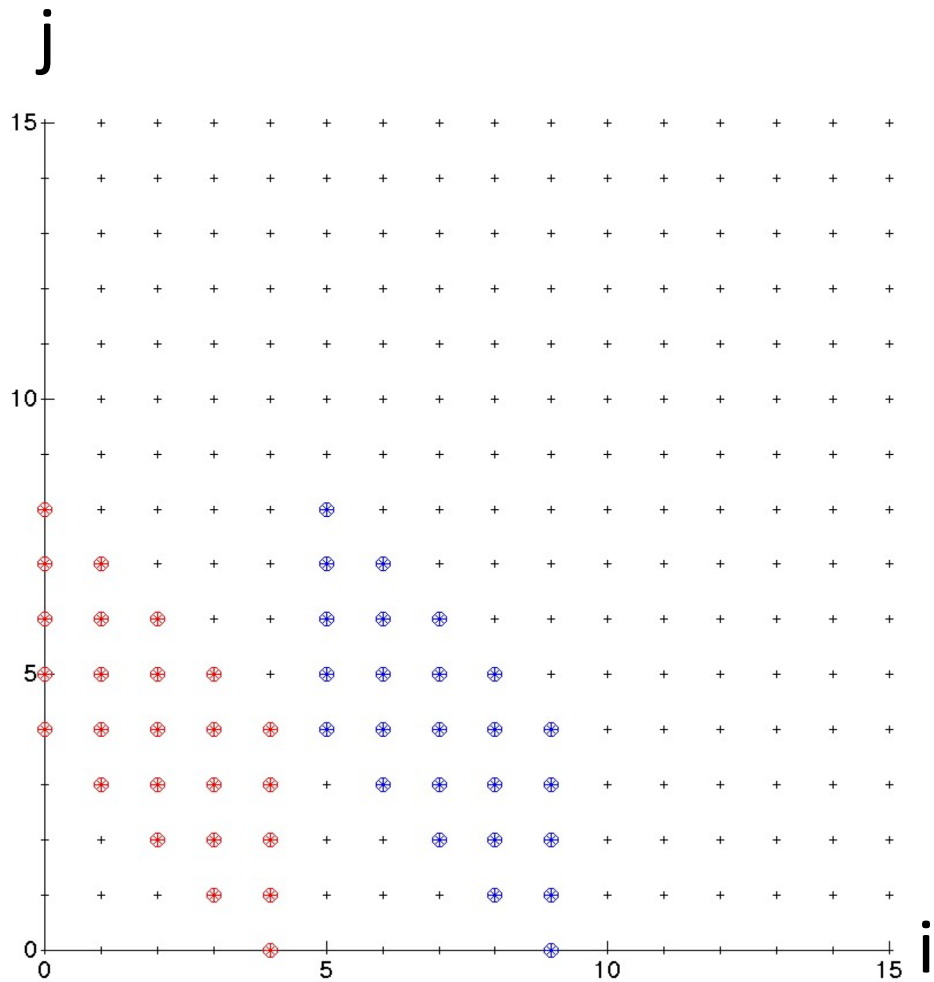


for i = 0:n
  for j = 0:n
    access A(i), B(i+j)

Tiling:
  Read 5 entries of A:
    A([0,1,2,3,4])
  Read 5 entries of B:
    B([4,5,6,7,8])
  Perform $5^2 = 25$
    loop iterations

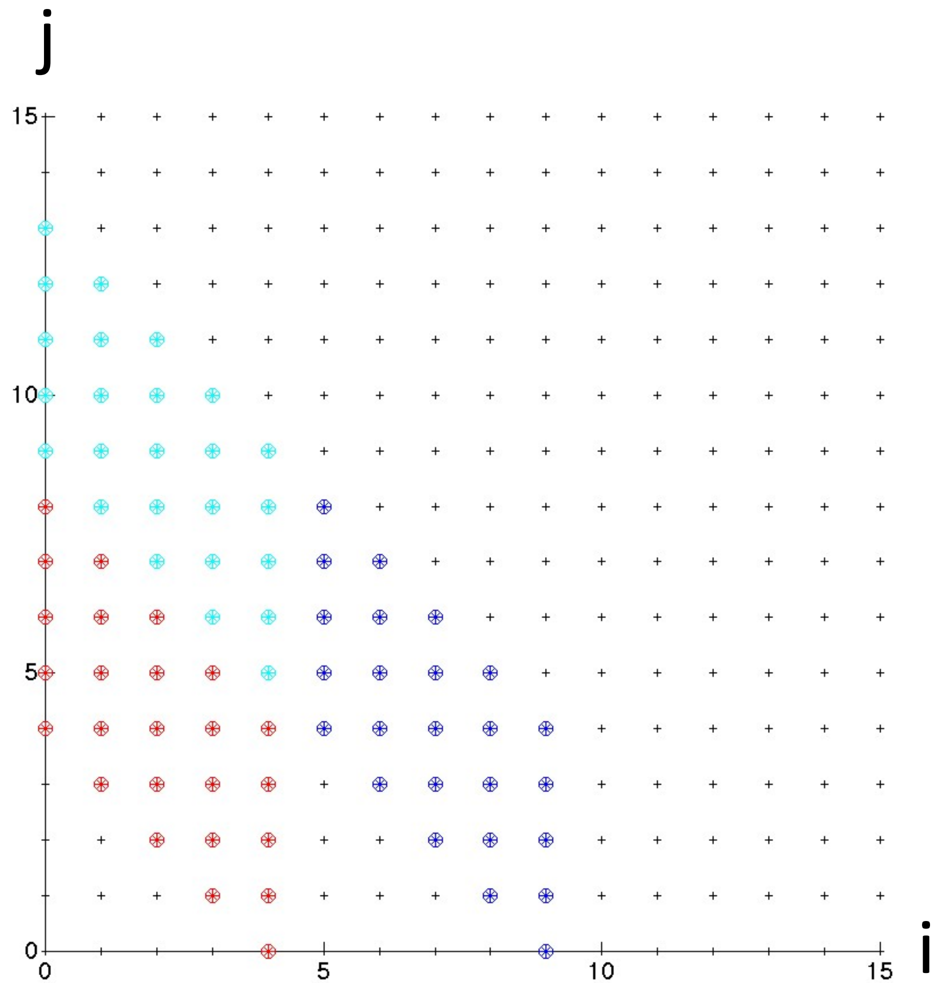
Optimal tiling for "slanted" n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)
```



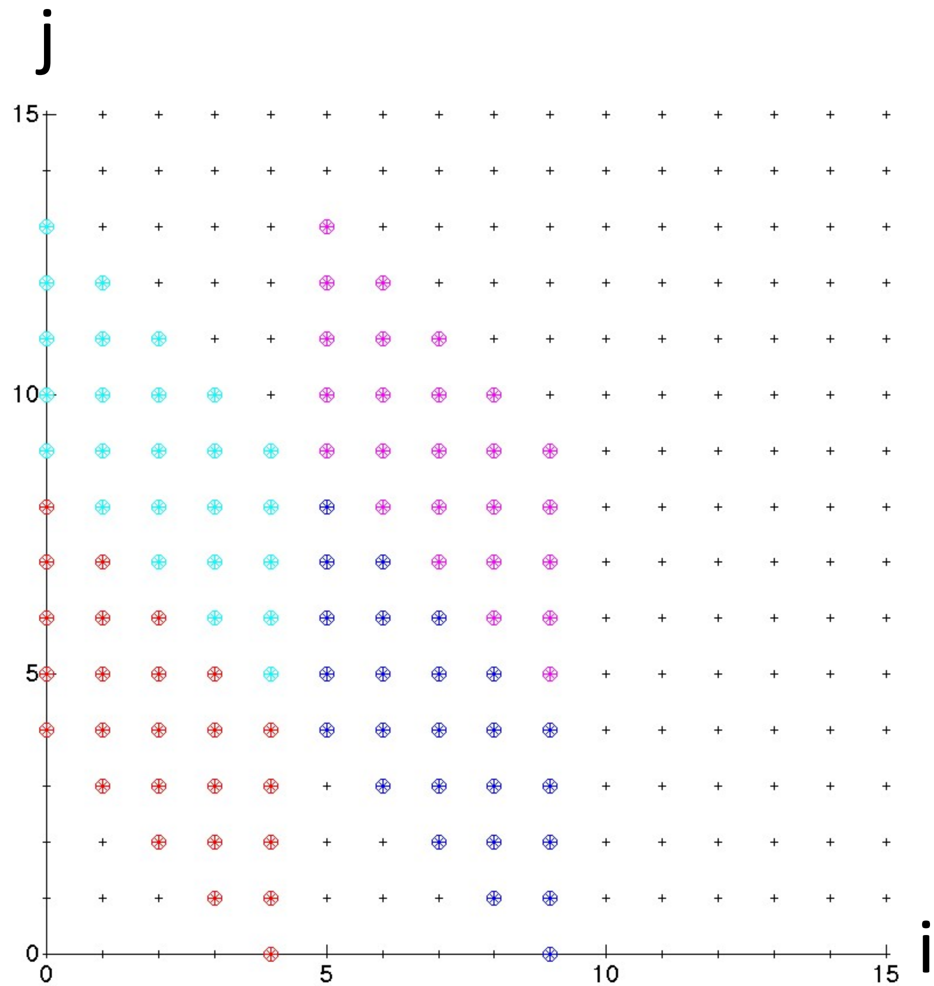
Optimal tiling for “slanted” n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)
```



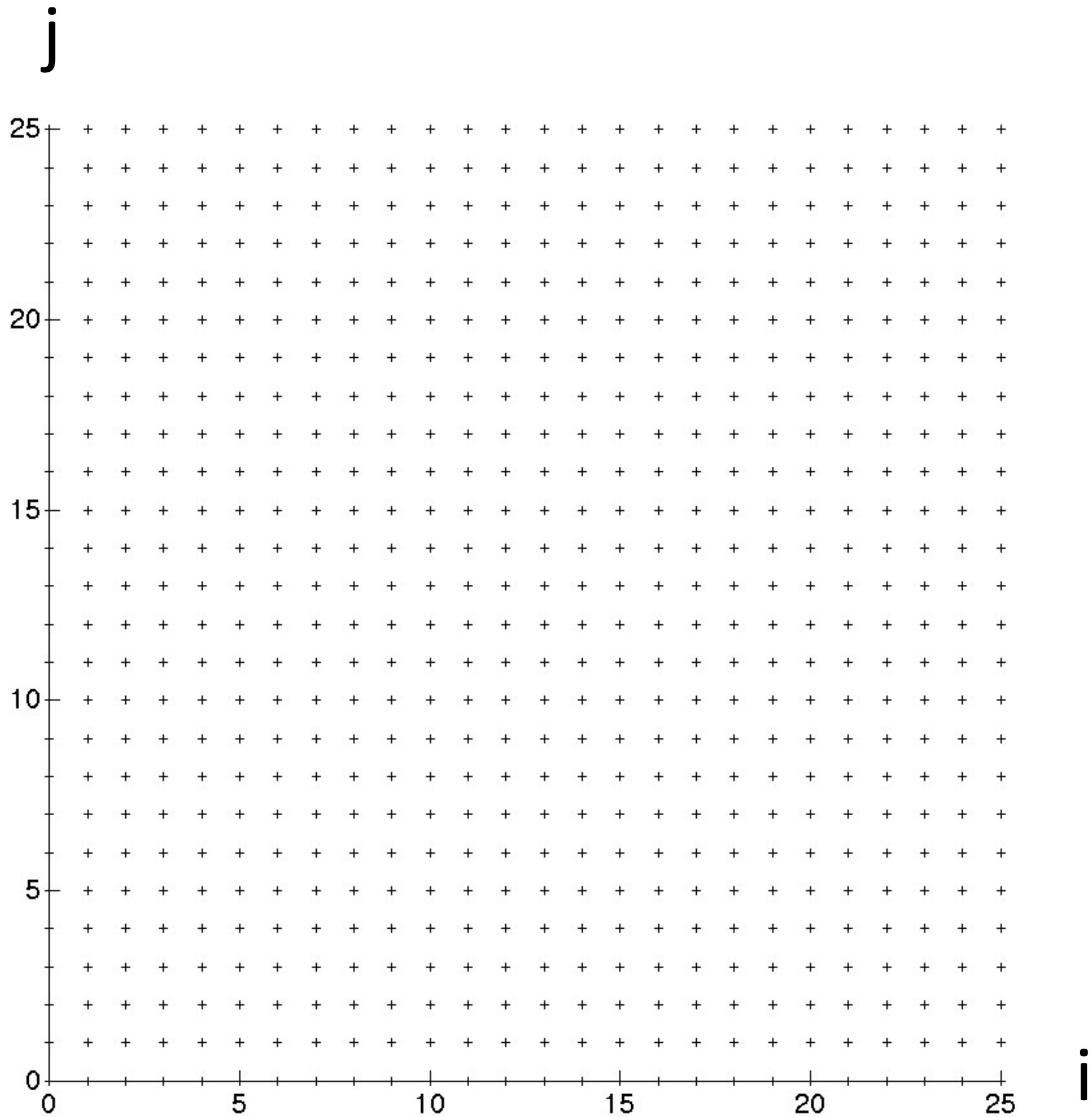
Optimal tiling for "slanted" n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)
```



Optimal tiling for "twisted" n-body

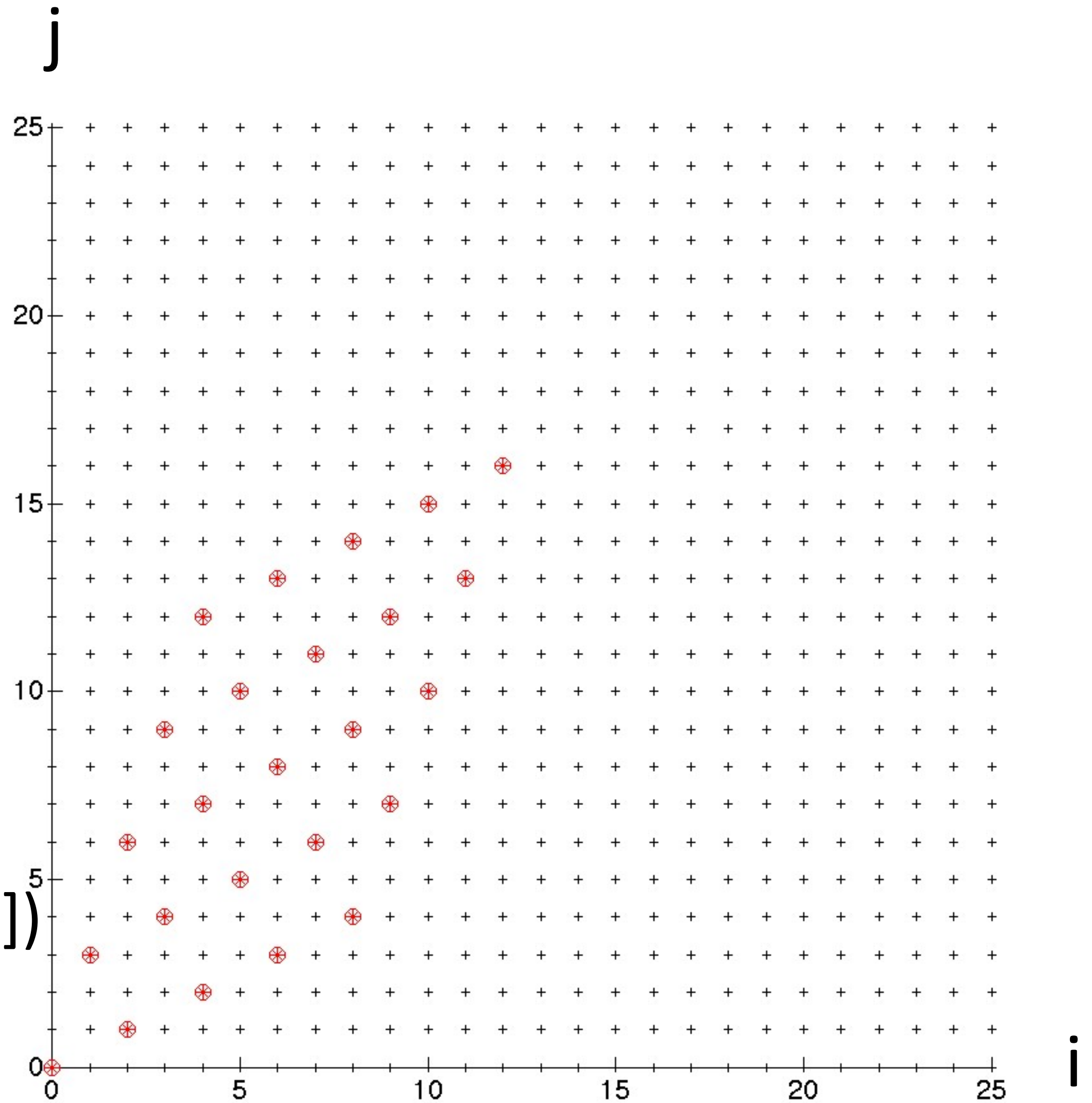
```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



Optimal tiling for “twisted” n-body

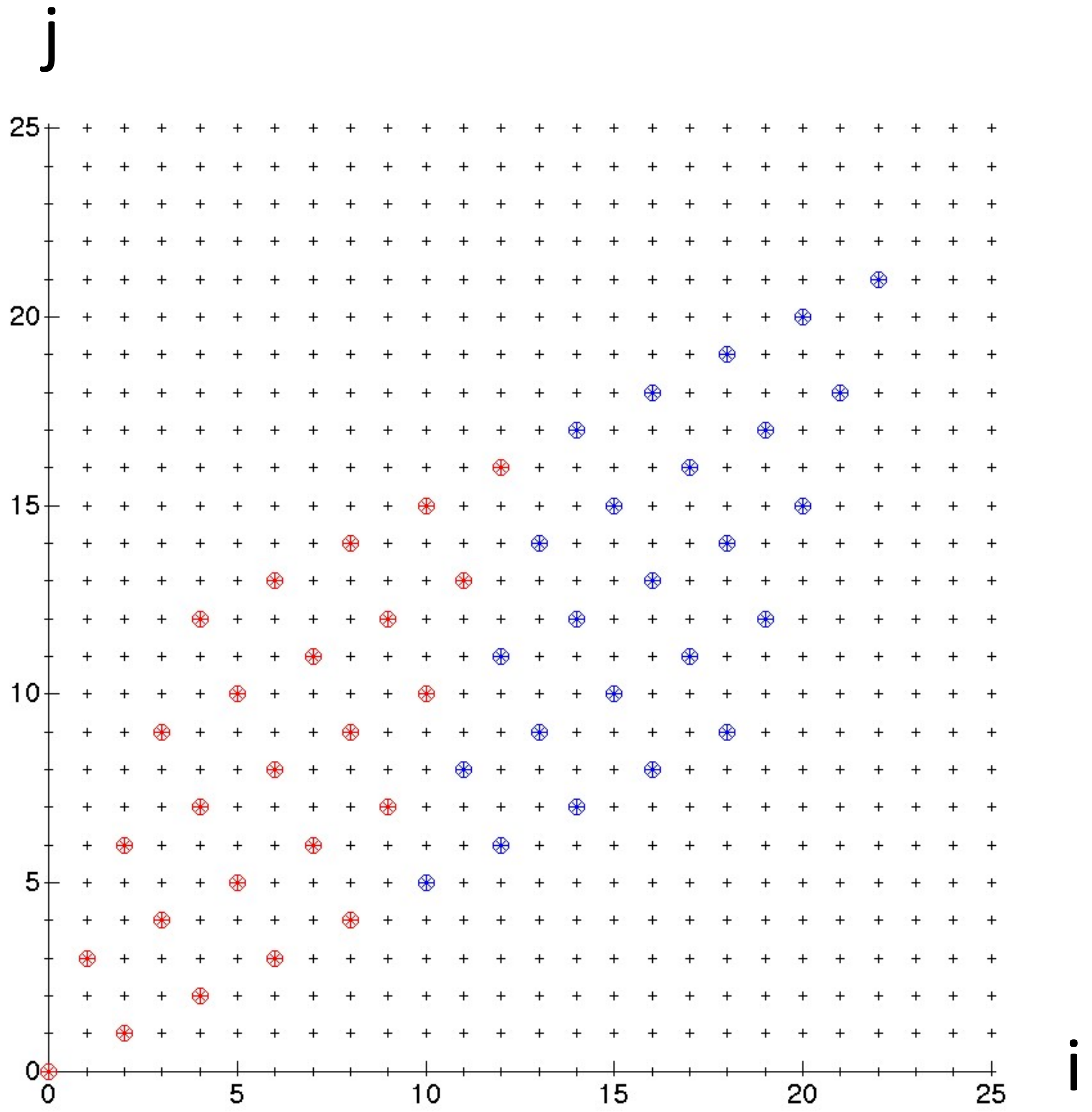
for i = 0:n
  for j = 0:n
    access A(3*i-j),
          B(i-2*j)

Tiling:
  Read 5 entries of A:
    A([0,5,10,15,20])
  Read 5 entries of B:
    B([0,-5,-10,-15,-20])
  Perform 5^2 = 25
    loop iterations



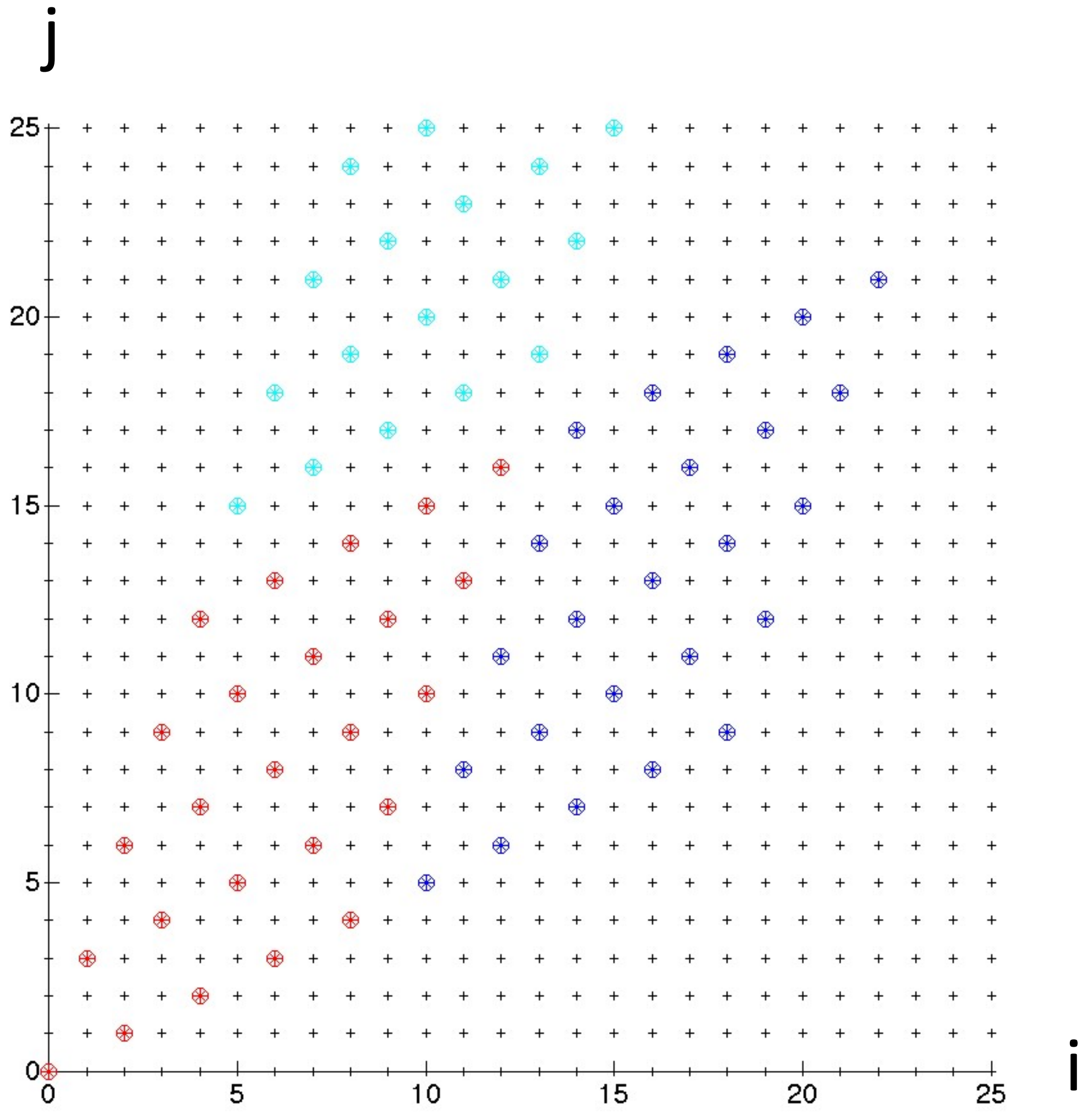
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
             B(i-2*j)
```



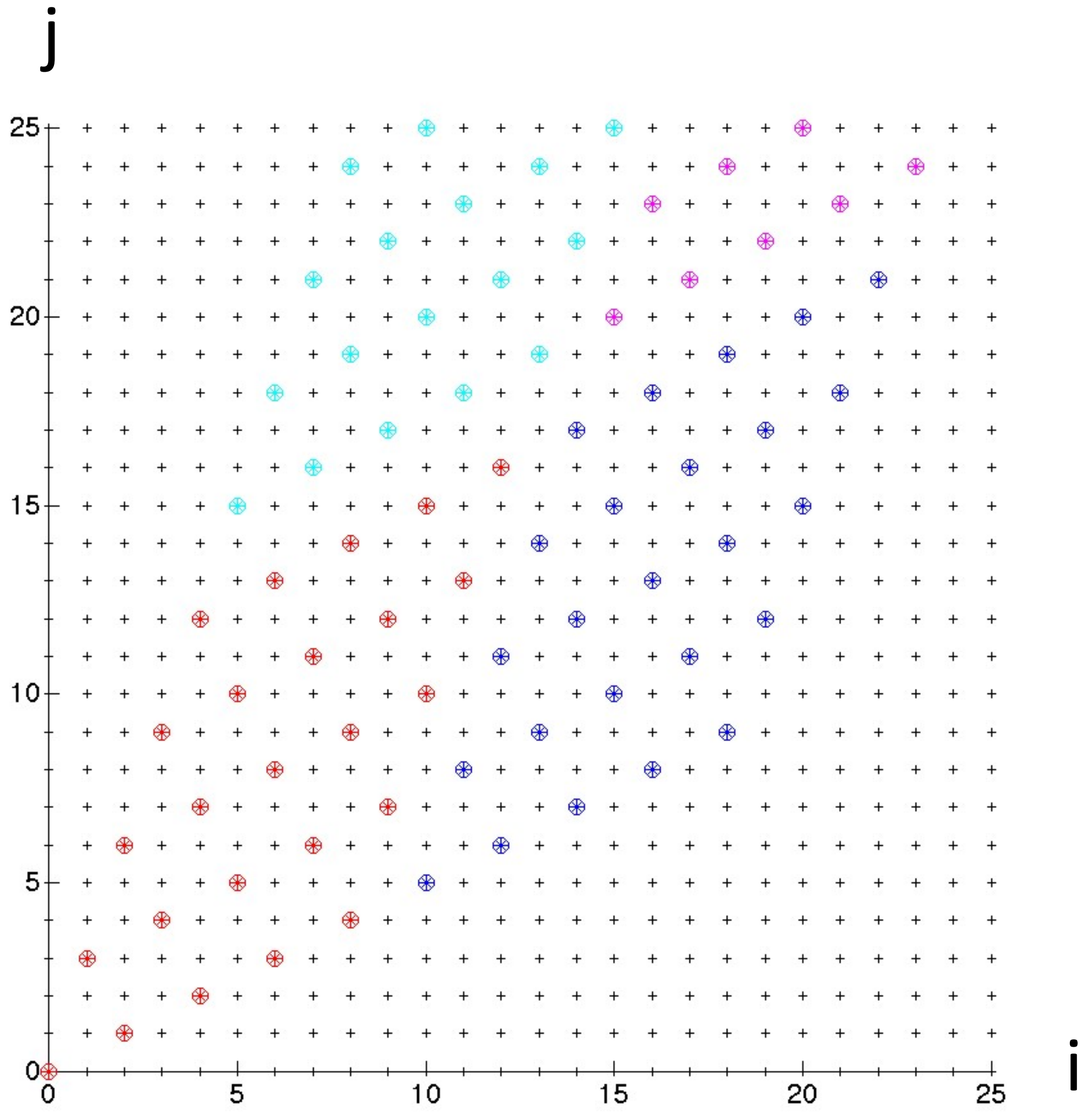
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



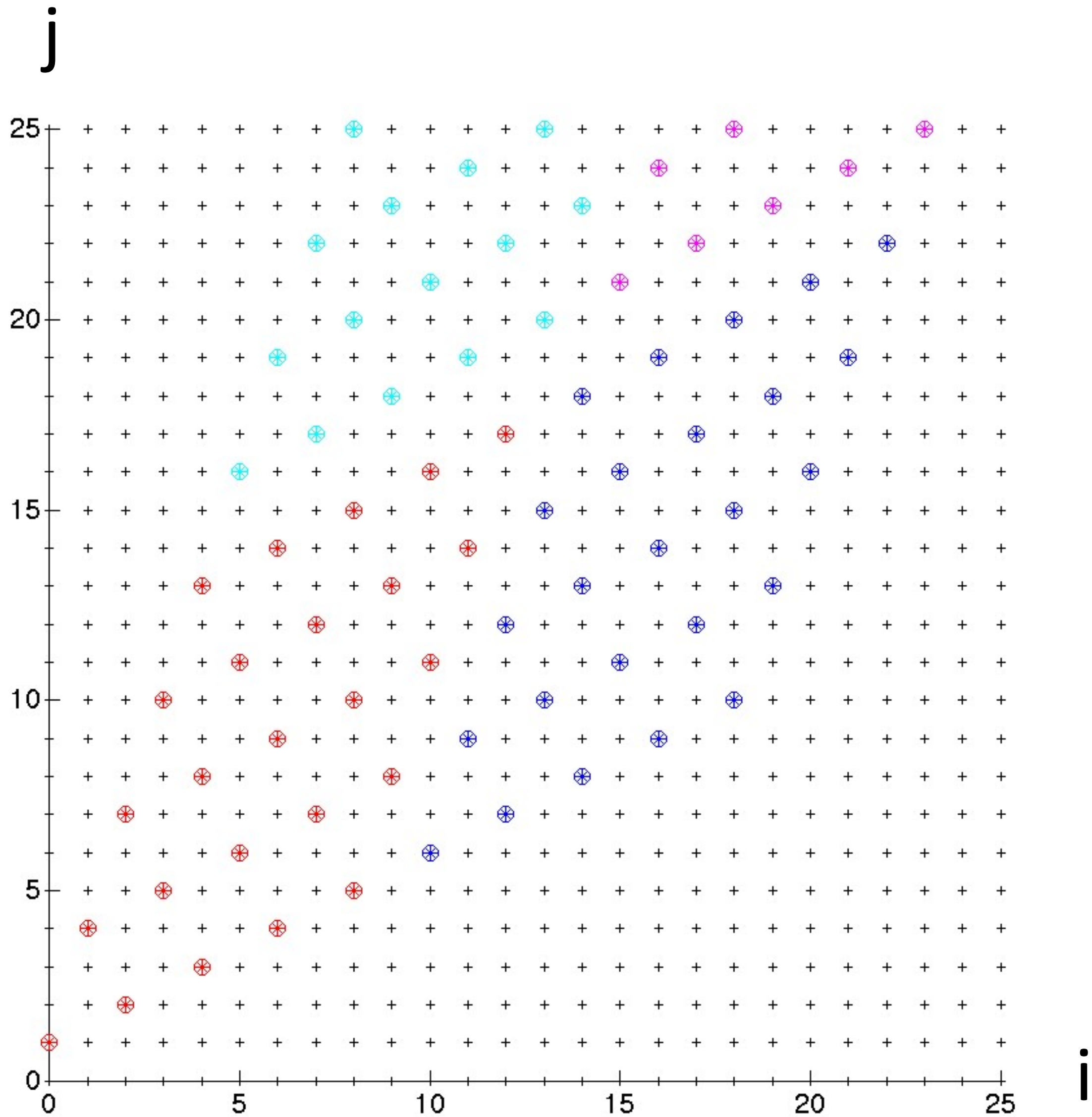
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



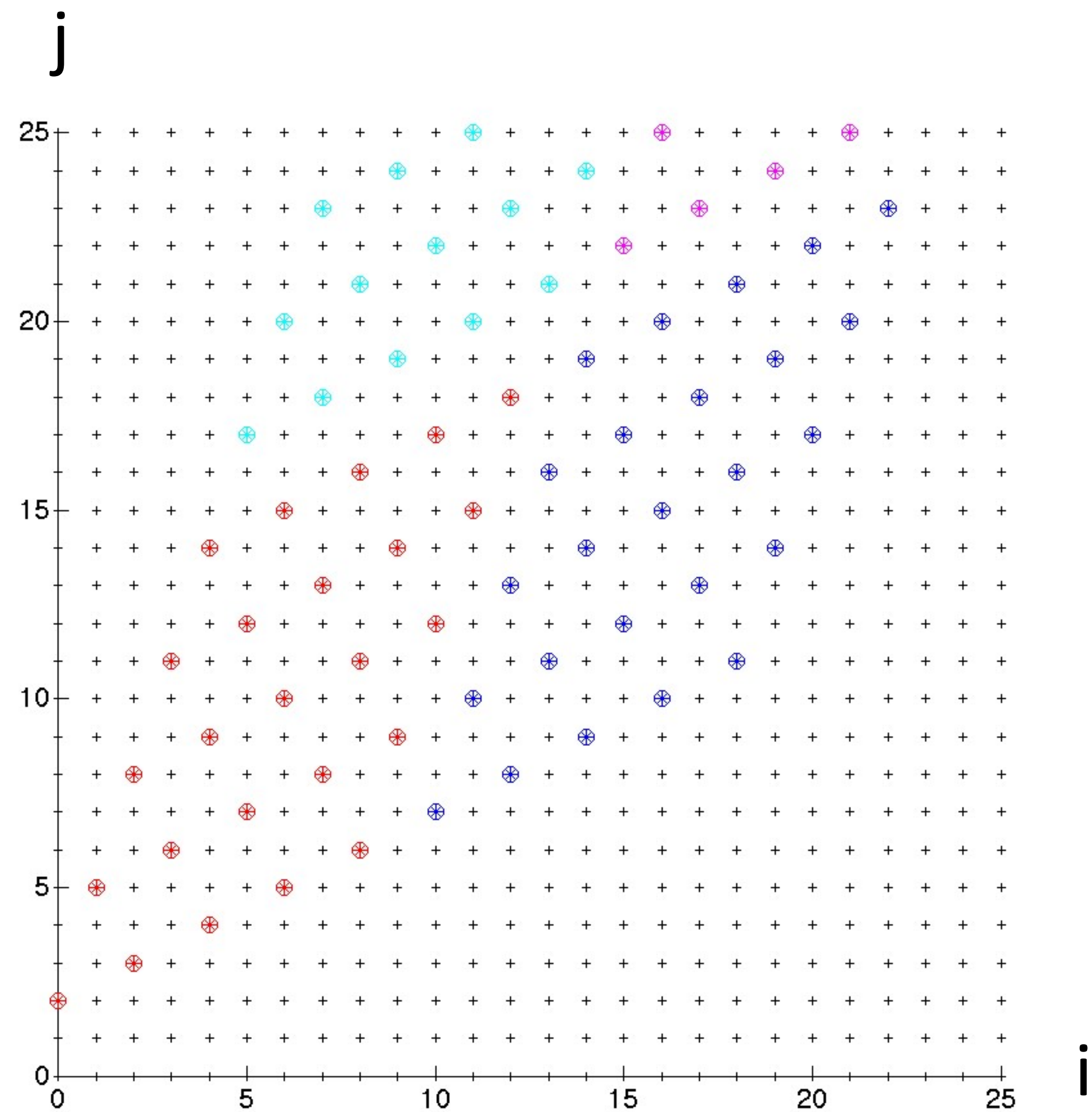
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
            B(i-2*j)
```



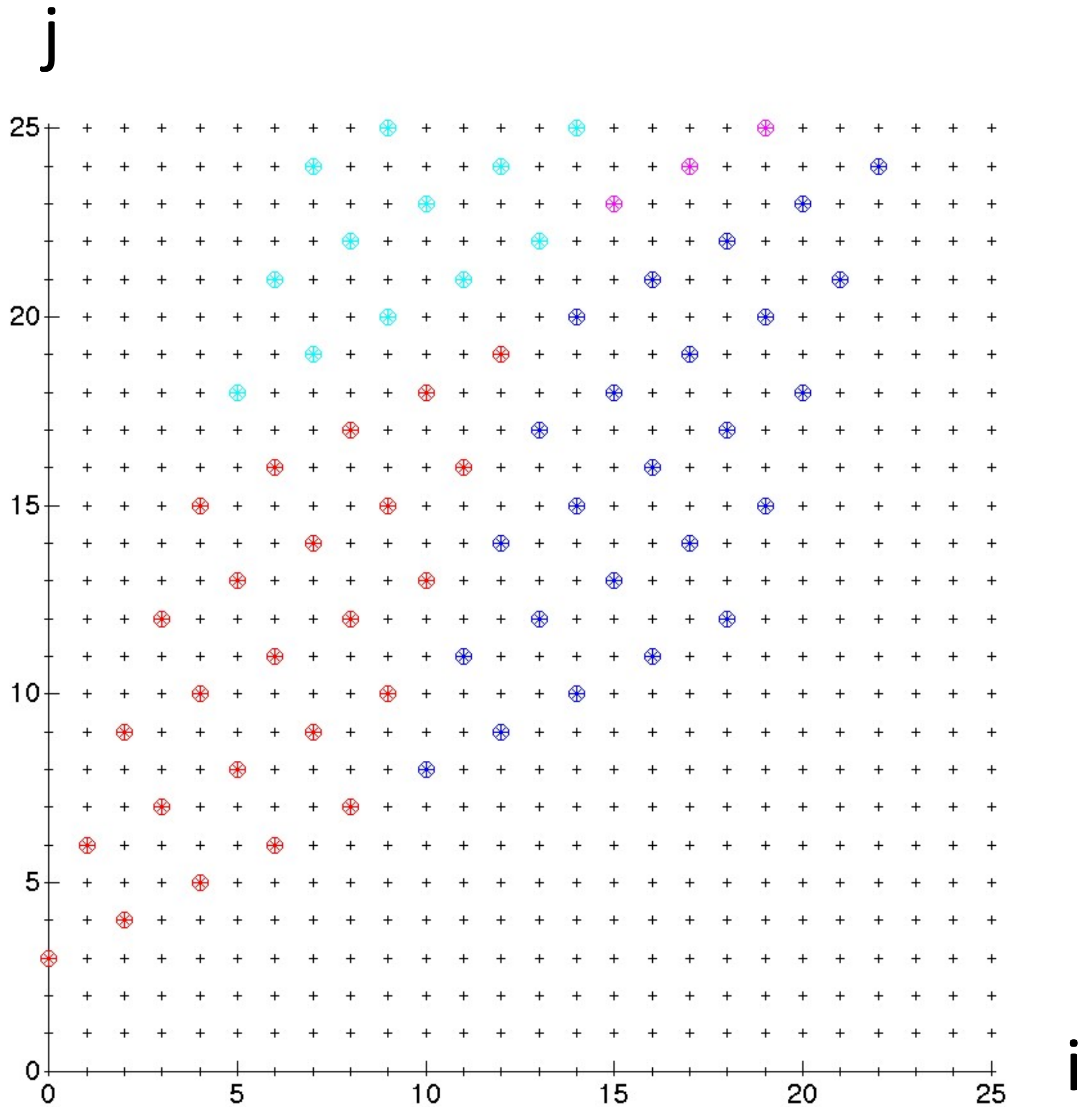
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
            B(i-2*j)
```



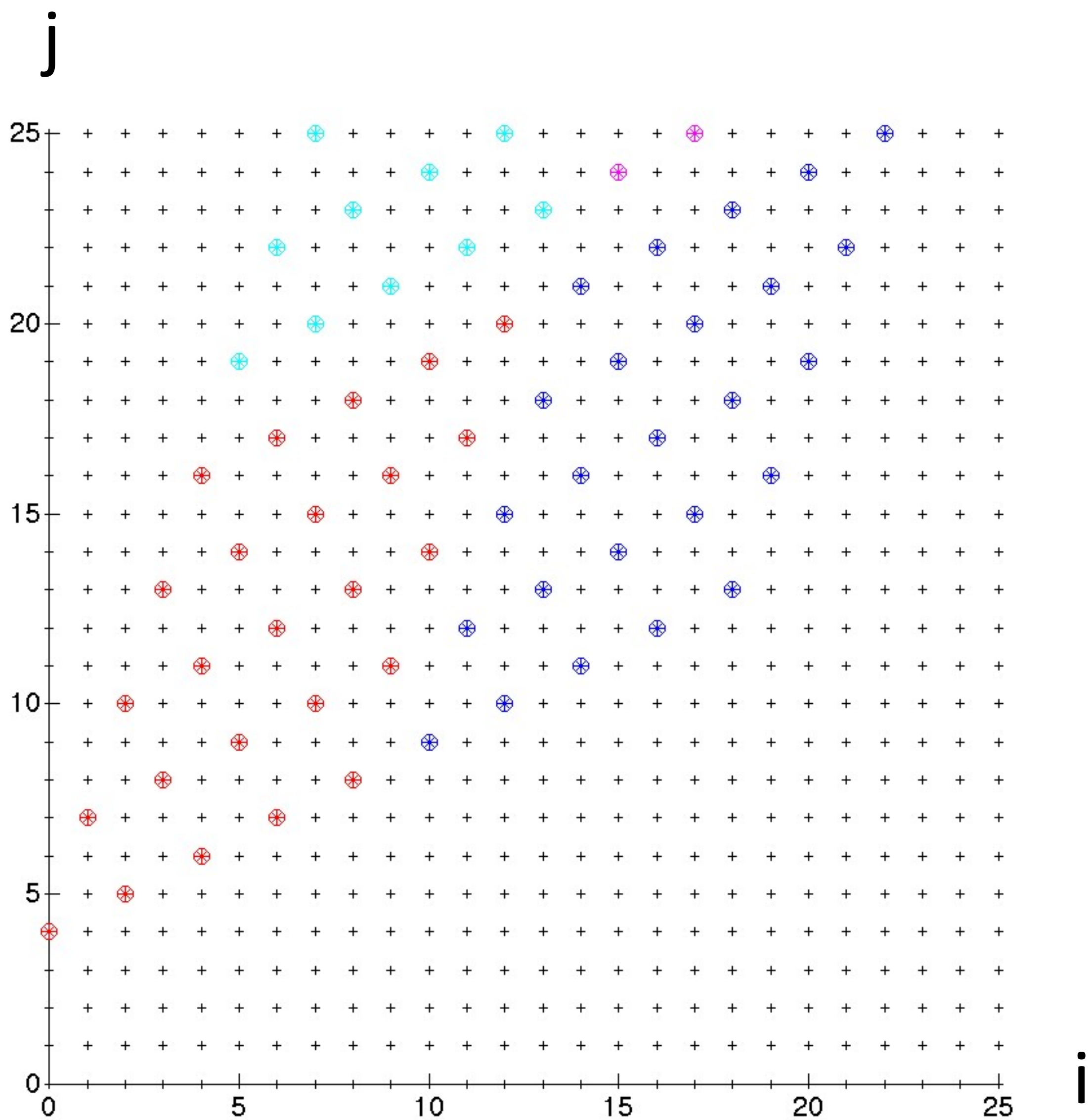
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```