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

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

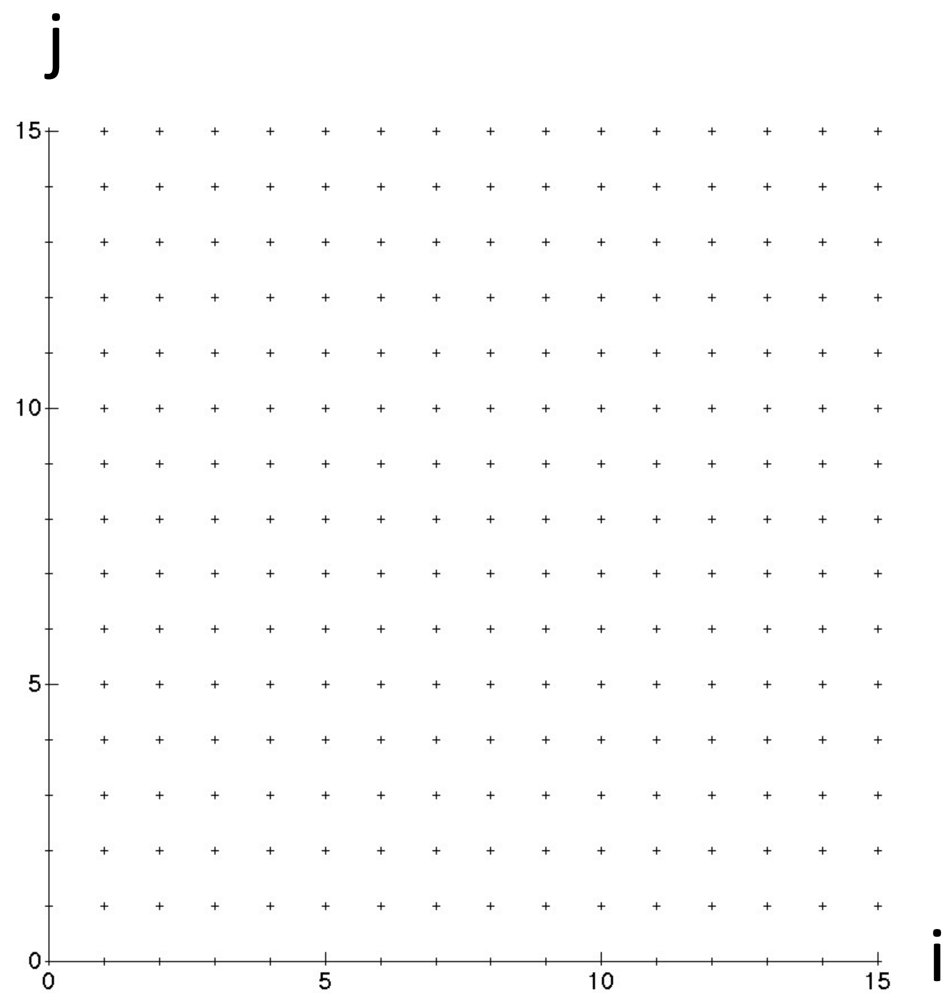
Outline

- 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

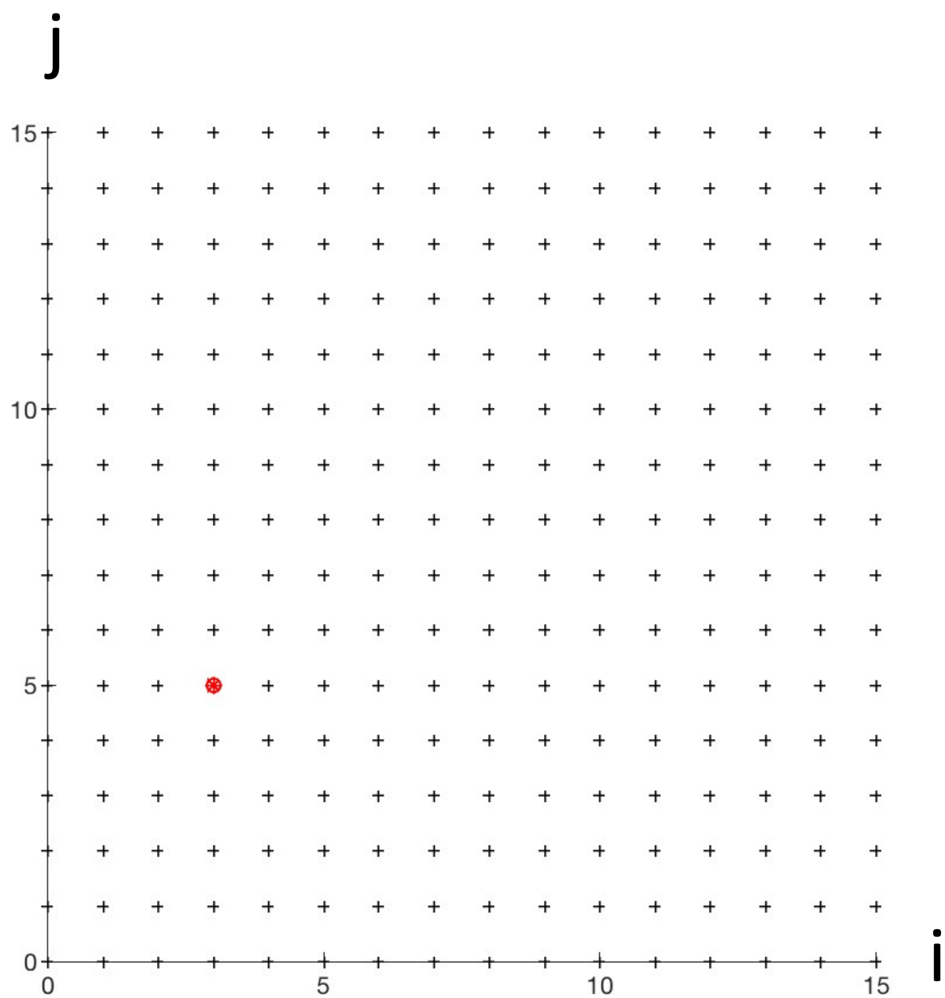


for i = 0:n
  for j = 0:n
    access A(i), B(j)

Data access for n-body

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

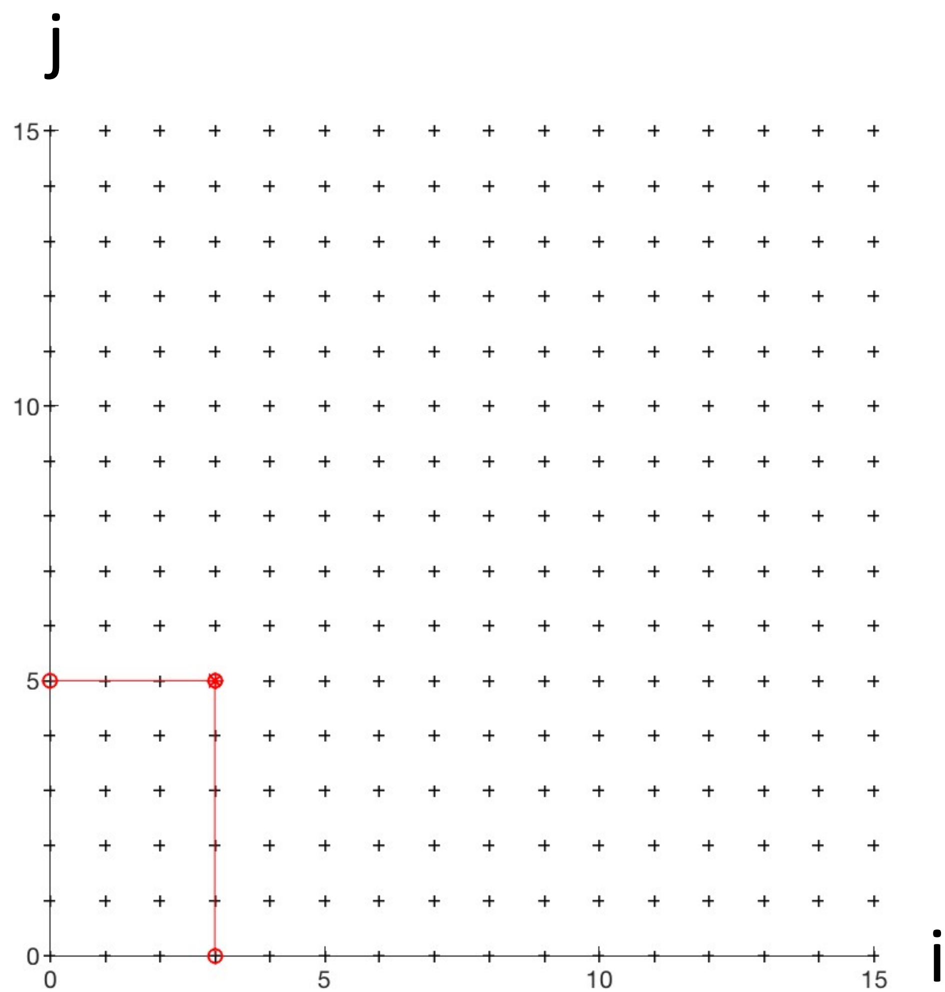
Ex: execute loop for
    i = 3, j = 5



Data access for n-body

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

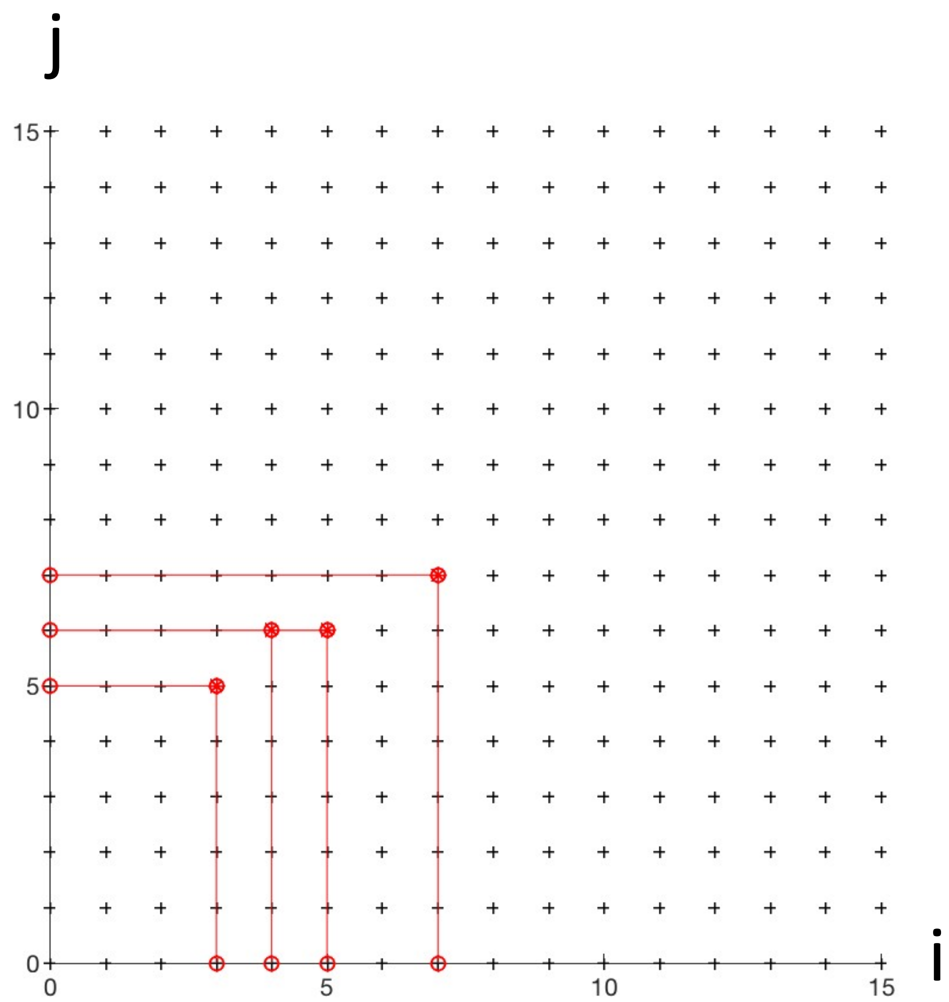
Ex: execute loop for
   i = 3, j = 5
access A(3), B(5)



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)

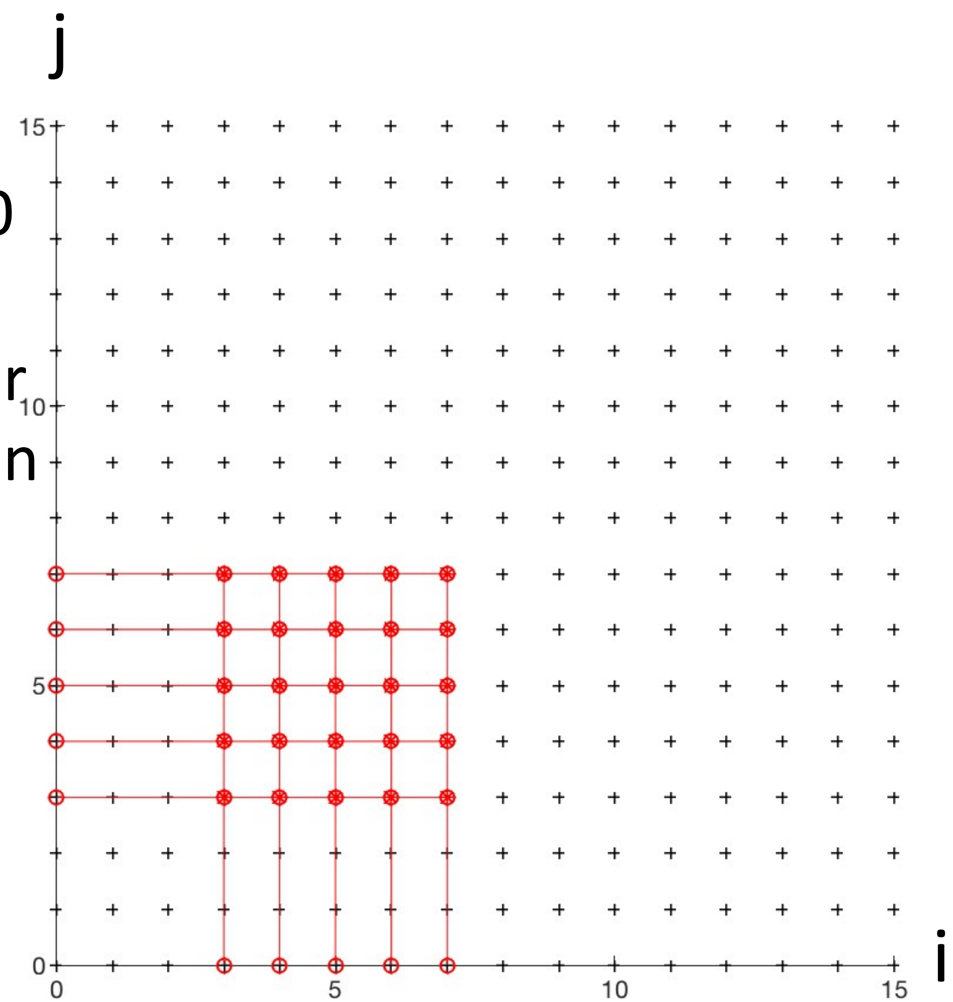


Data access for n-body

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



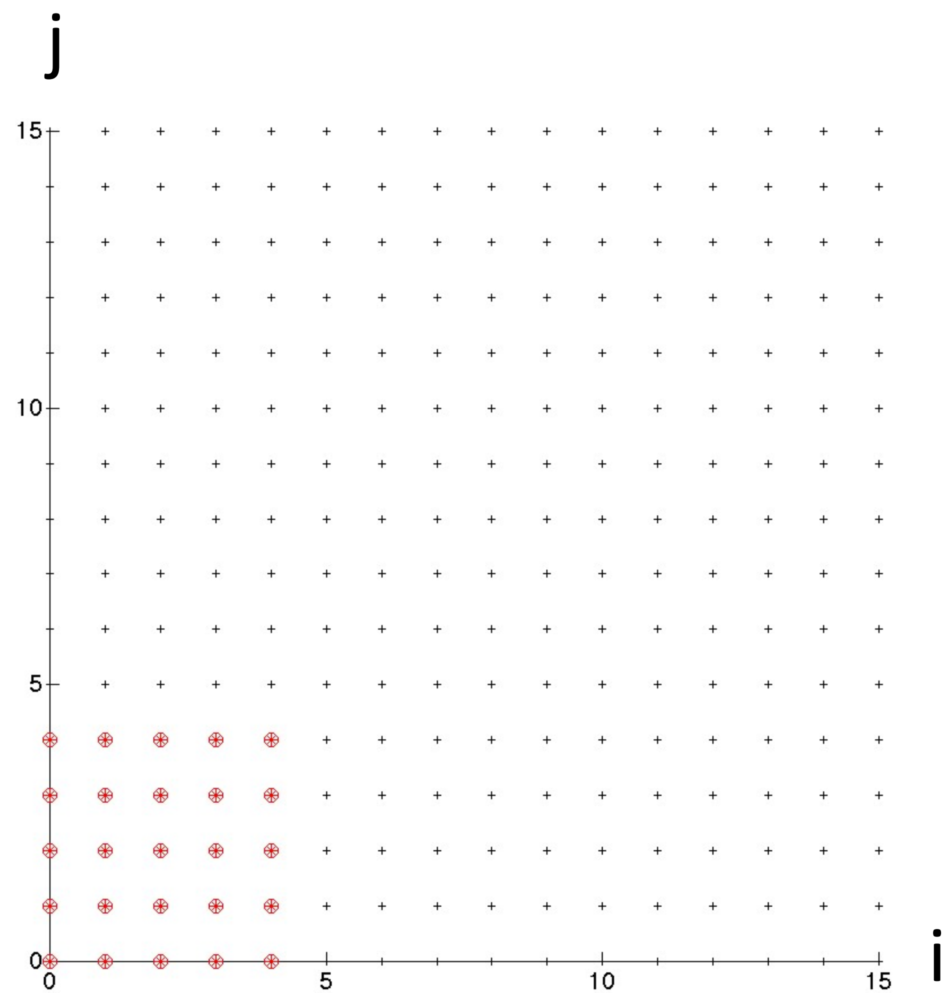
Communication lower bound for n-body (intuition)

- 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

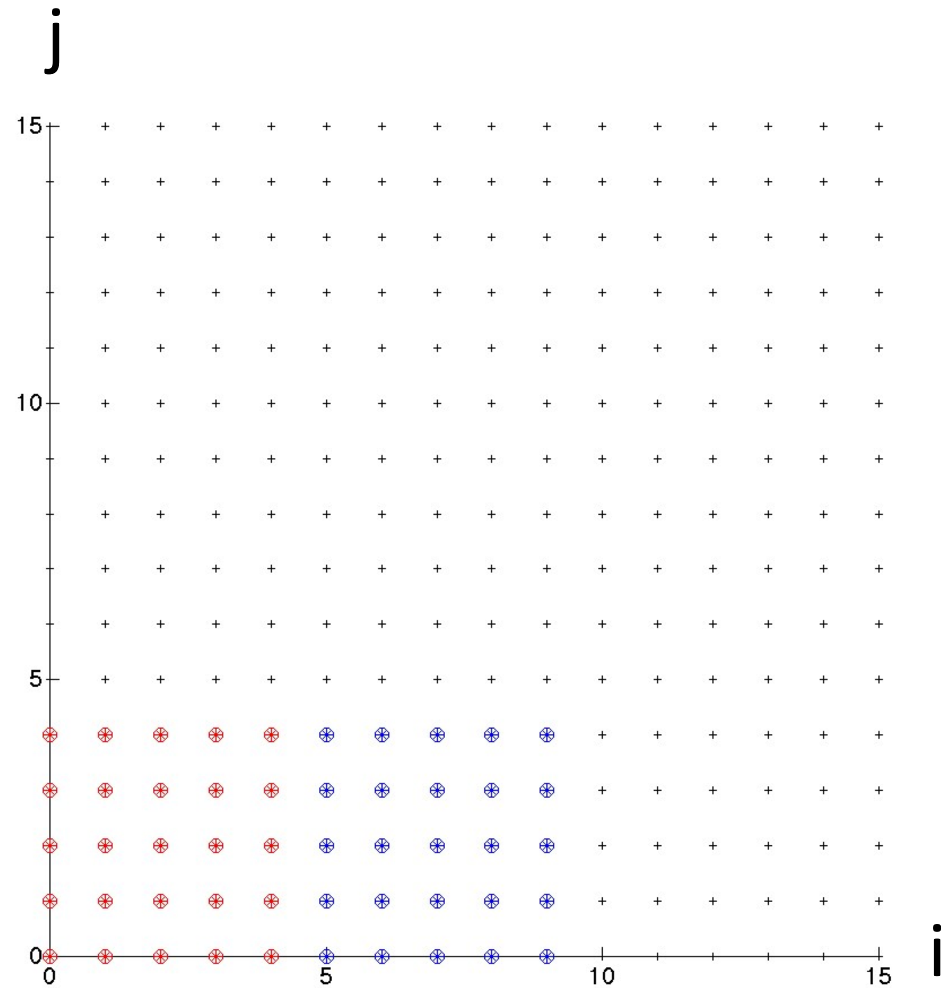
for i = 0:n
  for j = 0:n
    access A(i), B(j)

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



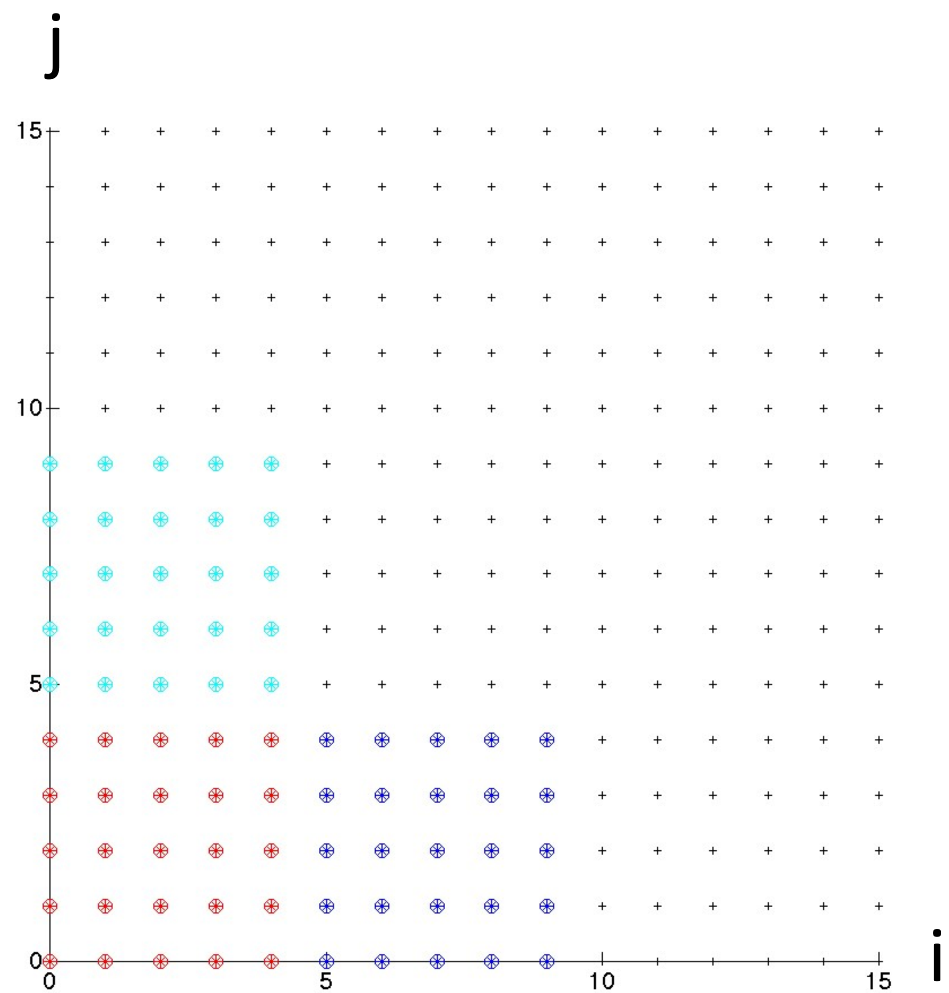
Optimal tiling for usual n-body

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



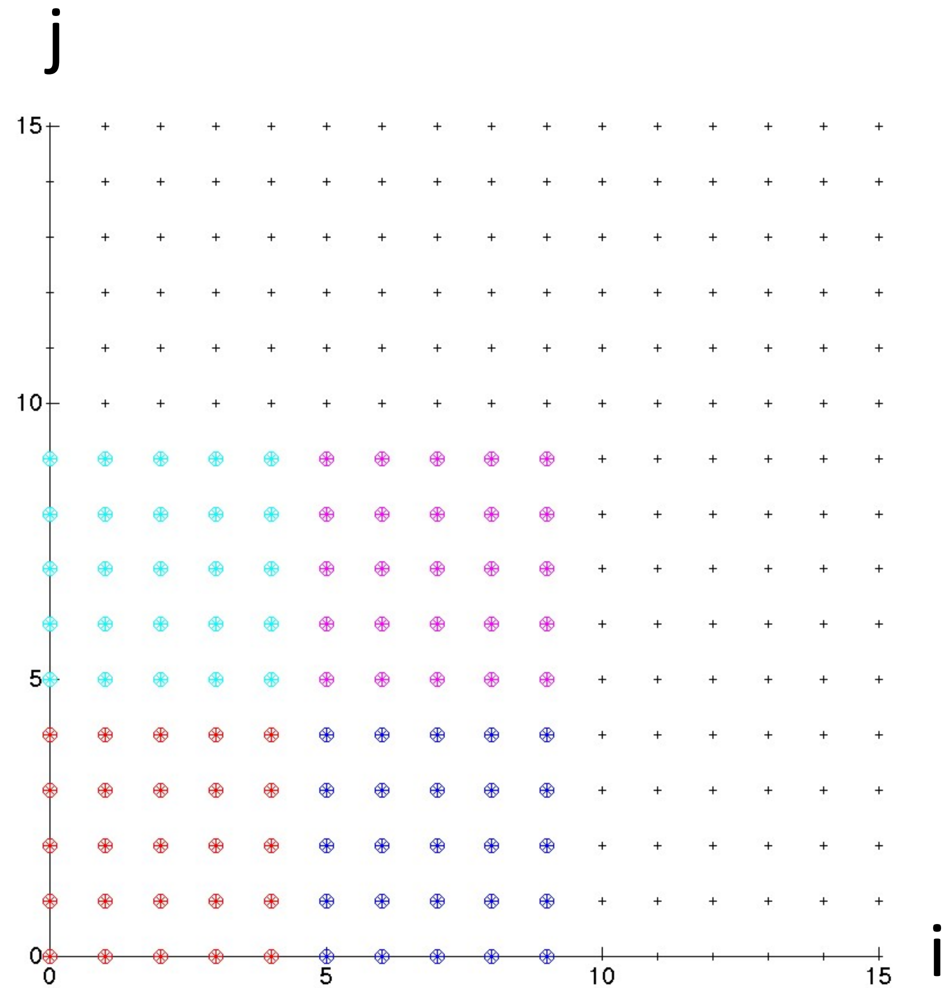
Optimal tiling for usual n-body

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

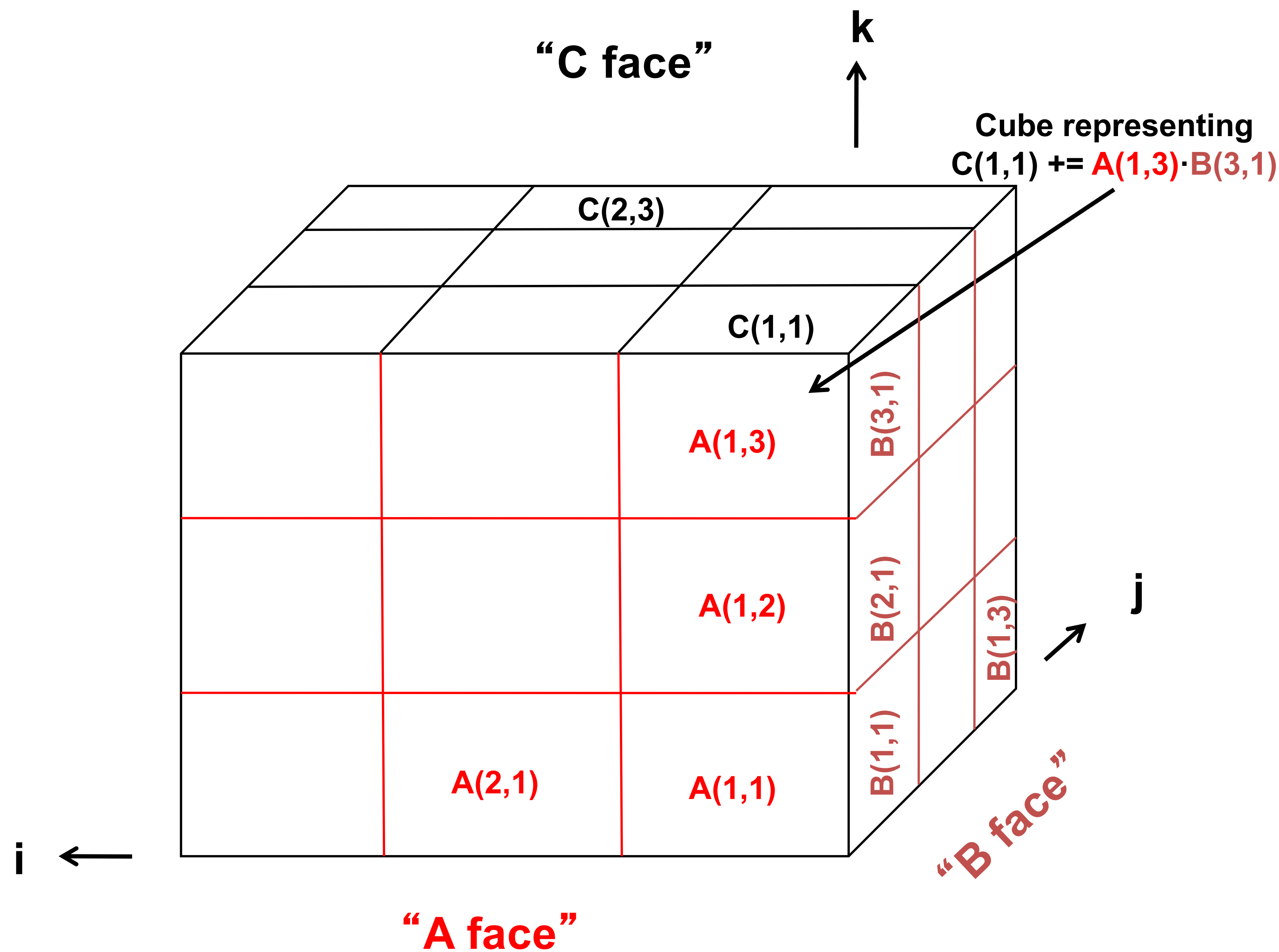
- Many algorithms look like nested loops accessing arrays
  - Linear Algebra (dense and sparse)
  - Grids (structured and unstructured)
  - Convolutional Neural Nets (CNNs) …
- Matmul: C = A*B
  - for i=1:n, for j=1:n, for k=1:n

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

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

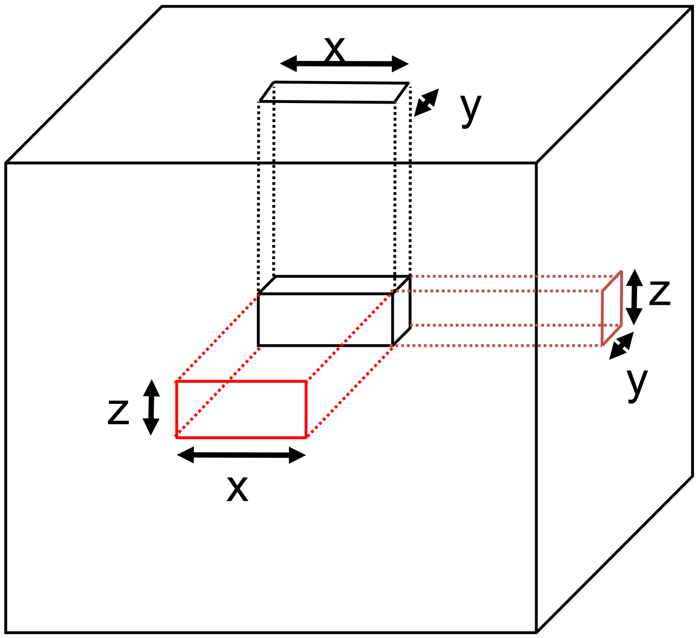
- Analogous to n-body:
  - Only M entries of A, B and C are available in cache
  - Find an upper bound F on the number of different iterations C(i,j) = C(i,j) + A(i,k)*B(k,j) we can perform
  - Need to refill cache n^3/F times to complete algorithm
  - Need to read/write at least M $n^3/$ F words to/from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

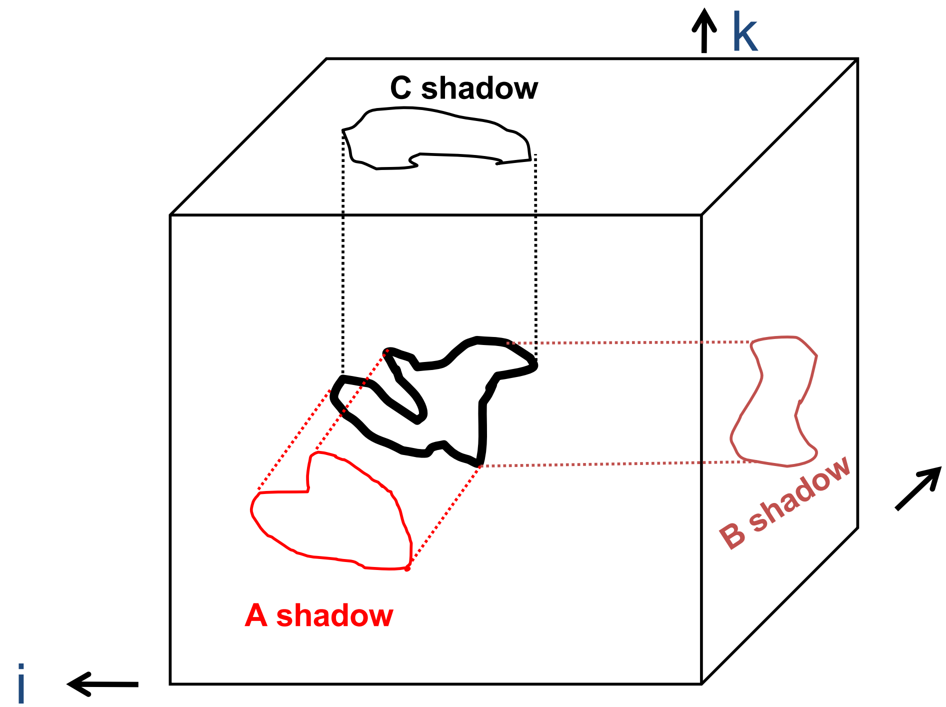
Proof of Communication Lower Bound on C = A·B (2/4)



- 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
= $x \cdot y \cdot z$
= $(xz \cdot zy \cdot yx)^{1/2}$
= (#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) $\cdot$ area(B shadow) $\cdot$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” $\geq M \cdot (n^3 /P) / (M)^{3/2} = \Omega(n^2 / P^{1/2})$

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

  $$\#words\_moved = \Omega\ (\#iterations/M^{sHBL-1})$$

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

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

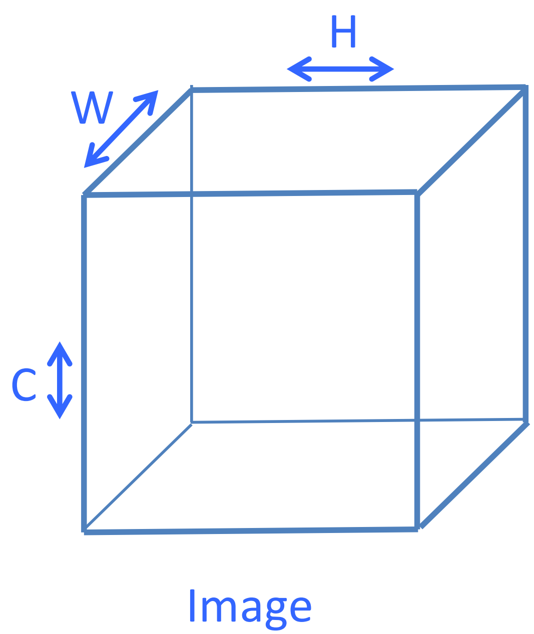
  $\text{rank}(H) \leq \Sigma_j\ e_j * \text{rank}(\phi_j(H))$ for all subgroups $H < Z^k$

- 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, …

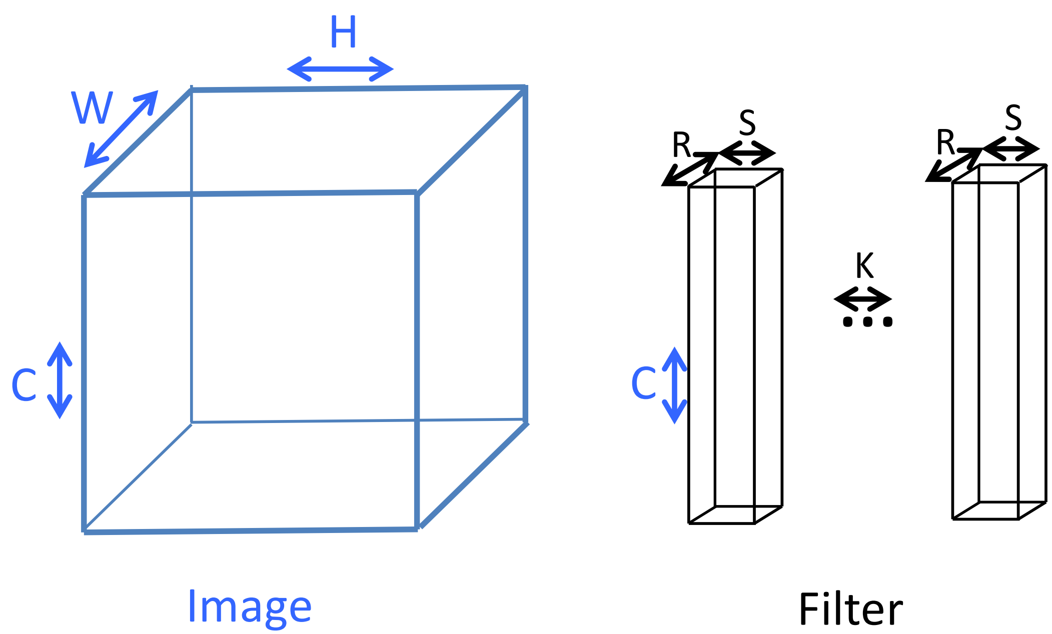
Is this bound attainable?

- 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
    - Recent extension to arbitrary loop bounds, assuming all subscripts "projective" eg (i), (i,j), (i,j,k) etc,

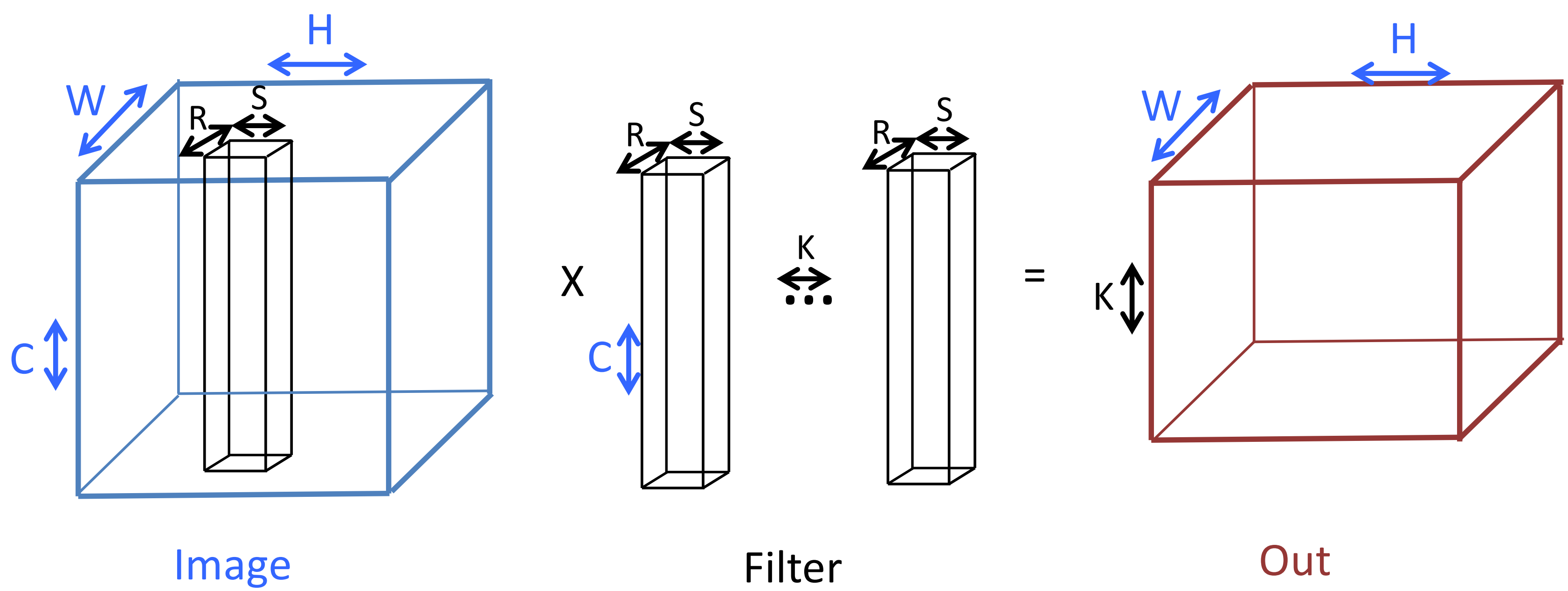
What CNNs compute



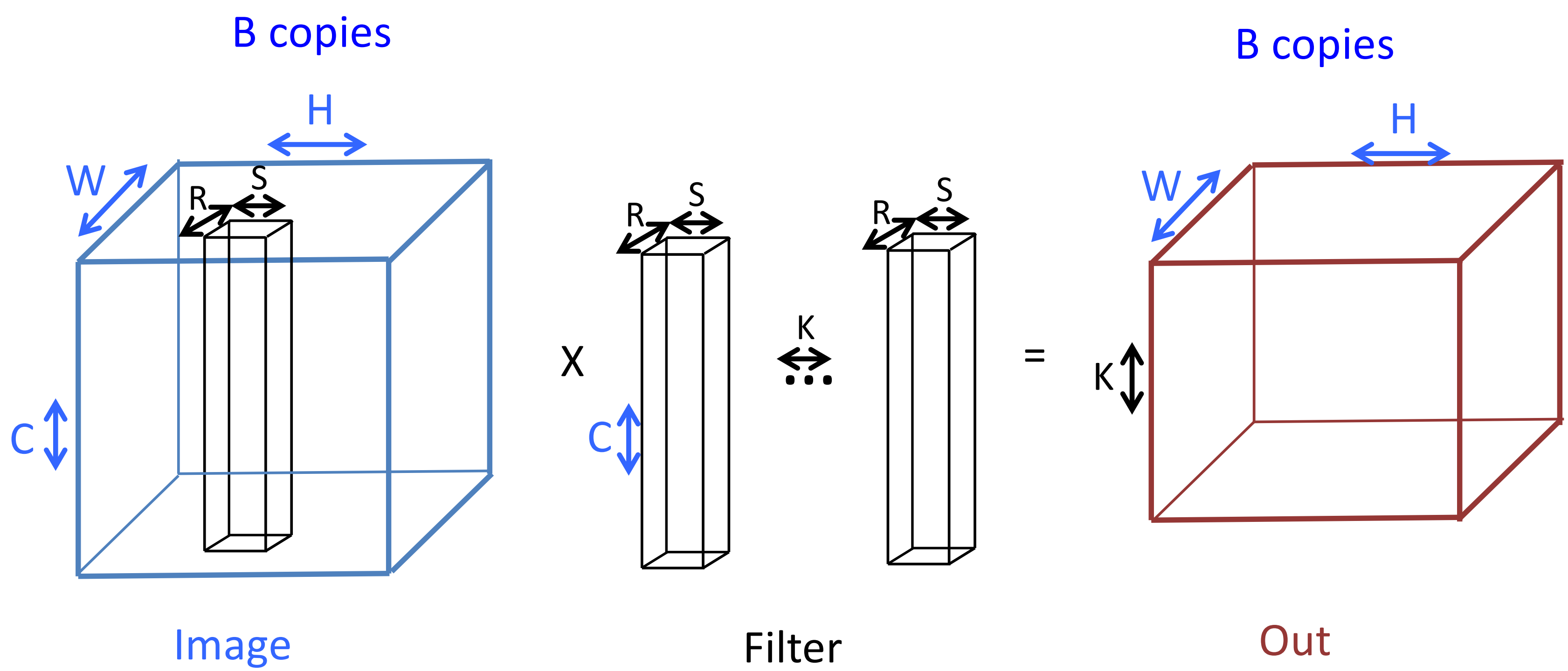
What CNNs compute



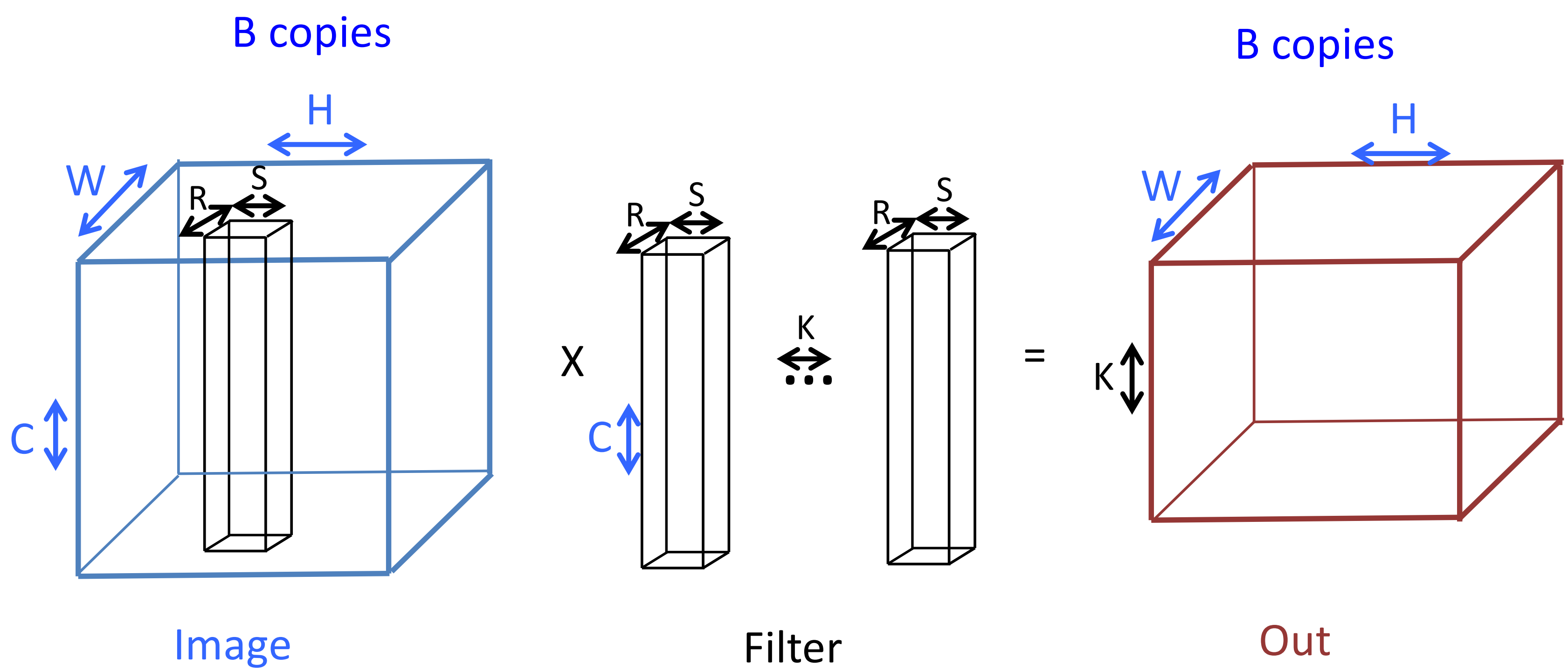
What CNNs compute



What CNNs compute

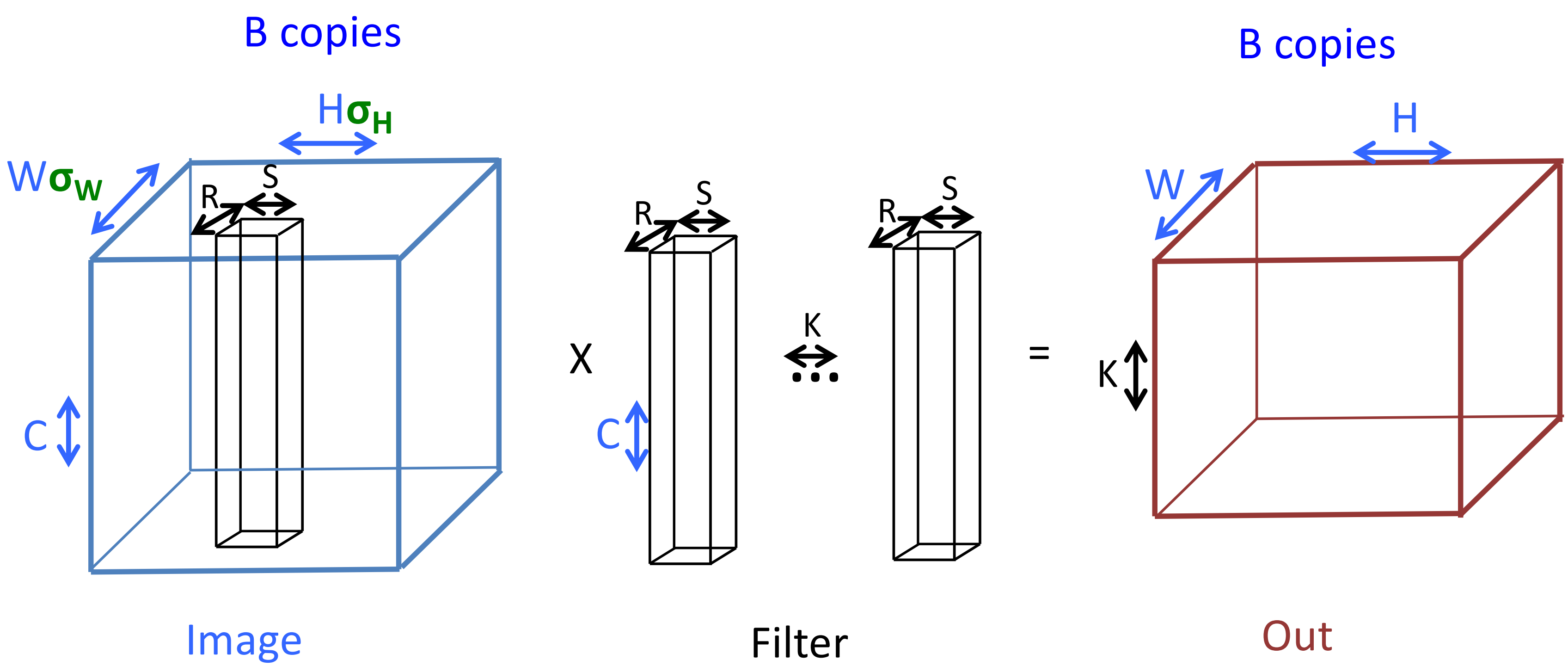


What CNNs compute



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

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

How a CNN is often done – 1D case

- Ex: 1 1x3 filter, 1 1x5 image, shift σ = 1
  - [f1,f2,f3], [im1,im2,im3,im4,im5]
  - 3 dot products of length 3
- Convert image to matrix, do vector*matrix (BLAS2)

$$[\ f1,\ f2,\ f3\ ] \ * \begin{pmatrix} im1 & im2 & im3 \\ Im2 & im3 & im4 \\ im3 & im4 & im5 \end{pmatrix}$$

- Multiple filters -> matrix*matrix (BLAS3)

How a CNN is often done – 2D case

- Same idea:
  - Convert each 2D image to a matrix: im2col (Matlab)
  - Convert each 2D filter into a row vector, stack them
  - Do matrix-matrix multiply
- Ex: 2x2 filter => 1x4 vector
  - [f11,f21,f12,f22]
- Ex: 5x5 image => 4x20 matrix (1 col per conv.)

$$\begin{pmatrix} im11 & im21 & im31 & im41 & im12 & \ldots & im44 \\ im21 & im31 & im41 & im51 & im22 & \ldots & im54 \\ im12 & im22 & im32 & im42 & im13 & \ldots & im45 \\ im22 & im32 & im42 & im52 & im23 & \ldots & im55 \end{pmatrix}$$

CNN using Im2col

- Same operations as 7 nested loops
- Can exploit optimized matmul
- Need to replicate data
- Can we communicate less, by doing convolutions directly?
  - Ex: Intel MKL-DNN, some NVIDIA libraries

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

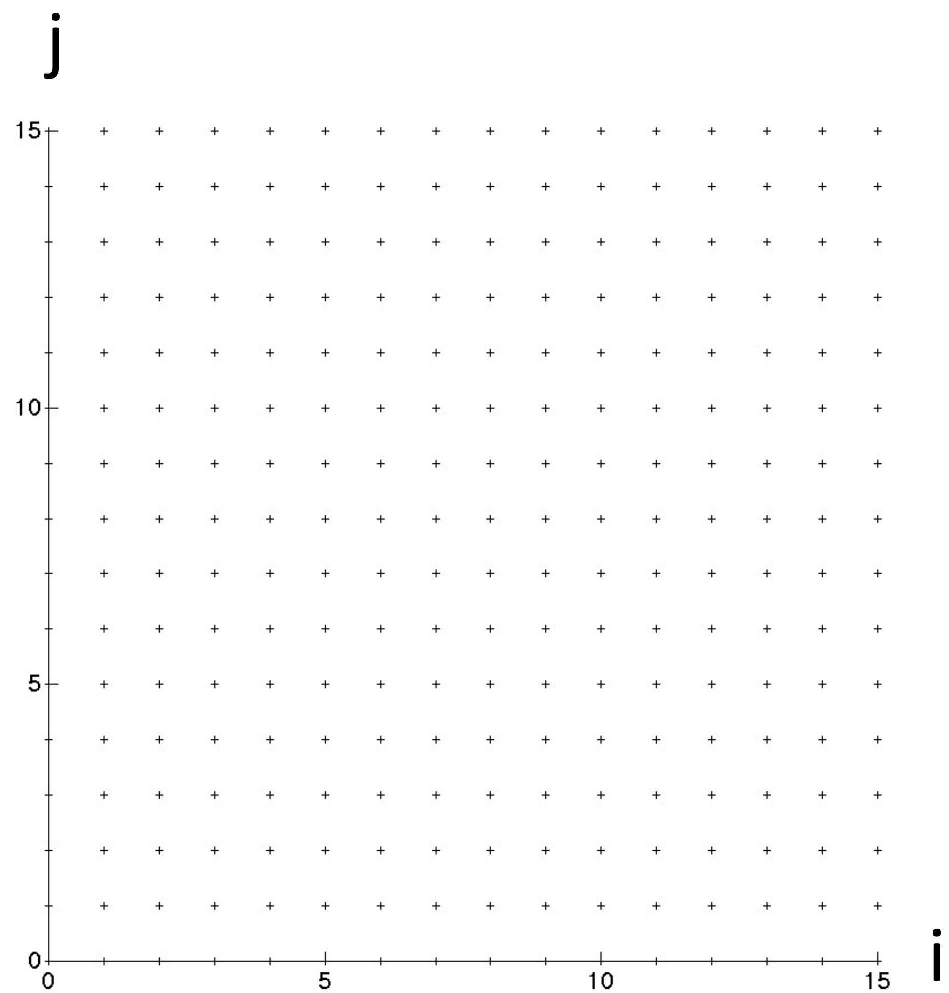
  CKRS,   ...   size of Filter

  N/M,   ...   same lower bound as 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, solved using Mathematica)

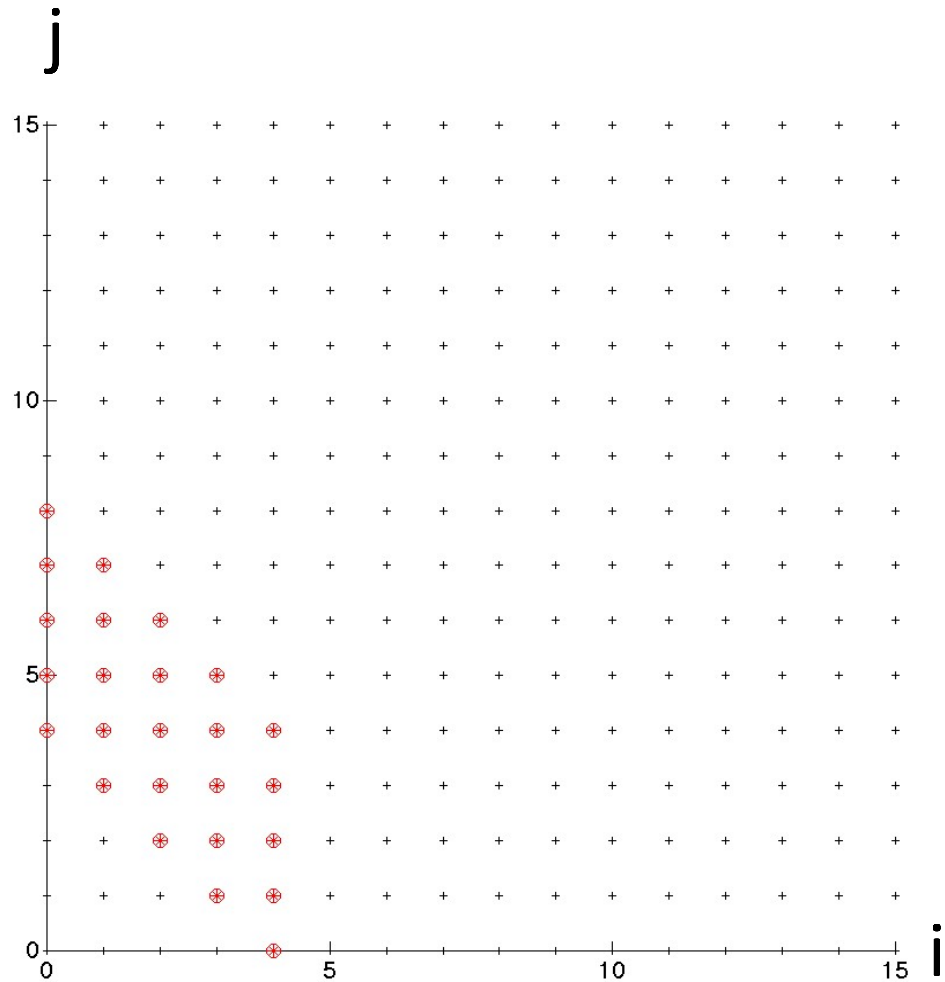
Optimal tiling for “slanted” n-body

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



Optimal tiling for "slanted" n-body

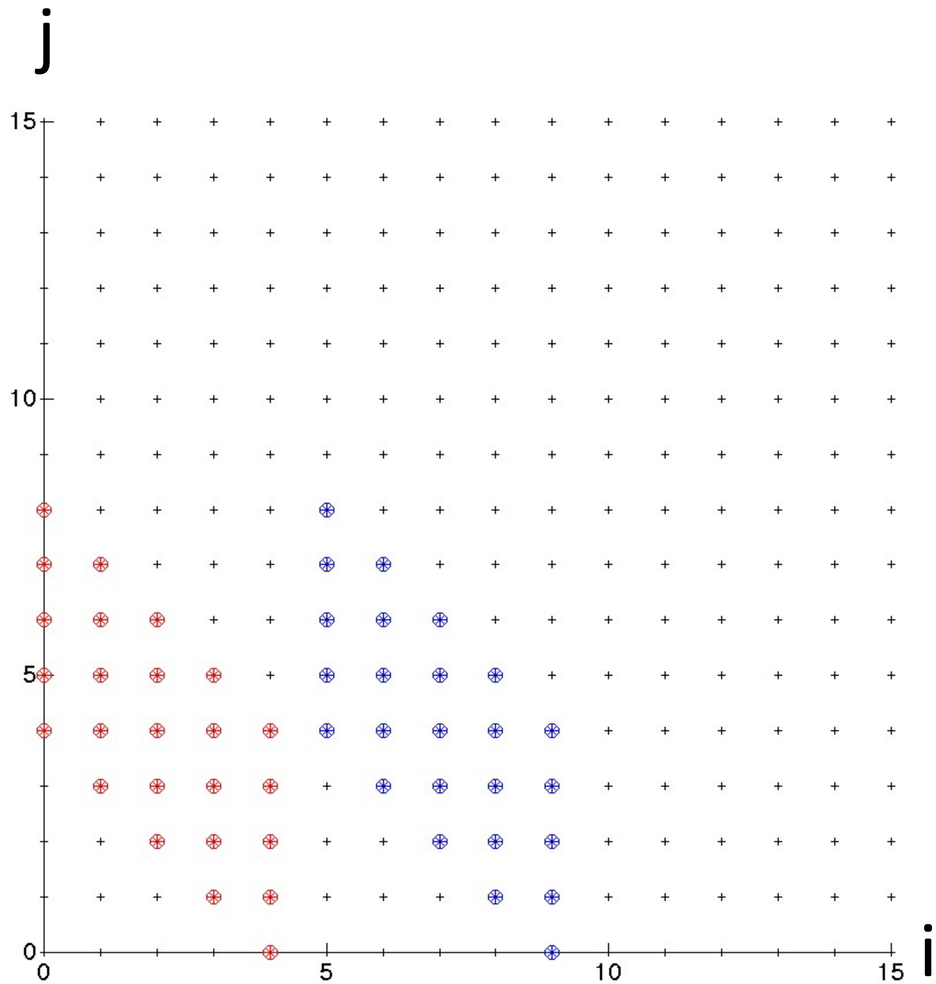
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)



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

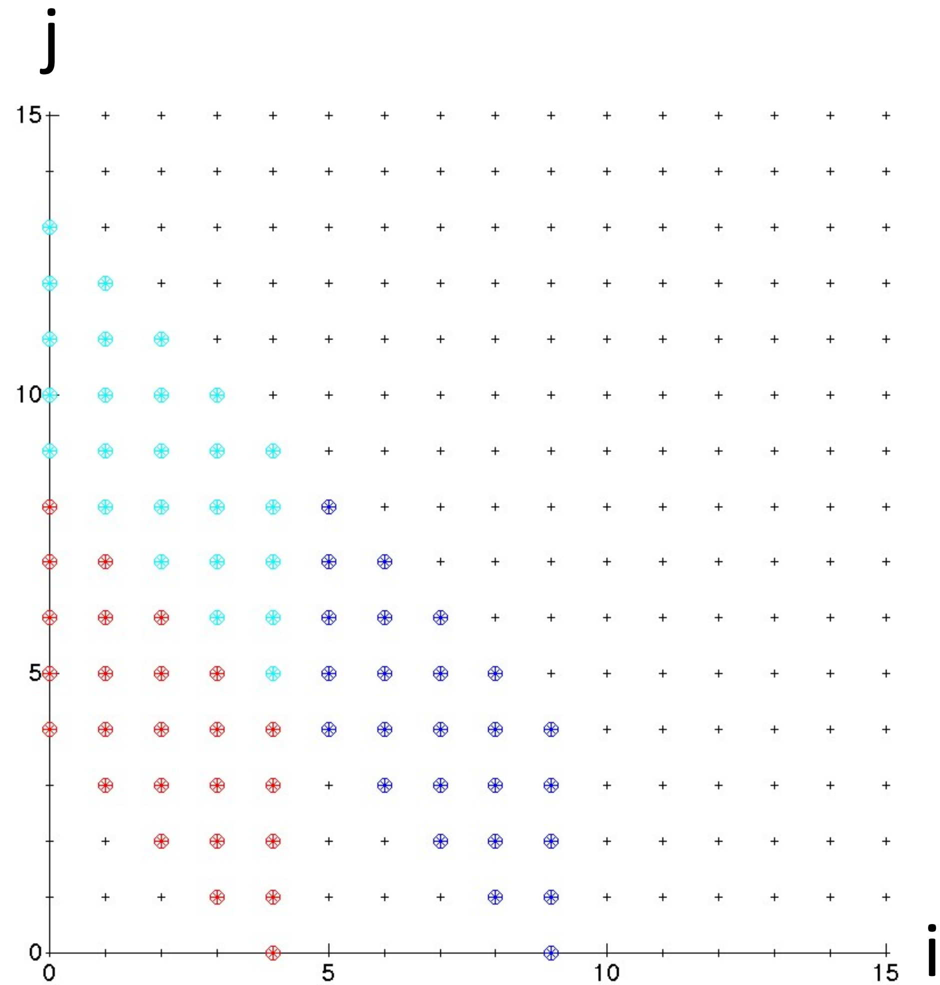
Optimal tiling for “slanted” n-body

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



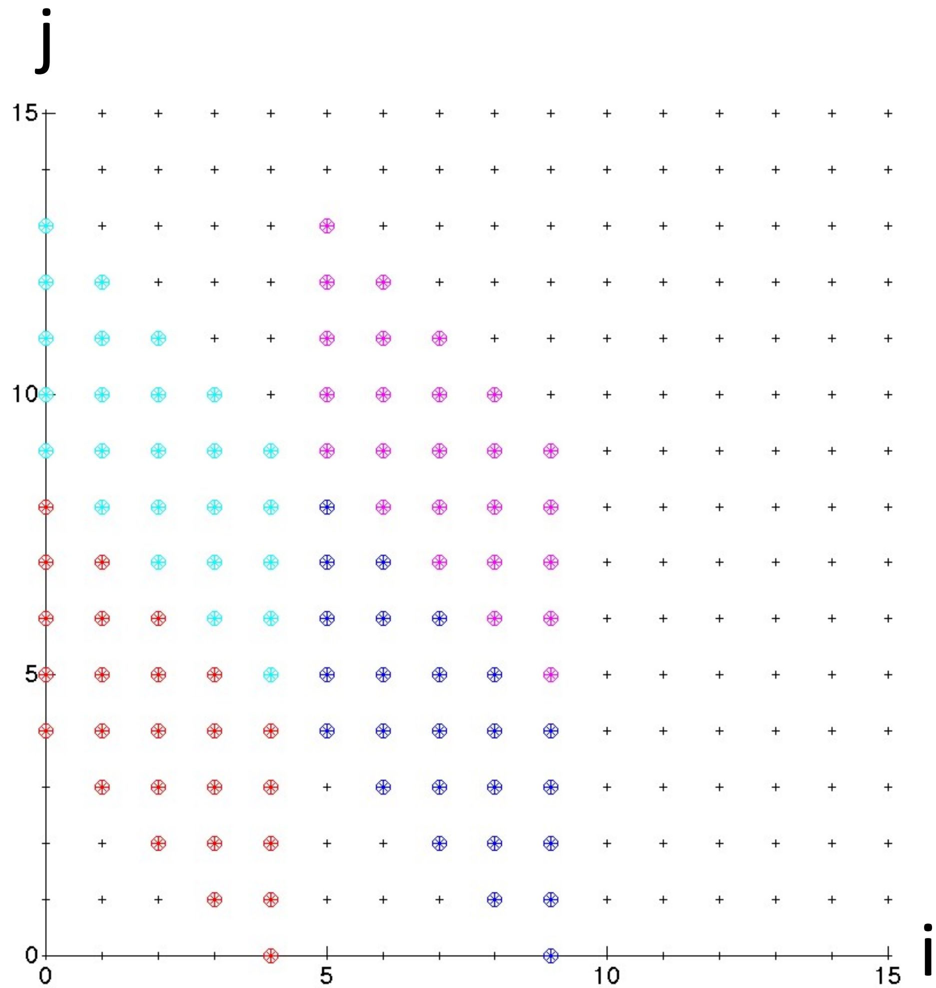
Optimal tiling for “slanted” n-body

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



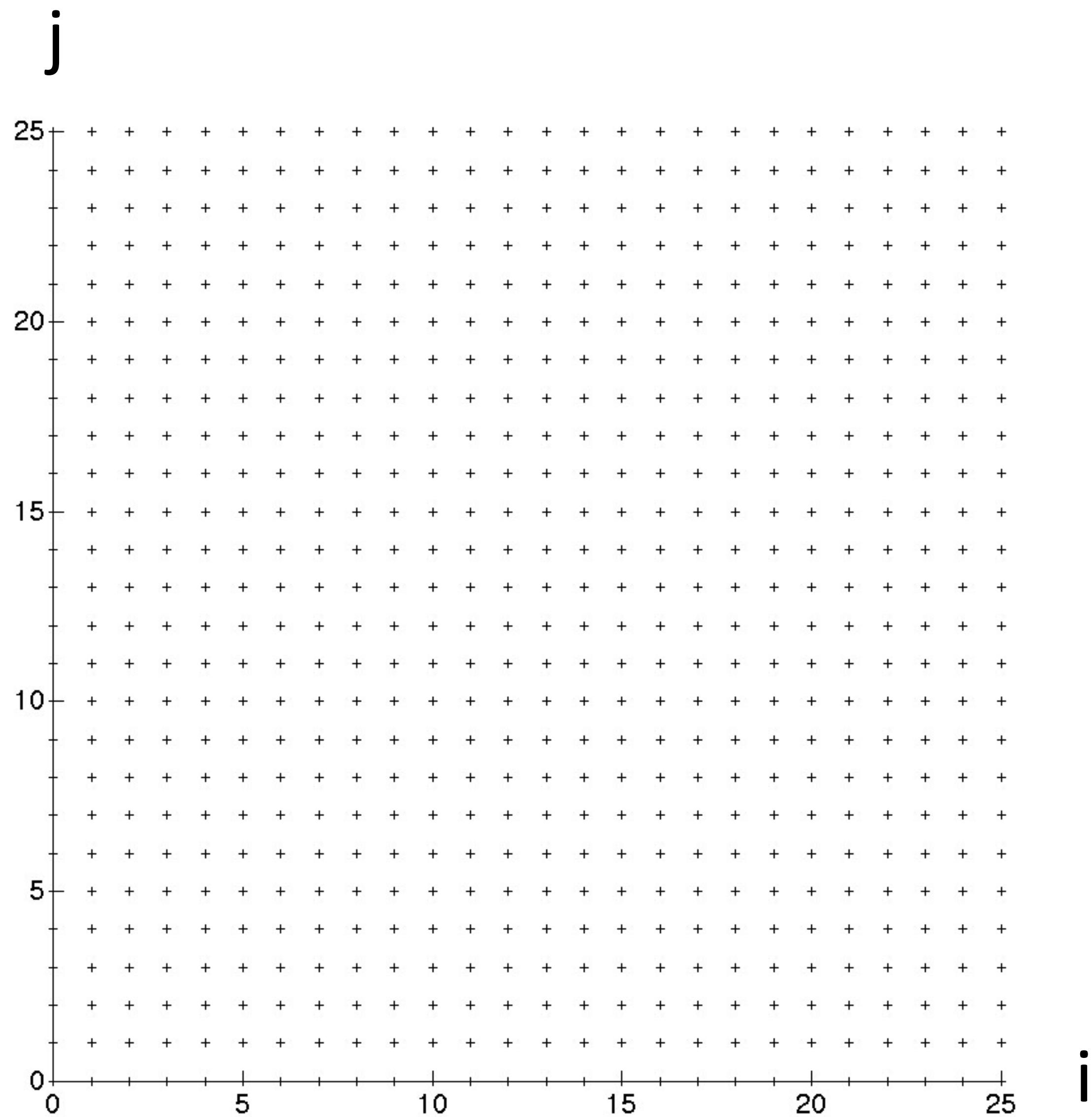
Optimal tiling for “slanted” n-body

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



Optimal tiling for "twisted" n-body

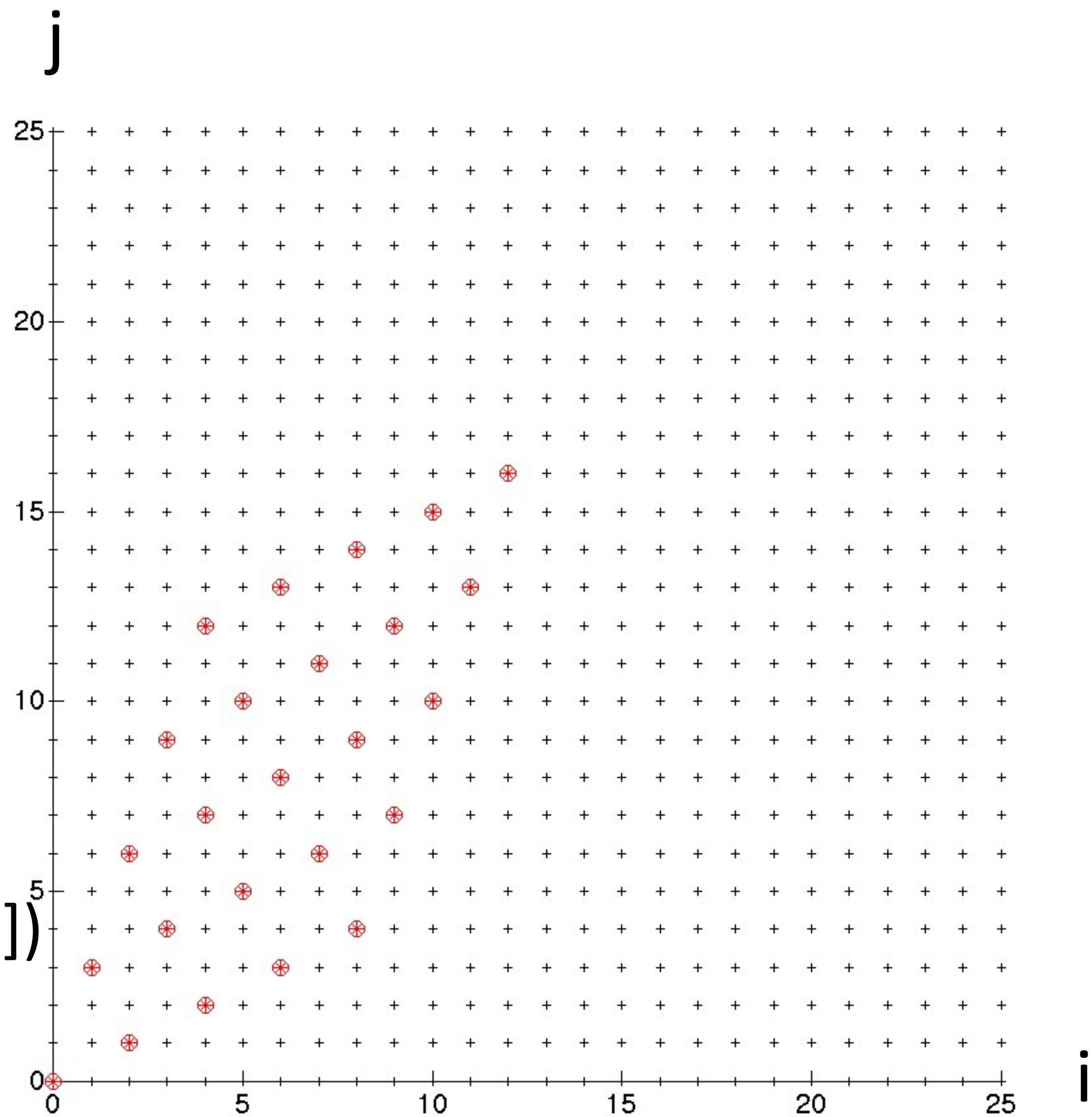
```
for i = 0:n
   for j = 0:n
      access A(3*i-j),
               B(i-2*j)
```



Optimal tiling for "twisted" n-body

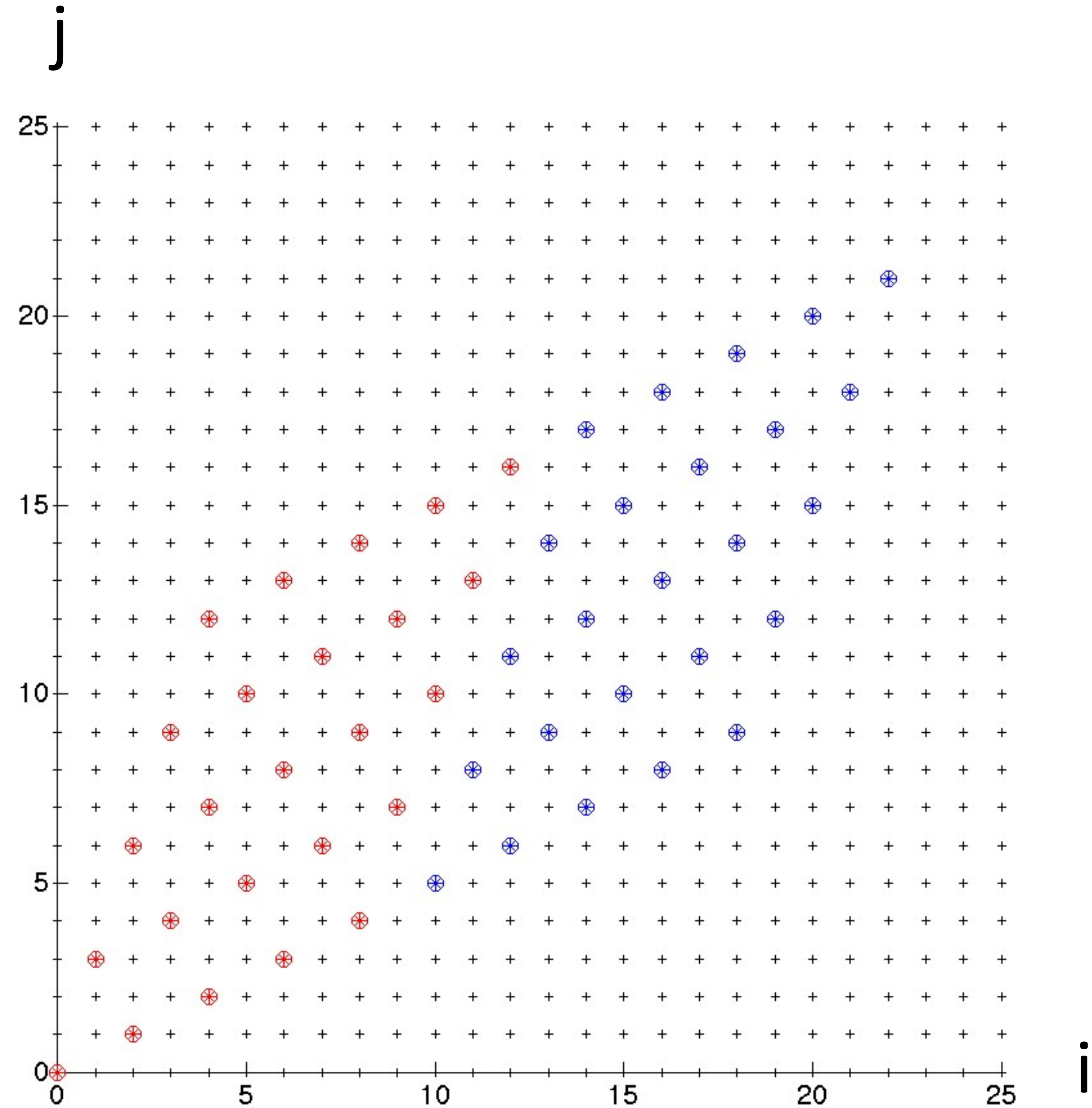
for i = 0:n
  for j = 0:n
    access A(3*i-j),
         B(i-2*j)

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



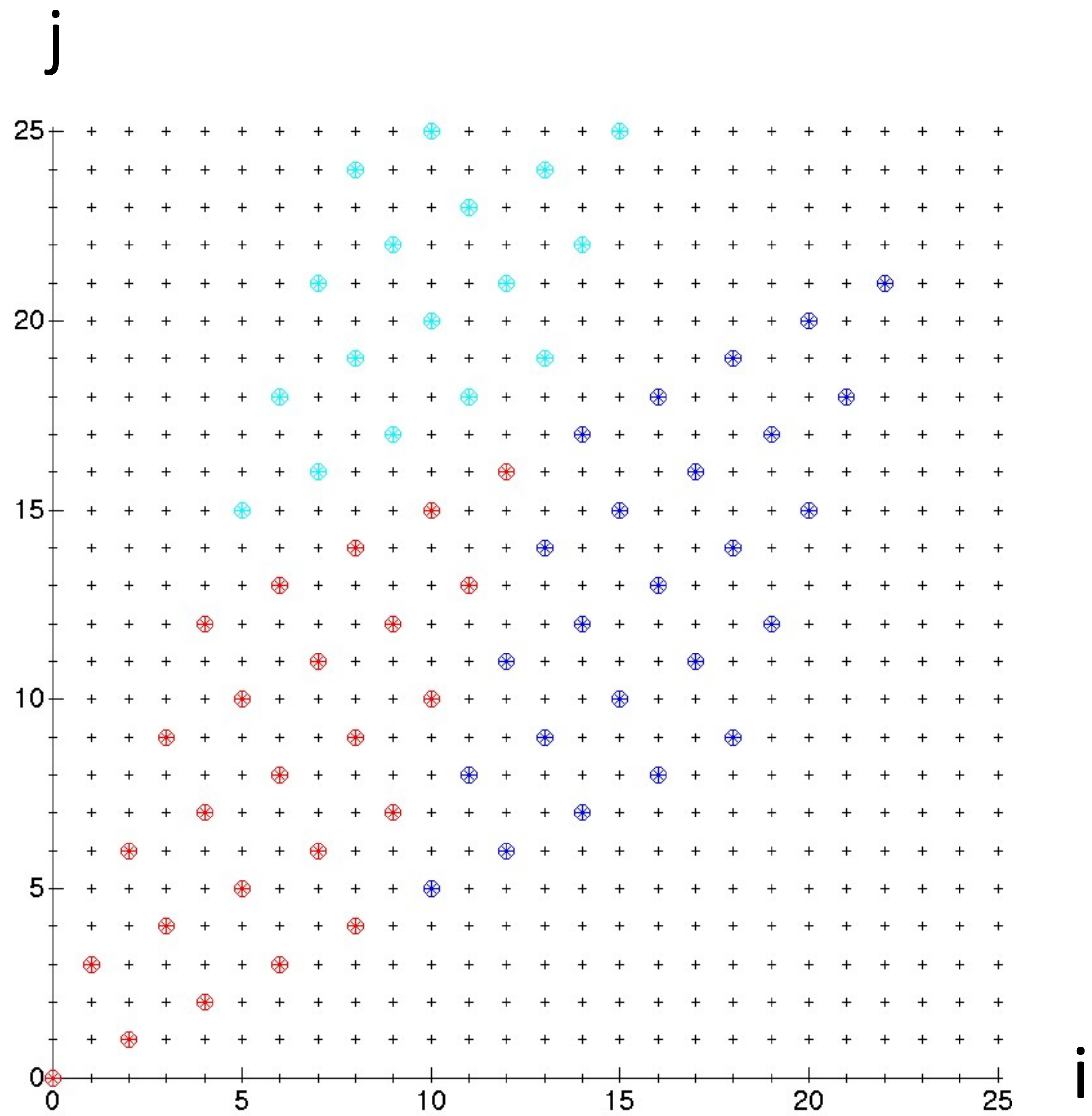
Optimal tiling for "twisted" n-body

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



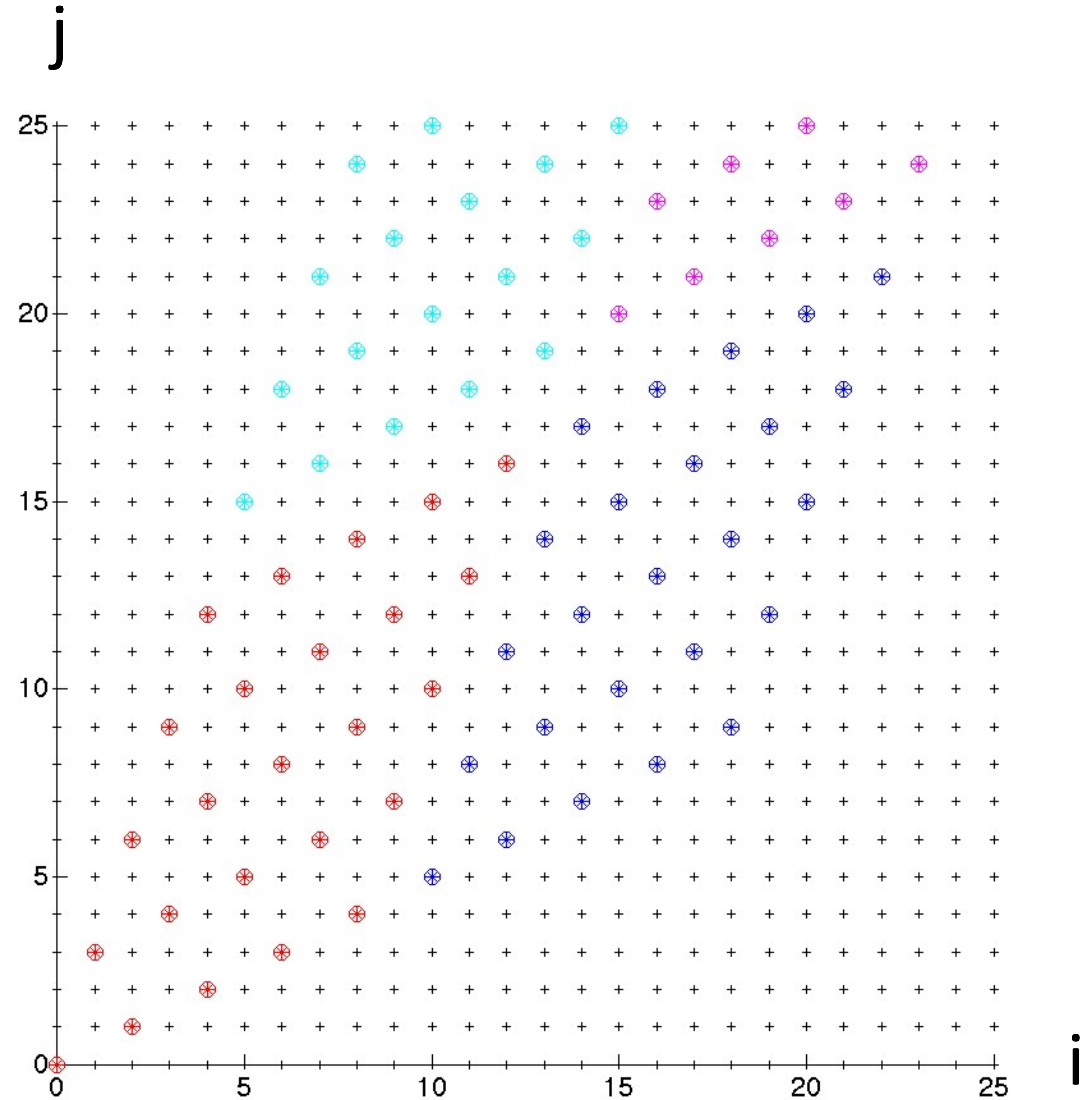
Optimal tiling for "twisted" n-body

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



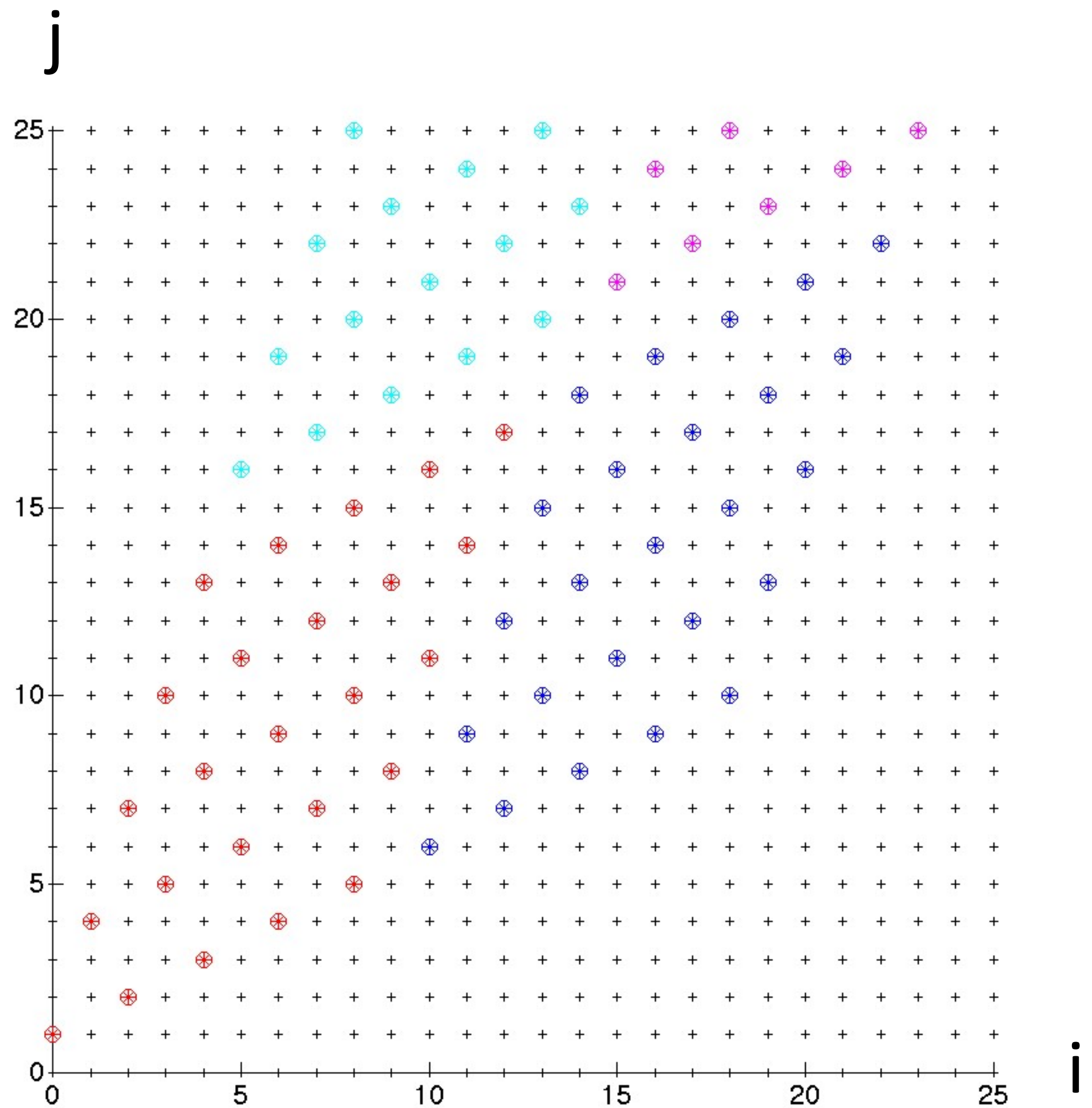
Optimal tiling for “twisted” n-body

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



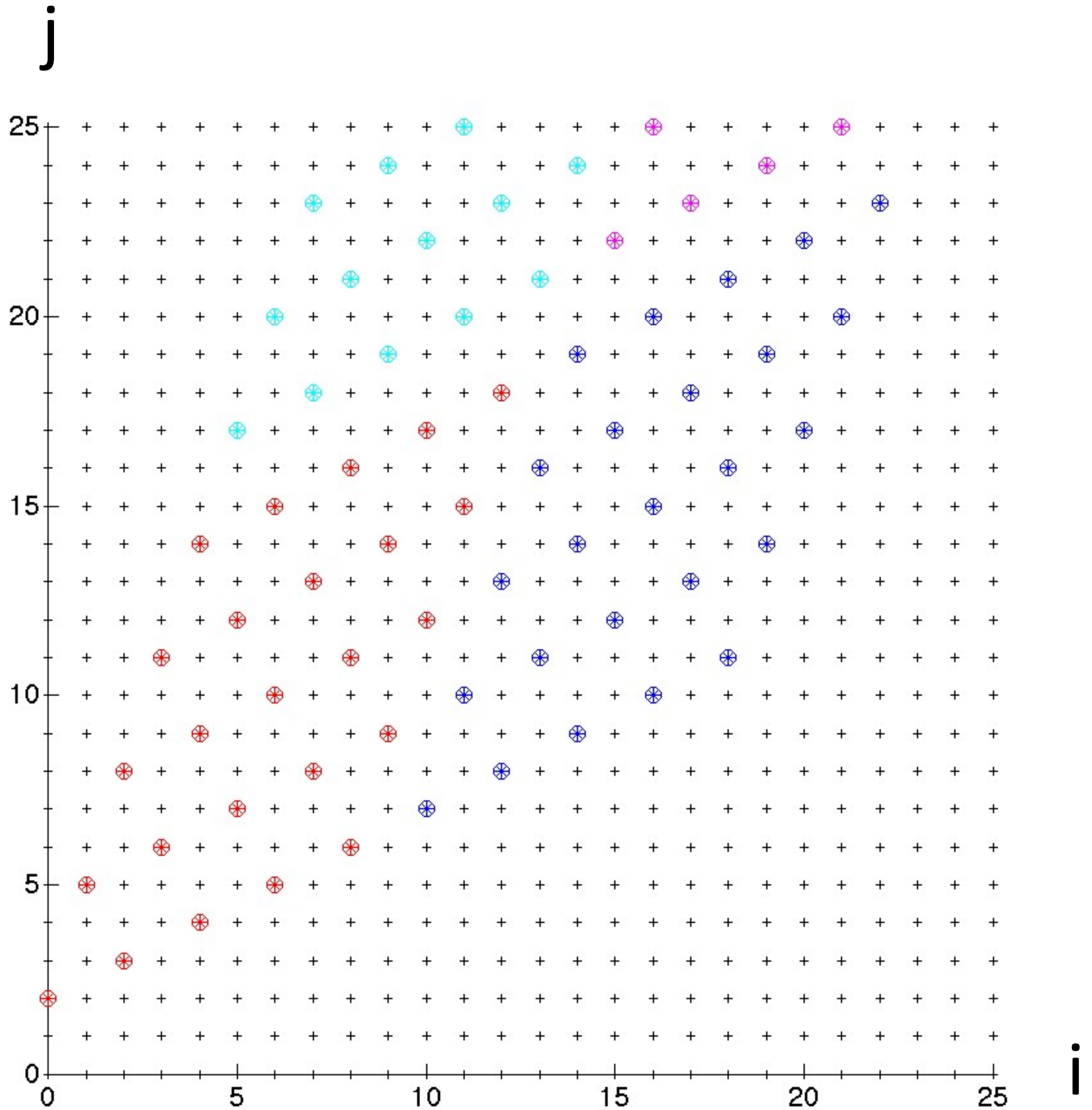
Optimal tiling for “twisted” n-body

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



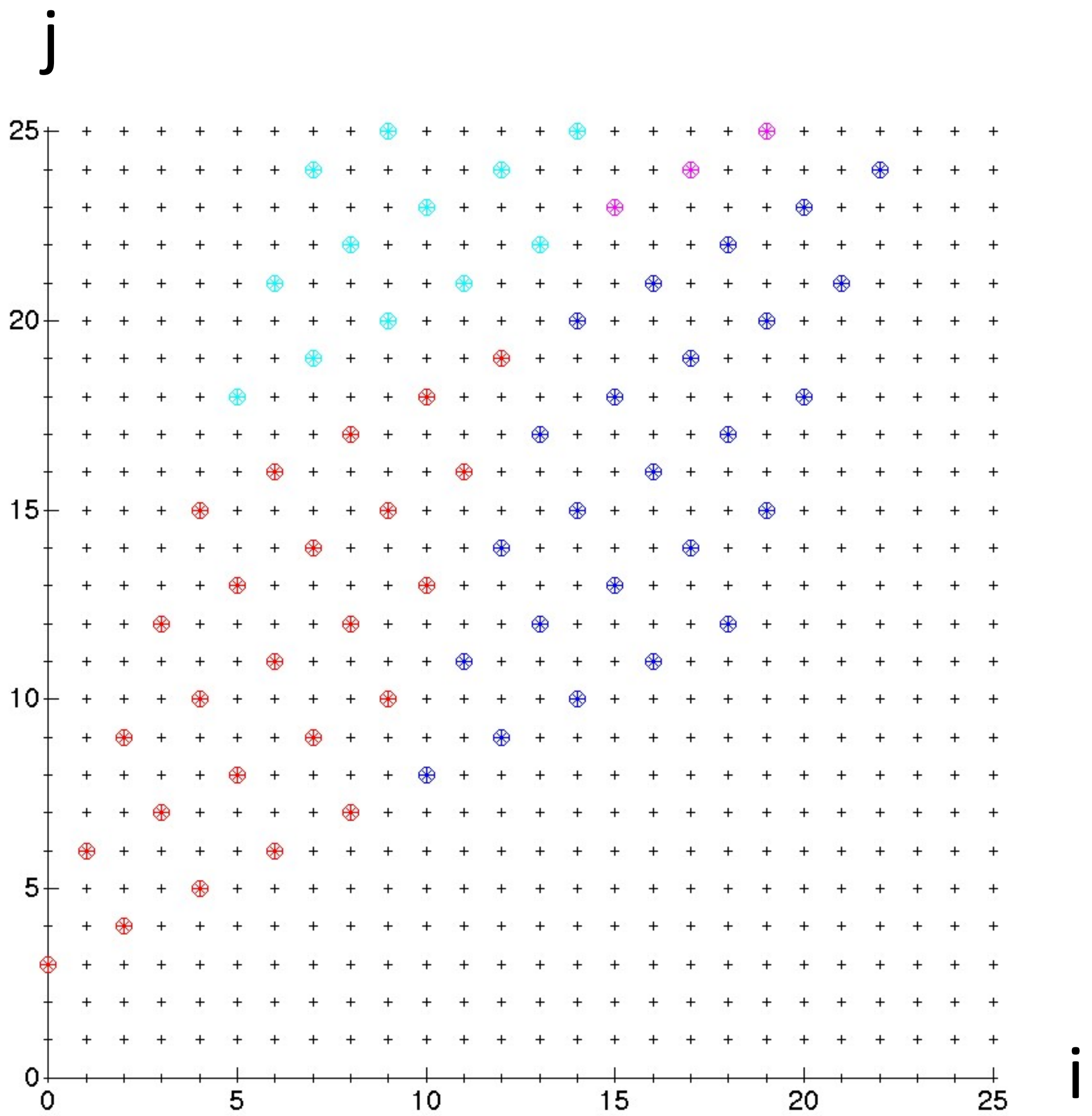
Optimal tiling for "twisted" n-body

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



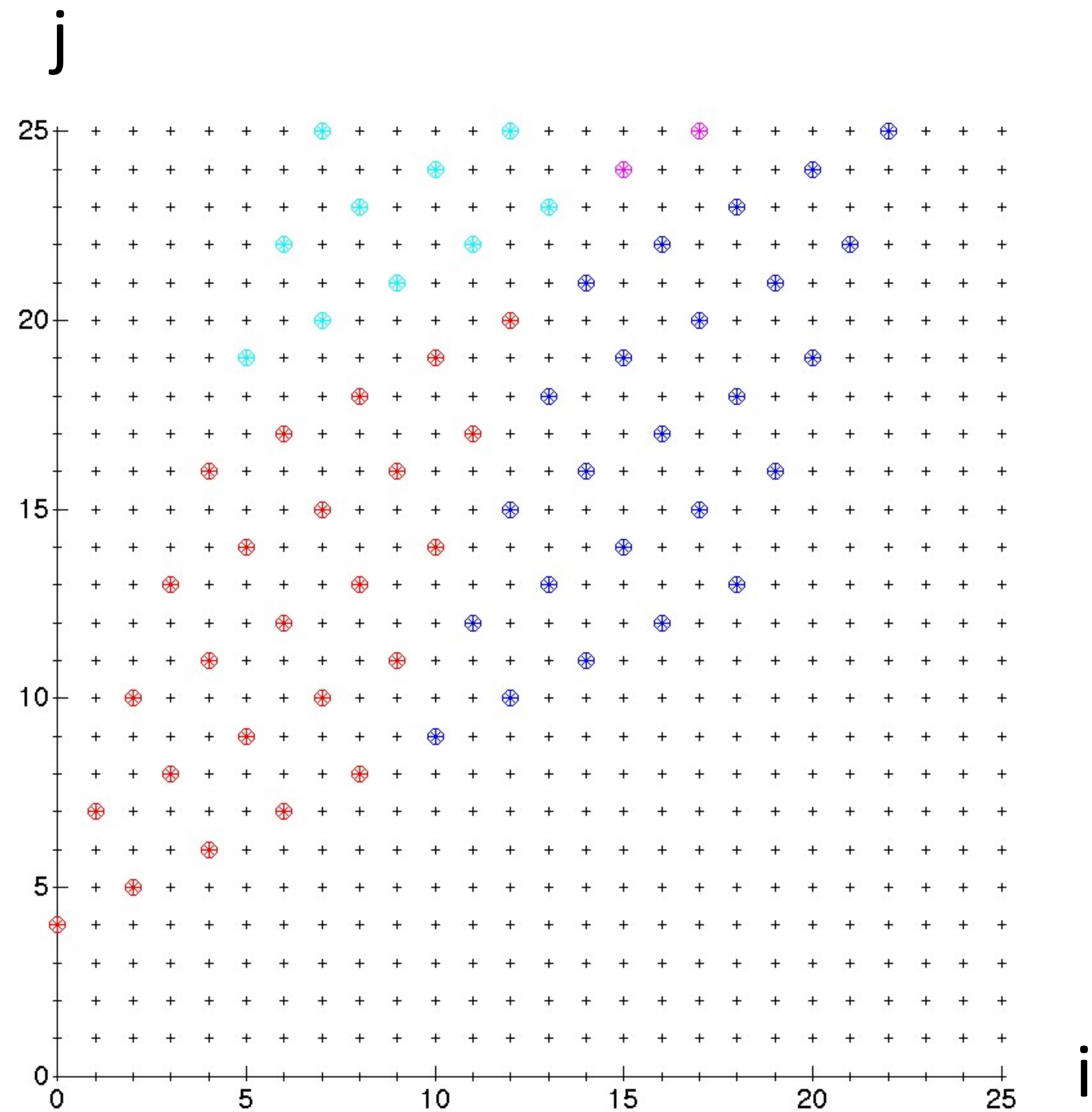
Optimal tiling for "twisted" n-body

```
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)
```