Communication avoiding algorithms for LU and QR factorizations

Laura Grigori

Alpines

INRIA Paris - LJLL, Sorbonne Université

October 2021

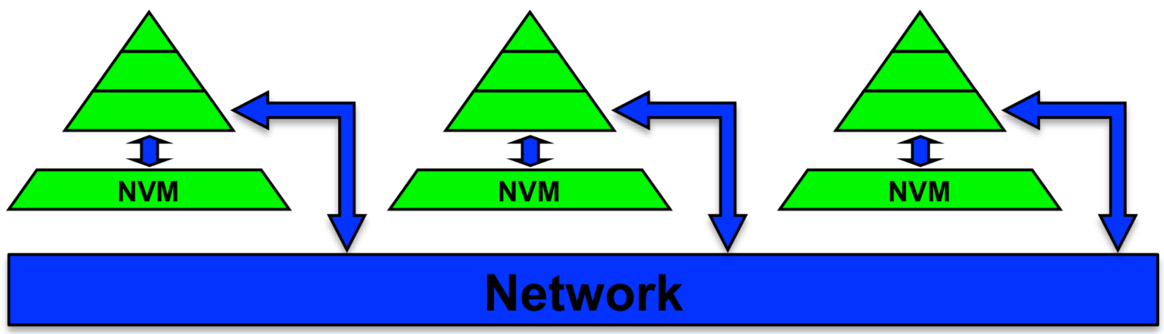
Plan

- Motivation
- Communication complexity of linear algebra operations
- Communication avoiding for dense linear algebra
  - LU, QR, Rank Revealing QR factorizations
  - Progressively implemented in ScaLAPACK, LAPACK
  - Algorithms for multicore processors
- Conclusions

Motivation - the communication wall

- Runtime of an algorithm is the sum of:
  - #flops x **time_per_flop**
  - #words_moved / **bandwidth**
  - #messages x **latency**

- Time to move data >> time per flop
  - Gap steadily and exponentially growing over time

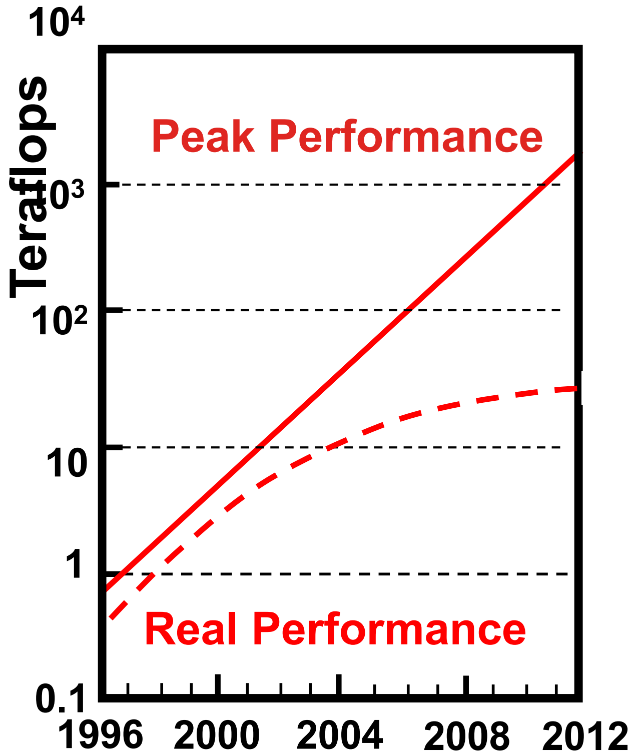


Motivation - the communication wall

- Runtime of an algorithm is the sum of:
  - #flops x **time_per_flop**
  - #words_moved / **bandwidth**
  - #messages x **latency**

- Time to move data >> time per flop
  - Gap steadily and exponentially growing over time

Annual improvements			
Time/flop		Bandwidth	Latency
59%	Network	**26%**	**15%**
	DRAM	**23%**	**5%**



Adapted from J. Demmel

- Performance of an application is less than 10% of the peak performance

*"We are going to hit the **memory wall**, unless something basic changes"*
[W. Wulf, S. McKee, 95]

Compelling numbers

DRAM latency:

- DDR2 (2007) ~ 120 ns 1x
- DDR4 (2014) ~ 45 ns 2.6x in 7 yrs
- Stacked memory ~ similar to DDR4

Time/flop

- 2008 Intel Nehalem 3.2GHz × 4 cores (51.2 GFlops/socket) 1x
- 2017 Intel Skylake XP 2.1GHz × 28 cores (1.8 TFlops/socket) 35x in 9 yrs

Network latency

- Interconnect (example one machine today): 0.25μs to 3.7μs MPI latency

Source: G. Bosilca (UTK), S. Knepper (Intel), J. Shalf (LBL)

Selected past work on reducing communication

- Only few examples shown, many references available

A. Tuning

- Overlap communication and computation, at most a factor of 2 speedup

B. Ghosting

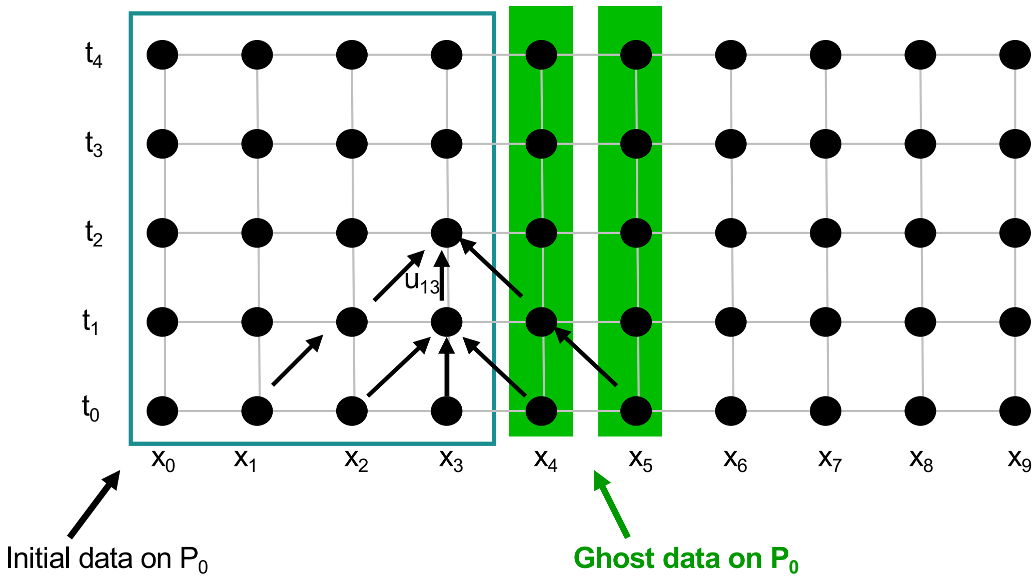
- Standard approach in *explicit methods*
- Store redundantly data from neighboring processors for future computations

Example of a parabolic PDE

$$u_t = \alpha \Delta u$$

with a finite difference,
the solution at a grid point is:

$$u_{i,j+1} = u(x_i, t_{j+1})$$
$$= f(u_{i-1,j}, u_{ij}, u_{i+1,j})$$



Selected past work on reducing communication

C. Same operation, different schedule of the computation

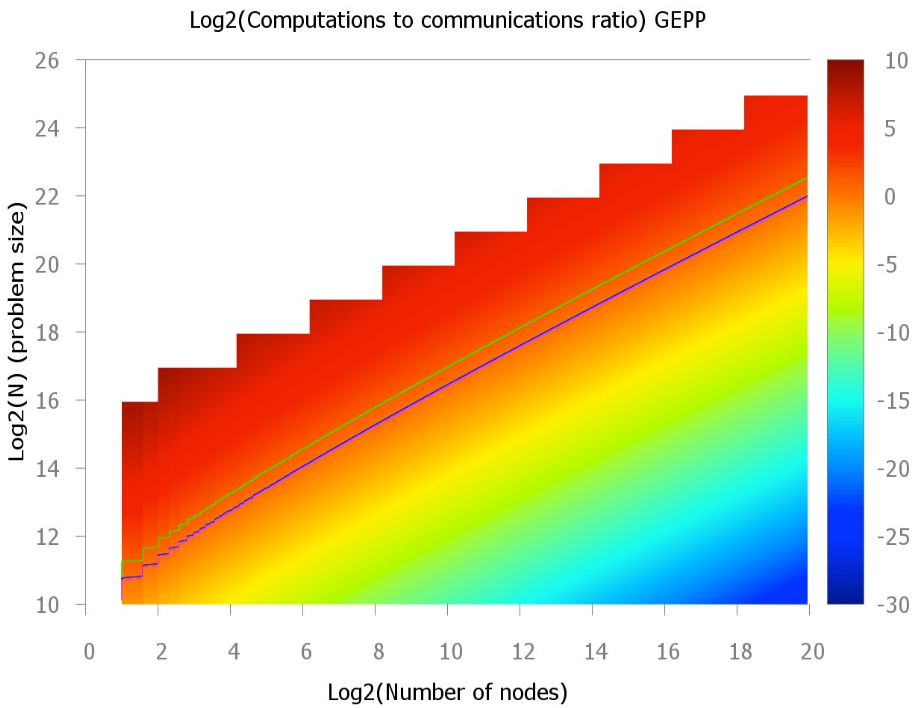
Block algorithms for dense linear algebra

- Barron and Swinnerton-Dyer, 1960
  - LU factorization used to solve a system with 31 equations - first subroutine written for EDSAC 2
  - Block LU factorization used to solve a system with 100 equations using an auxiliary magnetic-tape
  - The basis of the algorithm used in LAPACK

Cache oblivious algorithms

- recursive Cholesky, LU, QR

(Gustavson '97, Toledo '97,
Elmroth and Gustavson '98,
Frens and Wise '03,
Ahmed and Pingali '00)



Selected past work on reducing communication

D. Same algebraic framework, different numerical algorithm

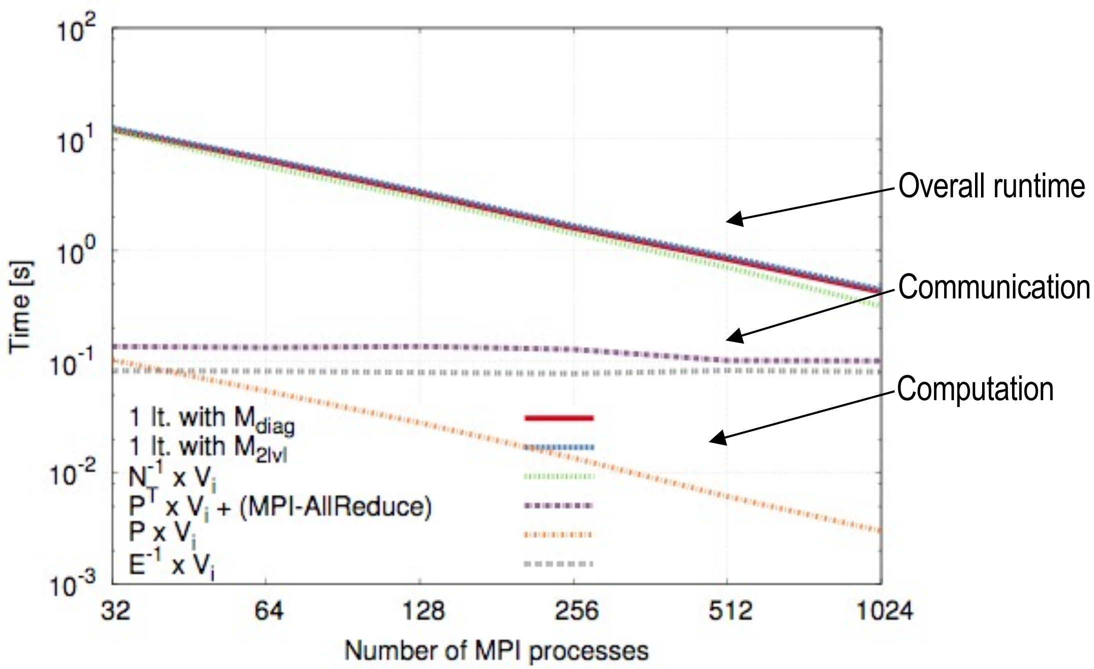
More opportunities for reducing communication, may affect stability

Dense LU-like factorization (Barron and Swinnerton-Dyer, 60)

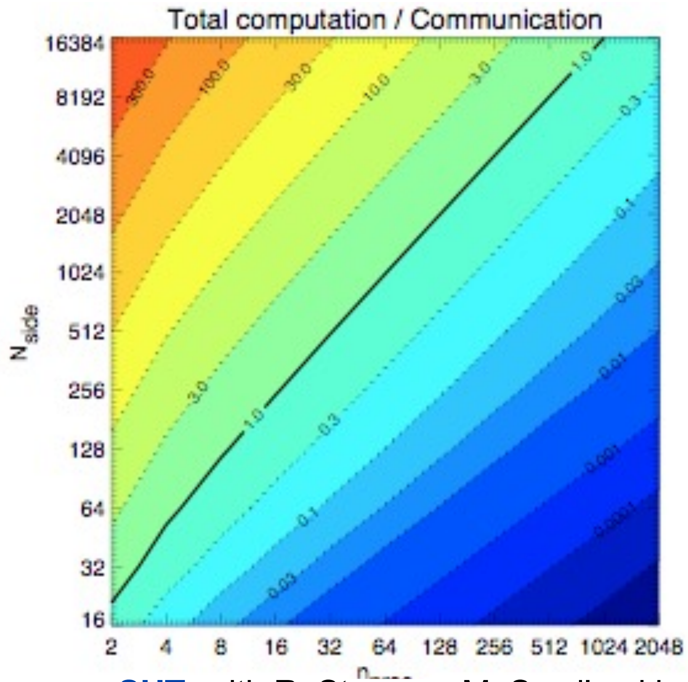
- LU-like factorization based on pairwise pivoting and its block version
  $PA = L_1 L_2 \dots L_n U$
- With small modifications, minimizes communication between two levels of fast-slow memory
- Stable for small matrices, unstable for nowadays matrices

Communication in CMB data analysis

- Map-making problem
  - Find the best map *x* from observations *d*, scanning strategy *A*, and noise N^{-1}
  - Solve generalized least squares problem involving sparse matrices of size 10^{12}-by-10^{7}
- Spherical harmonic transform (SHT)
  - Synthesize a sky image from its harmonic representation
    - Computation over rows of a 2D object (summation of spherical harmonics)
    - Communication to transpose the 2D object
    - Computation over columns of the 2D object (FFTs)



Map making, with R. Stompor, M. Szydlarski
Results obtained on Hopper, Cray XE6, NERSC



SHT, with R. Stompor, M. Szydlarski
Simulation on a petascale computer

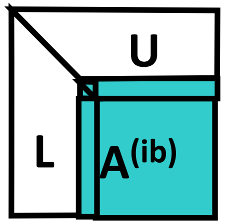
Motivation

- The communication problem needs to be taken into account higher in the computing stack
- A paradigm shift in the way the numerical algorithms are devised is required
- Communication avoiding algorithms - a novel perspective for numerical linear algebra
  - Minimize volume of communication
  - Minimize number of messages
  - Minimize over multiple levels of memory/parallelism
  - Allow redundant computations (preferably as a low order term)

Evolution of numerical libraries

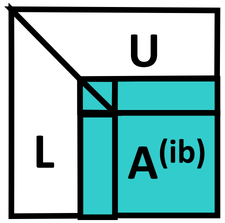
LINPACK (70's)

- vector operations, use BLAS1
- HPL benchmark based on Linpack LU factorization



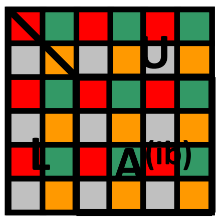
LAPACK (80's)

- Block versions of the algorithms used in LINPACK
- Uses BLAS3



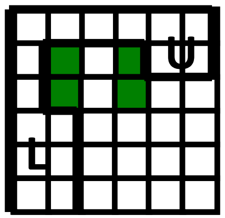
ScaLAPACK (90's)

- Targets distributed memories
- 2D block cyclic distribution of data
- PBLAS based on message passing



PLASMA (2008): new algorithms

- Targets many-core
- Block data layout
- Low granularity, high asynchronicity



Project developed by U Tennessee Knoxville, UC Berkeley, other collaborators.

Source: inspired from J. Dongarra, UTK, J. Langou, CU Denver

Communication Complexity of Dense Linear Algebra

- Matrix multiply, using $2n^3$ flops (sequential or parallel)
  - Hong-Kung (1981), Irony/Tishkin/Toledo (2004)
  - Lower bound on Bandwidth = Ω (#flops / $M^{1/2}$)
  - Lower bound on Latency = Ω (#flops / $M^{3/2}$)

- Same lower bounds apply to LU using reduction
  - Demmel, LG, Hoemmen, Langou 2008

$$\begin{pmatrix} I & & -B \\ A & I & \\ & & I \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ & & I \end{pmatrix} \cdot \begin{pmatrix} I & & -B \\ & I & AB \\ & & I \end{pmatrix}$$

- And to almost all direct linear algebra [Ballard, Demmel, Holtz, Schwartz, 09]

Lower bounds for linear algebra

- Computation modelled as an n-by-n-by-n set of lattice points
  (i,j,k) represents the operation $c(i,j) \mathrel{+}= f_{ij}(\ g_{ijk}\ (\ a(i,k)*b(k,j))\ )\ )$
- The computation is divided in S phases
- Each phase contains exactly M (the fast memory size) load and store instructions
- Determine how many flops the algorithm can compute in each phase, by applying discrete Loomis-Whitney inequality:

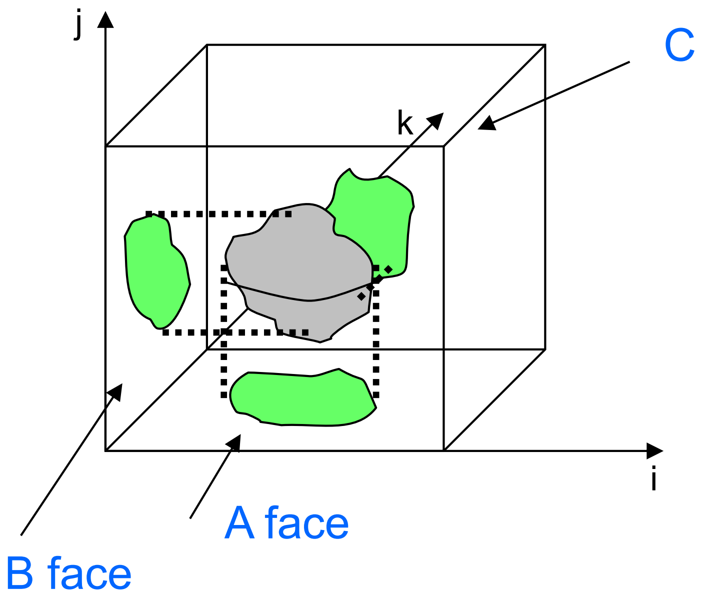
$$w^2 \le N_A N_B N_C$$

Algorithms in direct linear alg

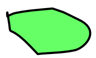
$for\ i, j, k = 1 : n$

$\quad c(i,j) = f_{ij}(g_{ijk}(a(i,k), b(k,j)))$

$endfor$



 - set of points in R^3, represent w arithmetics

 - orthogonal projections of the points onto coordinate planes N_A, N_B, N_C, represent values of A, B, C

Lower bounds for matrix multiplication (contd)

- Discrete Loomis-Whitney inequality:

$$w^2 \leq N_A N_B N_C$$

- Since there are at most 2M elements of A, B, C in a phase, the bound is:

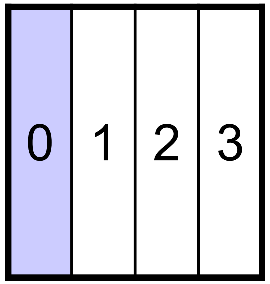
$$w \leq 2\sqrt{2} M^{3/2}$$

- The number of phases S is #flops/w, and hence the lower bound on communication is:

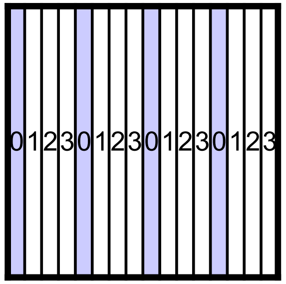
$$\#messages(S) \geq \frac{\#\,flops}{w} = \Omega\left(\frac{\#\,flops}{M^{3/2}}\right)$$

$$\#loads/stores \geq \Omega\left(\frac{\#\,flops}{M^{1/2}}\right)$$

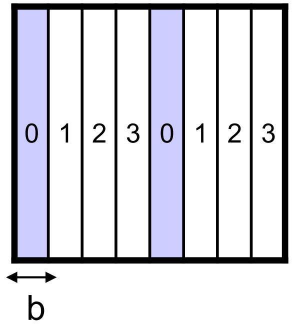
Matrix distributions



1) 1D Column Blocked Layout

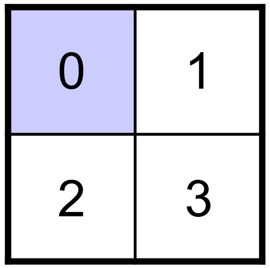


2) 1D Column Cyclic Layout

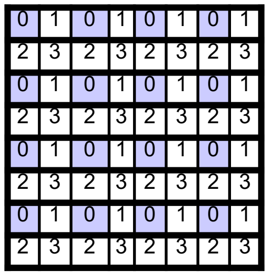


3) 1D Column Block Cyclic Layout

4) Row versions of the previous layouts



5) 2D Row and Column Blocked Layout



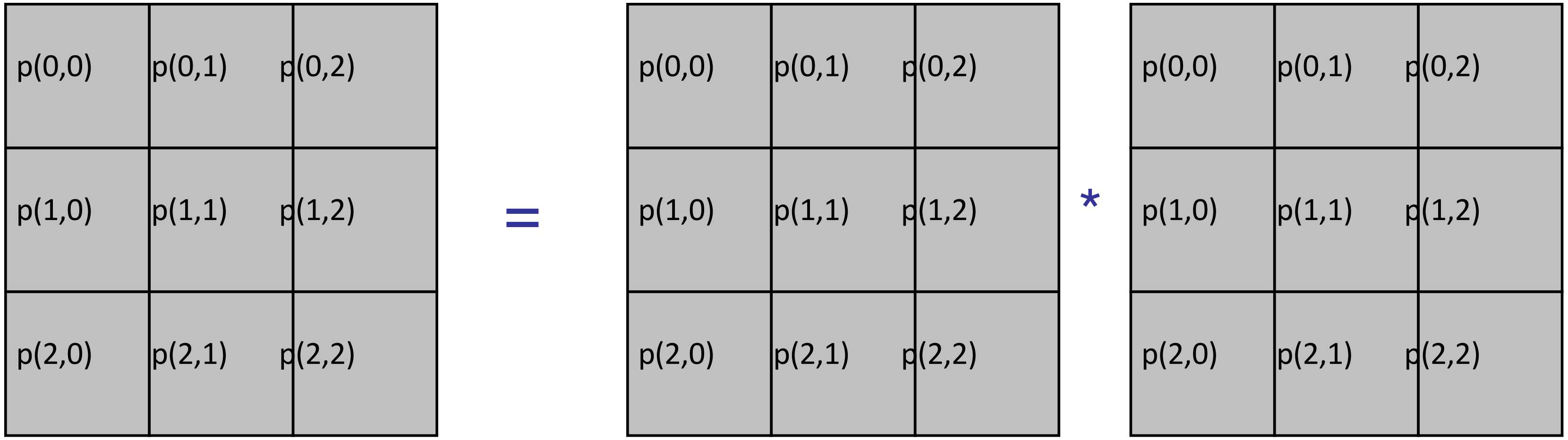
Generalizes others

6) 2D Row and Column Block Cyclic Layout

Source slide: J. Demmel

MatMul with 2D Layout

- Consider processors in 2D grid (physical or logical)
- Processors can communicate with 4 nearest neighbors
  - Broadcast along rows and columns



- Assume p processors form square s x s grid, $s = p^{1/2}$

Source slide: J. Demmel

Cannon's Algorithm

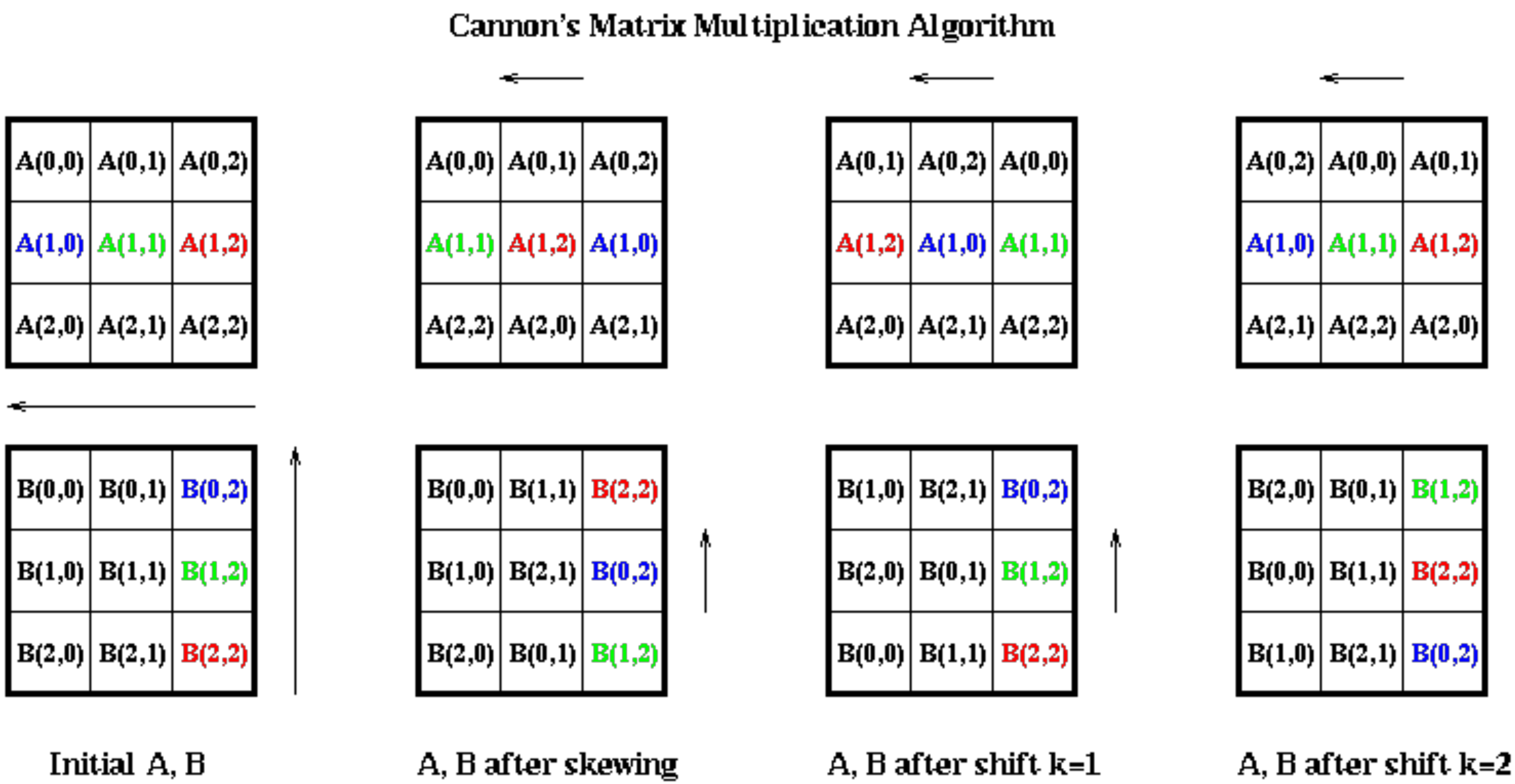
… $C(i,j) = C(i,j) + \sum_k A(i,k)*B(k,j)$

… assume s = sqrt(p) is an integer

```
forall  i=0 to s-1                … "skew" A
     left-circular-shift row i of A by i
     … so that A(i,j) overwritten by A(i,(j+i)mod s)
forall  i=0 to s-1                … "skew" B
     up-circular-shift column i of B by i
     … so that B(i,j) overwritten by B((i+j)mod s), j)
for k=0 to s-1         … sequential
     forall i=0 to s-1 and j=0 to s-1    … all processors in parallel
          C(i,j) = C(i,j) + A(i,j)*B(i,j)
          left-circular-shift each row of A by 1
          up-circular-shift each column of B by 1
```

Source slide: J. Demmel

Cannon's Matrix Multiplication



C(1,2) = A(1,0) * B(0,2) + A(1,1) * B(1,2) + A(1,2) * B(2,2)

Source slide: J. Demmel

Cost of Cannon's Algorithm

forall i=0 to s-1 … recall $s = \mathrm{sqrt}(p)$
    left-circular-shift row i of A by i … cost $\le s*(\alpha + \beta*n^2/p)$
forall i=0 to s-1
    up-circular-shift column i of B by i … cost $\le s*(\alpha + \beta*n^2/p)$
for k=0 to s-1
    forall i=0 to s-1 and j=0 to s-1
        C(i,j) = C(i,j) + A(i,j)*B(i,j) … cost $= 2*(n/s)^3 = 2*n^3/p^{3/2}$
        left-circular-shift each row of A by 1 … cost $= \alpha + \beta*n^2/p$
        up-circular-shift each column of B by 1 … cost $= \alpha + \beta*n^2/p$

- Total Time $= 2*n^3/p + 4*s*\alpha + 4*\beta*n^2/s$ - Optimal!
- Parallel Efficiency $= 2*n^3 / (p * \text{Total Time})$
  $= 1/(1 + \alpha * 2*(s/n)^3 + \beta * 2*(s/n))$
  $= 1/(1 + O(\mathrm{sqrt}(p)/n))$
- Grows to 1 as n/s = n/sqrt(p) = sqrt(data per processor) grows

Source slide: J. Demmel