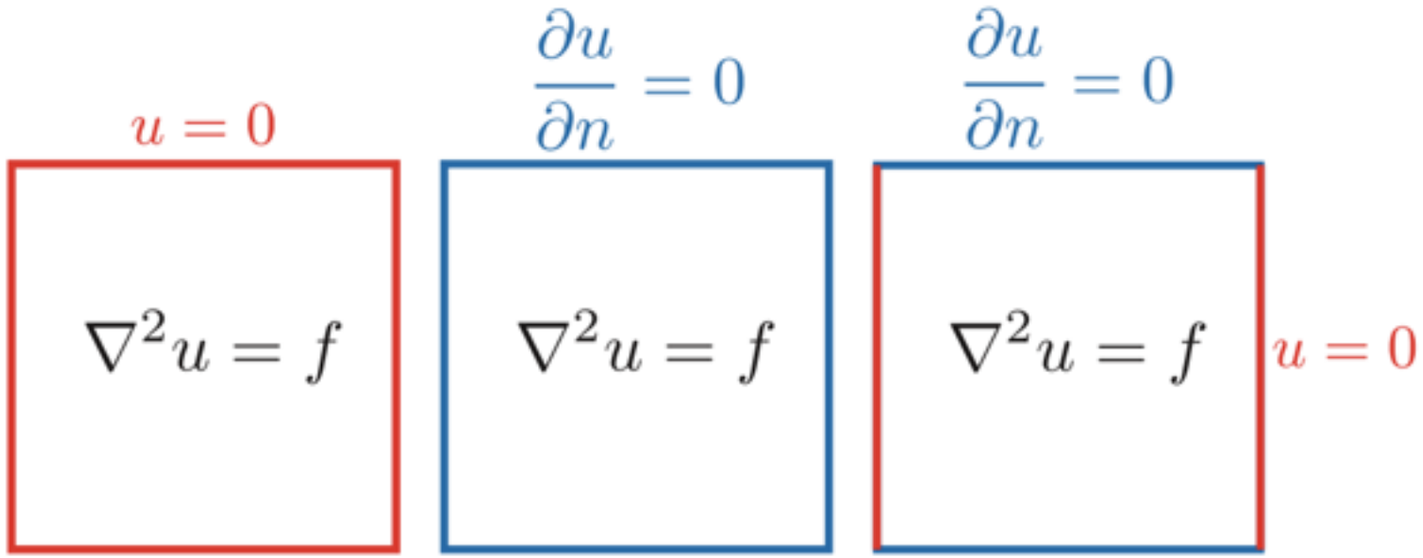
05/09	Class 9	Dense direct solvers	Understand the principle of LU decomposition and the optimization and parallelization techniques that lead to the LINPACK benchmark.
05/12	Class 10	Dense eigensolvers	Determine eigenvalues and eigenvectors and understand the fast algorithms for diagonalization and orthonormalization.
05/16	Class 11	Sparse direct solvers	Understand reordering in AMD and nested dissection, and fast algorithms such as skyline and multifrontal methods.
05/19	Class 12	Sparse iterative solvers	Understand the notion of positive definiteness, condition number, and the difference between Jacobi, CG, and GMRES.
05/23	Class 13	Preconditioners	Understand how preconditioning affects the condition number and spectral radius, and how that affects the CG method.
05/26	Class 14	Multigrid methods	Understand the role of smoothers, restriction, and prolongation in the V-cycle.
05/30	Class 15	Fast multipole methods, H-matrices	Understand the concept of multipole expansion and low-rank approximation, and the role of the tree structure.

Dense linear algebra

Linear systems	$Ax = b$	$A = LDU$
Least squares	$\|\|Ax - b\|\|$	
Eigenvalues	$Ax = \lambda x$	$A = Q\Lambda Q^{-1}$
Singular values	$A^T Ax = \sigma^2 x$	$A = U\Sigma V$

Linear system



(a) Dirichlet (b) Neumann (c) Mixed

$$N_\Gamma \left\{ \overbrace{\begin{bmatrix} \ddots & & \\ & \frac{1}{2}\delta_{ij} + \frac{\partial G(r_{ij})}{\partial n} & \\ & & \ddots \end{bmatrix}}^{} \underbrace{\begin{bmatrix} \vdots \\ u_j \\ \vdots \end{bmatrix}}_{\text{unknown}} \right. = \overbrace{\begin{bmatrix} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{bmatrix}}^{} \begin{bmatrix} \vdots \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{bmatrix} - \overbrace{\begin{bmatrix} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{bmatrix}}^{} \begin{bmatrix} \vdots \\ f_j \\ \vdots \end{bmatrix}$$

$$N_\Gamma \left\{ \overbrace{\left[\begin{array}{ccc} \ddots & & \\ G(r_{ij}) & -\frac{1}{2}\delta_{ij} - \frac{\partial G(r_{ij})}{\partial n} & \\ & \ddots & \end{array}\right]}^{N_\Gamma} \underbrace{\begin{bmatrix} \vdots \\ \frac{\partial u_j}{\partial n} \\ u_j \\ \vdots \end{bmatrix}}_{\text{unknown}} \right. = \overbrace{\left[\begin{array}{ccc} \ddots & & \\ \frac{1}{2}\delta_{ij} + \frac{\partial G(r_{ij})}{\partial n} & & -G(r_{ij}) \\ & \ddots & \end{array}\right]}^{N_\Gamma} \begin{bmatrix} \vdots \\ \frac{u_j}{\frac{\partial u_j}{\partial n}} \\ \vdots \end{bmatrix} + \overbrace{\begin{bmatrix} \ddots & & \\ & G(r_{ij}) & \\ & & \ddots \end{bmatrix}}^{N_\Omega} \begin{bmatrix} \vdots \\ f_j \\ \vdots \end{bmatrix}$$

bem/step02.py

```
for i in range(0,n):
    if bc[i] == 1:
        tmp = G[:,i]
        G[:,i] = -H[:,i]
        H[:,i] = -tmp

b = H.dot(u)
```

```
un = numpy.linalg.solve(G,b)
```

```
for i in range(0,n):
```

$$N_\Gamma \left\{ \overbrace{\left[\begin{array}{c|c} \ddots & \\ G(r_{ij}) & -\frac{1}{2}\delta_{ij} - \frac{\partial G(r_{ij})}{\partial n} \\ & \ddots \end{array}\right]}^{N_\Gamma} \underbrace{\left[\begin{array}{c} \vdots \\ \frac{\partial u_j}{\partial n} \\ u_j \\ \vdots \end{array}\right]}_{\text{unknown}} = \overbrace{\left[\begin{array}{c|c} \ddots & \\ \frac{1}{2}\delta_{ij} + \frac{\partial G(r_{ij})}{\partial n} & -G(r_{ij}) \\ & \ddots \end{array}\right]}^{N_\Gamma} \left[\begin{array}{c} \vdots \\ u_j \\ \frac{\partial u_j}{\partial n} \\ \vdots \end{array}\right] + \overbrace{\left[\begin{array}{cc} \ddots & \\ & G(r_{ij}) \\ & \ddots \end{array}\right]}^{N_\Omega} \left[\begin{array}{c} \vdots \\ f_j \\ \vdots \end{array}\right.$$

numpy.linalg.solve

numpy.linalg.solve(*a*, *b*) [source]

Solve a linear matrix equation, or system of linear scalar equations.

Computes the "exact" solution, *x*, of the well-determined, i.e., full rank, linear matrix equation *ax* = *b*.

Parameters:

a *: (..., M, M) array_like*
Coefficient matrix.

b *: {(..., M,), (..., M, K)}, array_like*
Ordinate or "dependent variable" values.

Returns:

x *: {(..., M,), (..., M, K)} ndarray*
Solution to the system a x = b. Returned shape is identical to *b*.

Raises:

LinAlgError
If *a* is singular or not square.

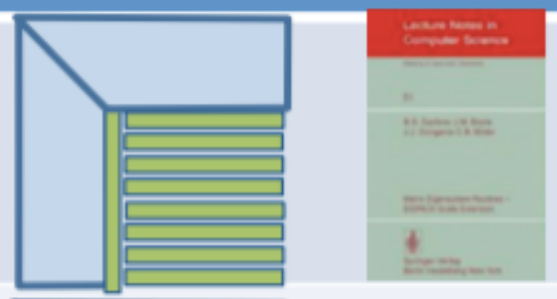
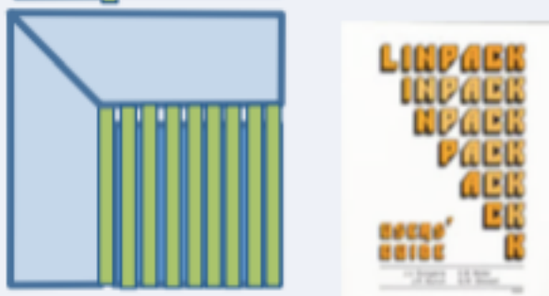
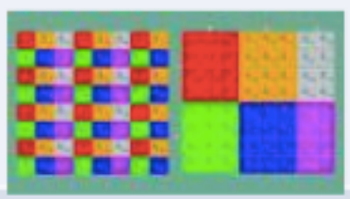
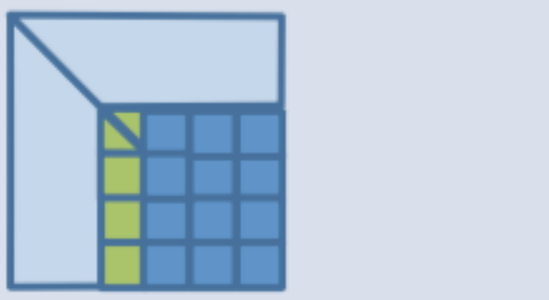
Notes

New in version 1.8.0.

Broadcasting rules apply, see the numpy.linalg documentation for details.

The solutions are computed using LAPACK routine _gesv

LAPACK

Software/Algorithms follow hardware evolution in time		
EISPACK (70's) (Translation of Algol)		Rely on - Fortran, but row oriented
LINPACK (80's) (Vector operations)		Rely on - Level-1 BLAS operations - Column oriented
LAPACK (90's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (00's) (Distributed Memory)	 	Rely on - PBLAS Mess Passing
PLASMA (10's) New Algorithms (many-core friendly)		Rely on - DAG/scheduler - block data layout - some extra kernels

http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

Mr. LAPACK

Jack Dongarra

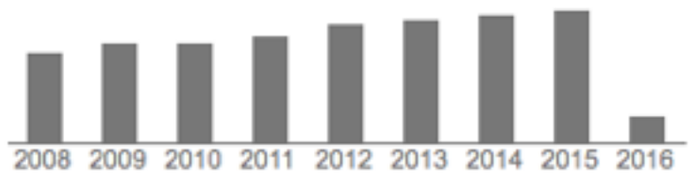
Follow

University of Tennessee; Oak Ridge National Laboratory; University of Manchester
scientific parallel computing
Verified email at eecs.utk.edu - Homepage

Title 1–20	Cited by	Year
LAPACK Users' Guide, SIAM, Philadelphia, 1999 E Anderson, Z Bai, C Bischof, S Blackford, J Demmel, J Dongarra, ...	6661 *	
LAPACK Users' Guide, Third Edition E Anderson, Z Bai, C Bischof, S Blackford, J Demmel, J Dongarra, ... Society for Industrial and Applied Mathematics	~~6618~~ *	1999
LAPACK Users' guide E Anderson, Z Bai, C Bischof Society for Industrial Mathematics	~~6616~~	1999
LAPACK Users' Guide, Second Edition E Anderson, Z Bai, C Bischof, S Blackford, J Demmel, J Dongarra, ... Society for Industrial and Applied Mathematics	~~6616~~ *	1995
LAPACK users' guide 2nd ed E Anderson, Z Bai, C Bischof, J Demmel, J Dongarra, J Du Croz, ... Soc. for Ind. and App	~~6616~~ *	1995

Google Scholar

Citation indices	All	Since 2011
Citations	90300	28026
h-index	114	69
i10-index	878	459



Co-authors View all...

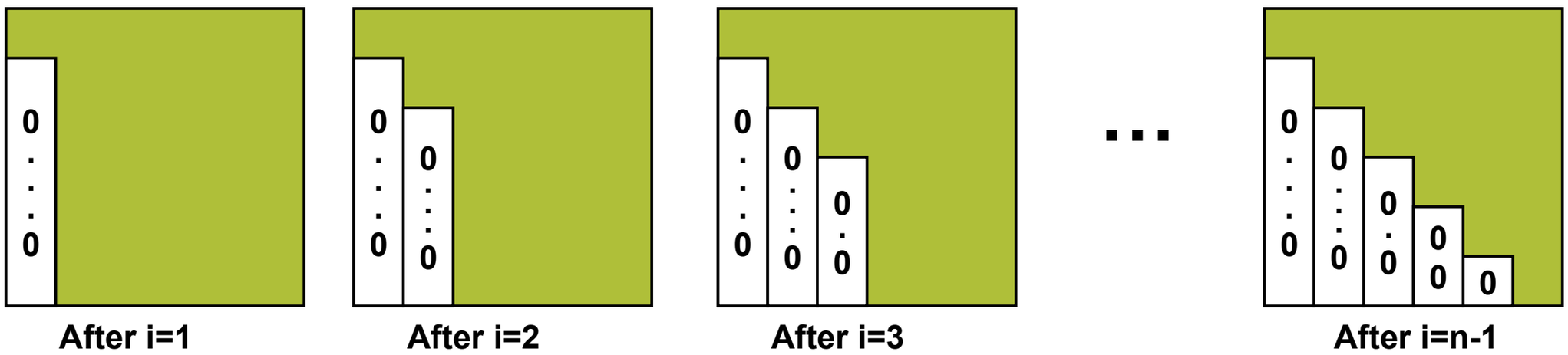
Zhaojun Bai
Piotr Luszczek
Sven Hammarling
Stanimire Tomov
Danny Sorensen
V. S. Sunderam
George Bosilca
David Walker

next few slides are from Jack Dongarra …

Gaussian Elimination (GE) for Solving Ax=b

- Add multiples of each row to later rows to make A upper triangular
- Solve resulting triangular system Ux = c by substitution

```
… for each column i
… zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    … for each row j below row i
    for j = i+1 to n
        … add a multiple of row i to row j
        tmp = A(j,i);
        for k = i to n
            A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
```



http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

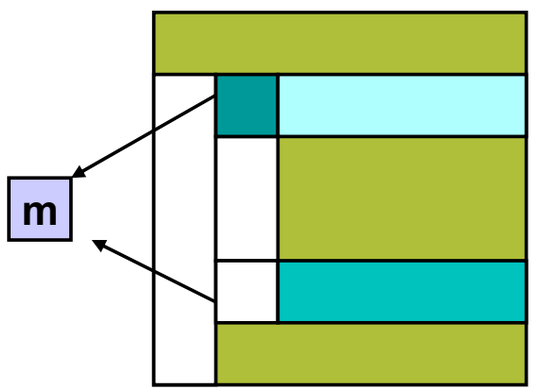
Refine GE Algorithm (1)

- Initial Version

```
… for each column i
… zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    … for each row j below row i
    for j = i+1 to n
        … add a multiple of row i to row j
        tmp = A(j,i);
        for k = i to n
            A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
```

- Remove computation of constant tmp/A(i,i) from inner loop.

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```



http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

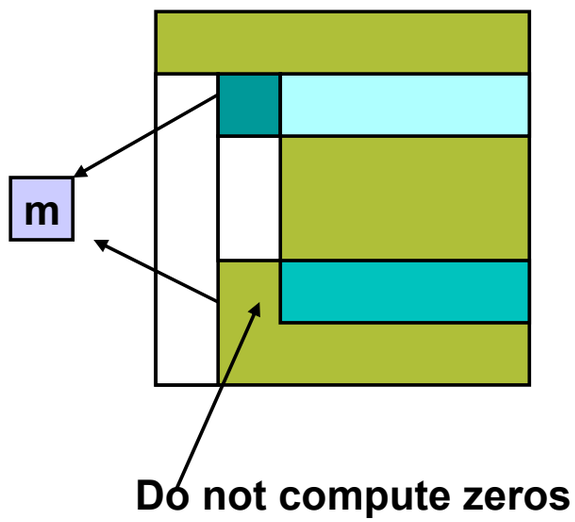
Refine GE Algorithm (2)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- Don't compute what we already know: zeros below diagonal in column i

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```



http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

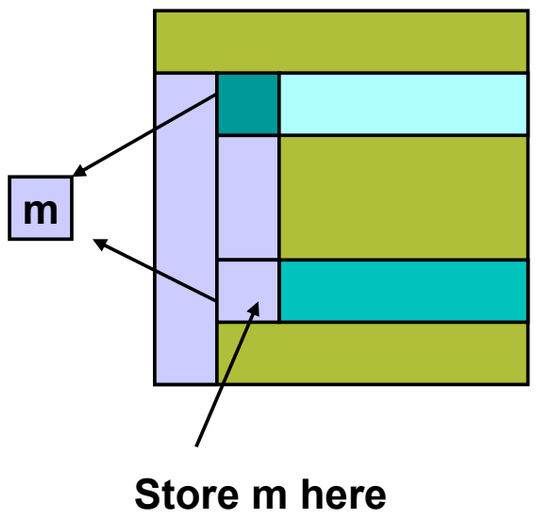
Refine GE Algorithm (3)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- Store multipliers m below diagonal in zeroed entries for later use

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```



http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

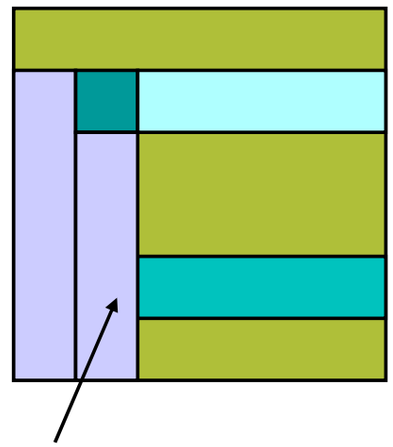
Refine GE Algorithm (4)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- **Split Loop**

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```



Store all m's here before updating rest of matrix

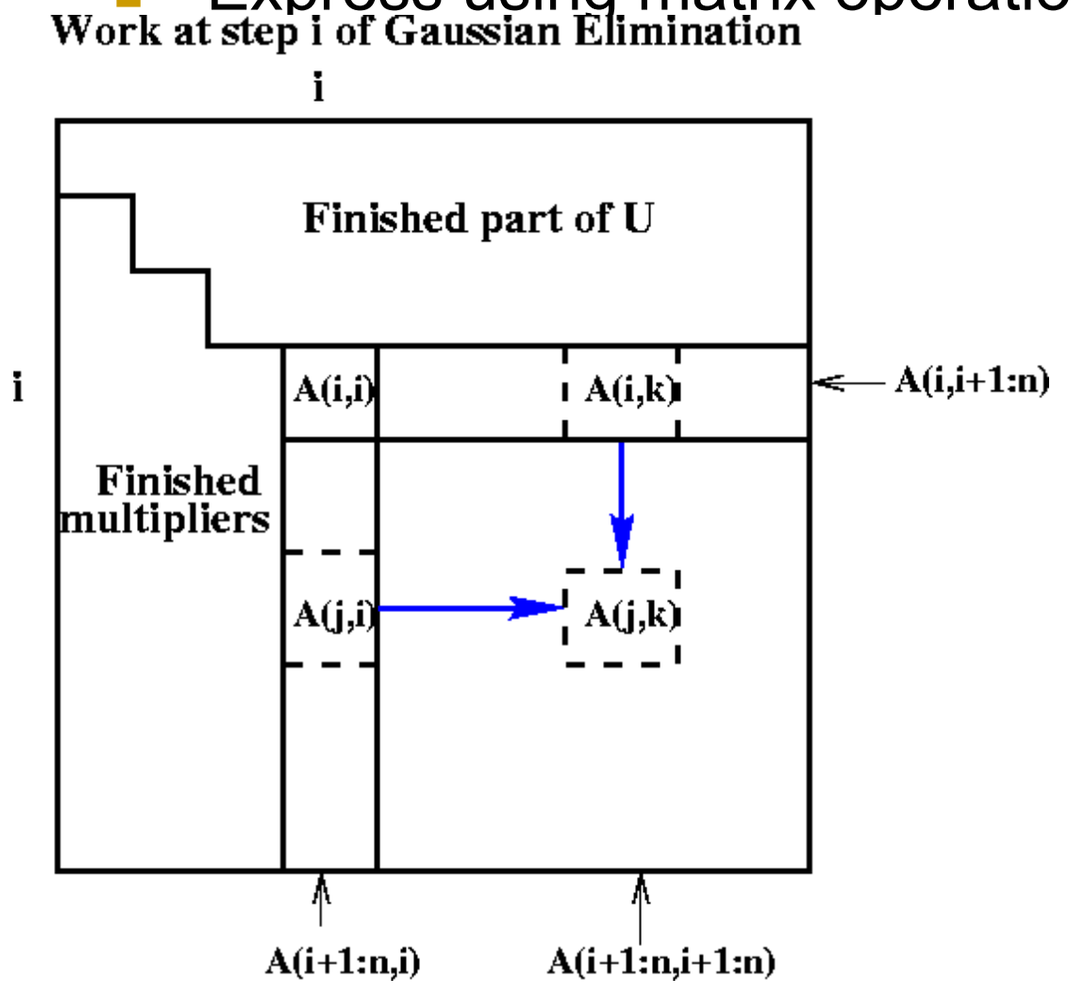
http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

Refine GE Algorithm (5)

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- Last version

- Express using matrix operations (BLAS)




```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) * ( 1 / A(i,i) )
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
        - A(i+1:n , i) * A(i , i+1:n)
```

http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

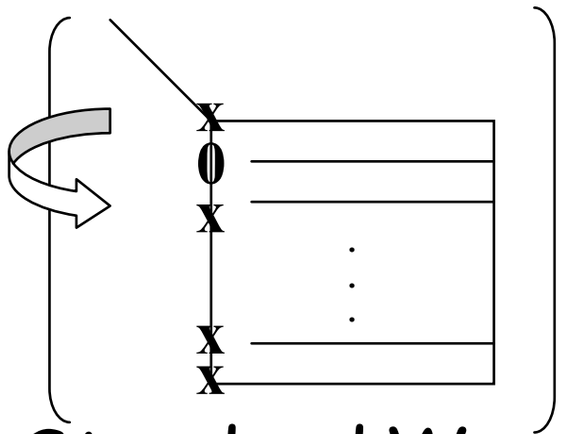
What GE really computes

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

- Call the strictly lower triangular matrix of multipliers M, and let L = I+M
- Call the upper triangle of the final matrix U
- *Lemma (LU Factorization):* If the above algorithm terminates (does not divide by zero) then A = L*U
- Solving A*x=b using GE
  - Factorize A = L*U using GE (cost = 2/3 n^3 flops)
  - Solve L*y = b for y, using substitution (cost = n^2 flops)
  - Solve U*x = y for x, using substitution (cost = n^2 flops)
- Thus A*x = (L*U)*x = L*(U*x) = L*y = b as desired

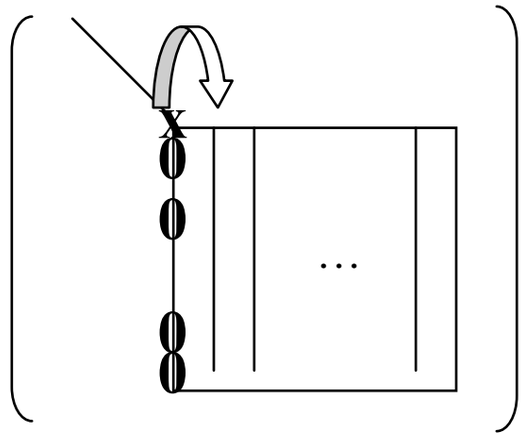
http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

Gaussian Elimination



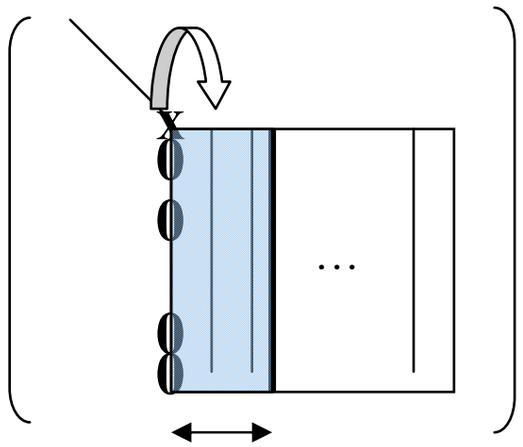
Standard Way

subtract a multiple of a row



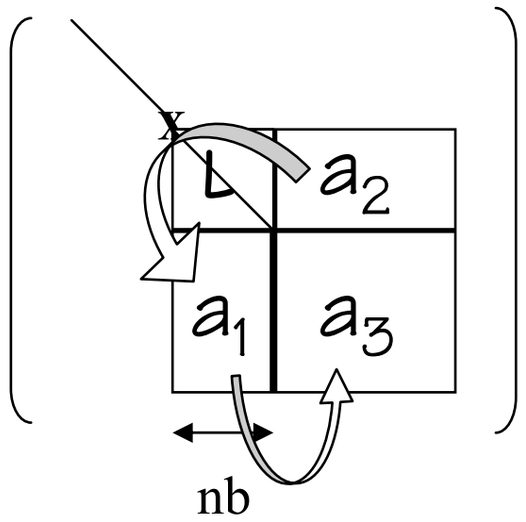
LINPACK

apply sequence to a column



LAPACK

apply sequence to nb



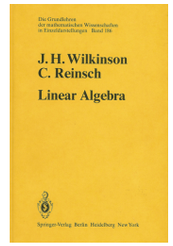
$a_2 = L^{-1} a_2$

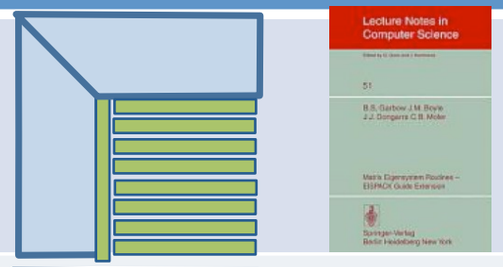
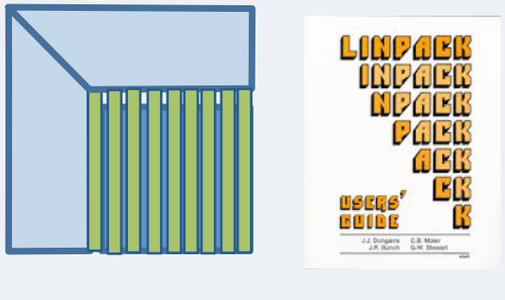
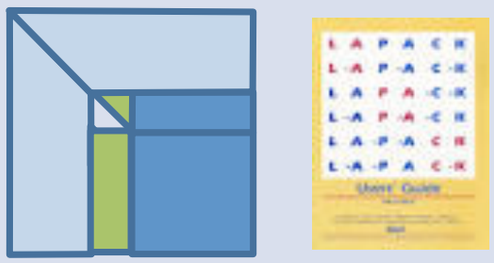
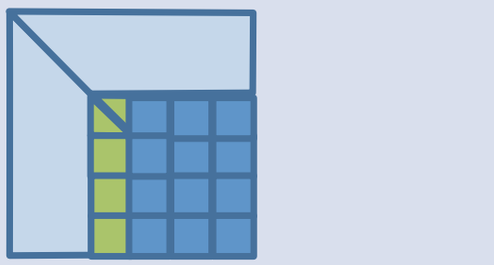
$a_3 = a_3 - a_1 * a_2$

then apply nb to rest of matrix

http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2007/Lect11.pdf

40 Years Evolving SW and Alg Tracking Hardware Developments

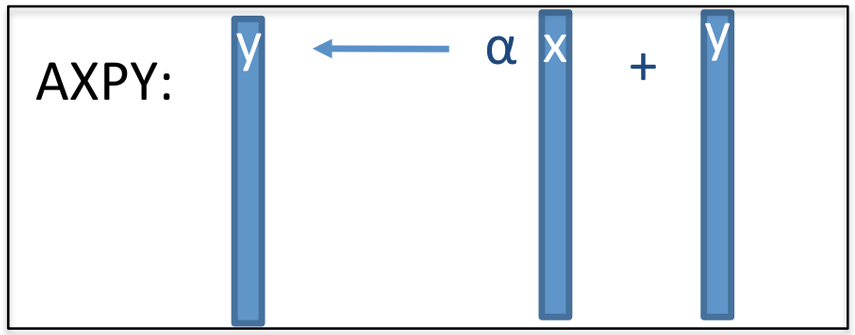


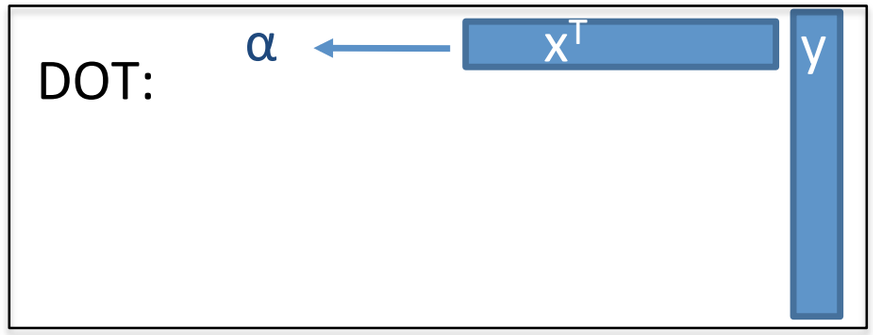
Software/Algorithms follow hardware evolution in time		
EISPACK (70's) (Translation of Algol)		Rely on - Fortran, but row oriented
LINPACK (80's) (Vector operations)		Rely on - Level-1 BLAS operations - Column oriented
LAPACK (90's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (00's) (Distributed Memory)		Rely on - PBLAS Mess Passing
PLASMA (10's) New Algorithms (many-core friendly)		Rely on - DAG/scheduler - block data layout - some extra kernels

http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

Level 1, 2 and 3 BLAS

Level 1 BLAS Matrix-Vector operations

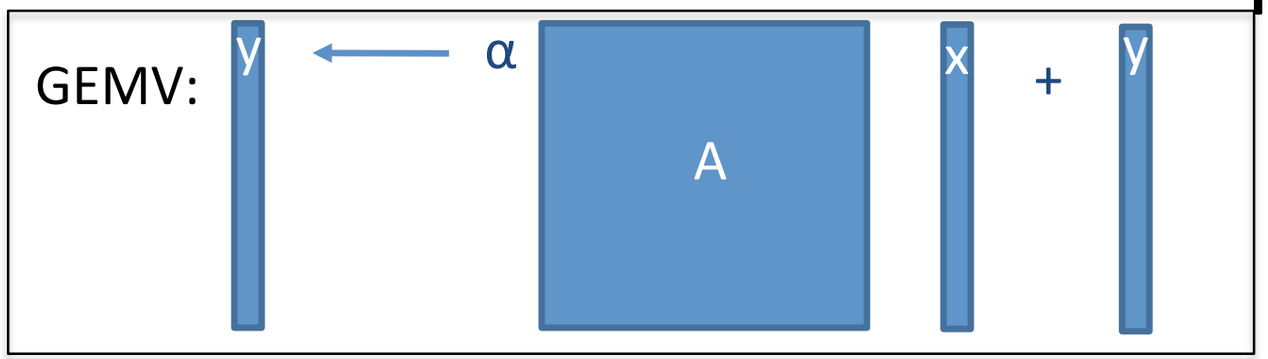




2n FLOP
2n memory reference
AXPY: 2n READ, n WRITE
DOT: 2n READ

RATIO: 1

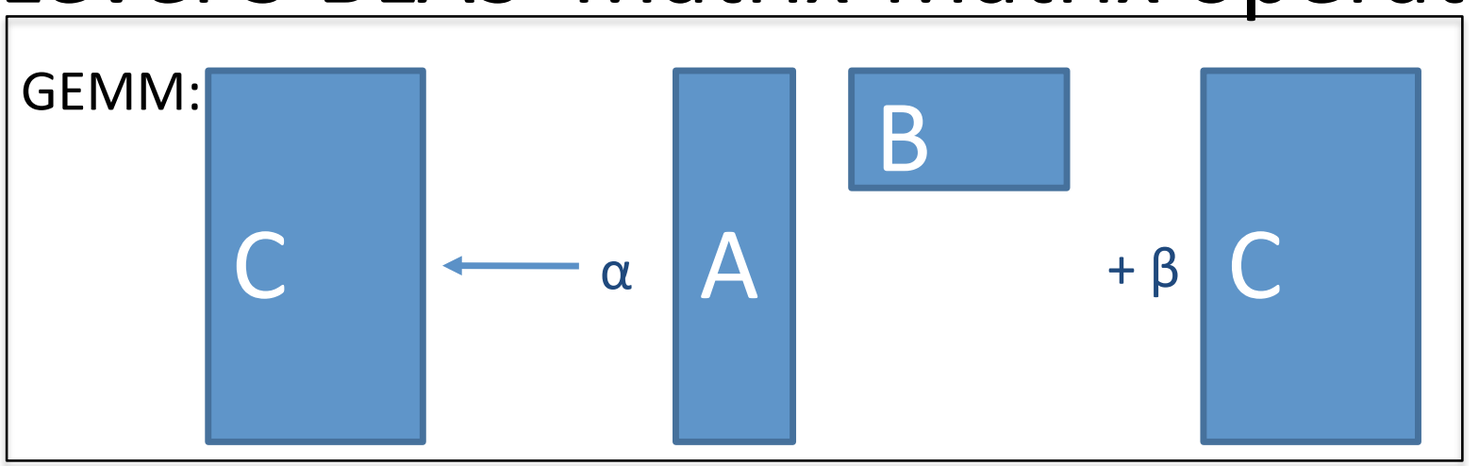
Level 2 BLAS Matrix-Vector operations



$2n^2$ FLOP
n^2 memory references

RATIO: 2

Level 3 BLAS Matrix-Matrix operations



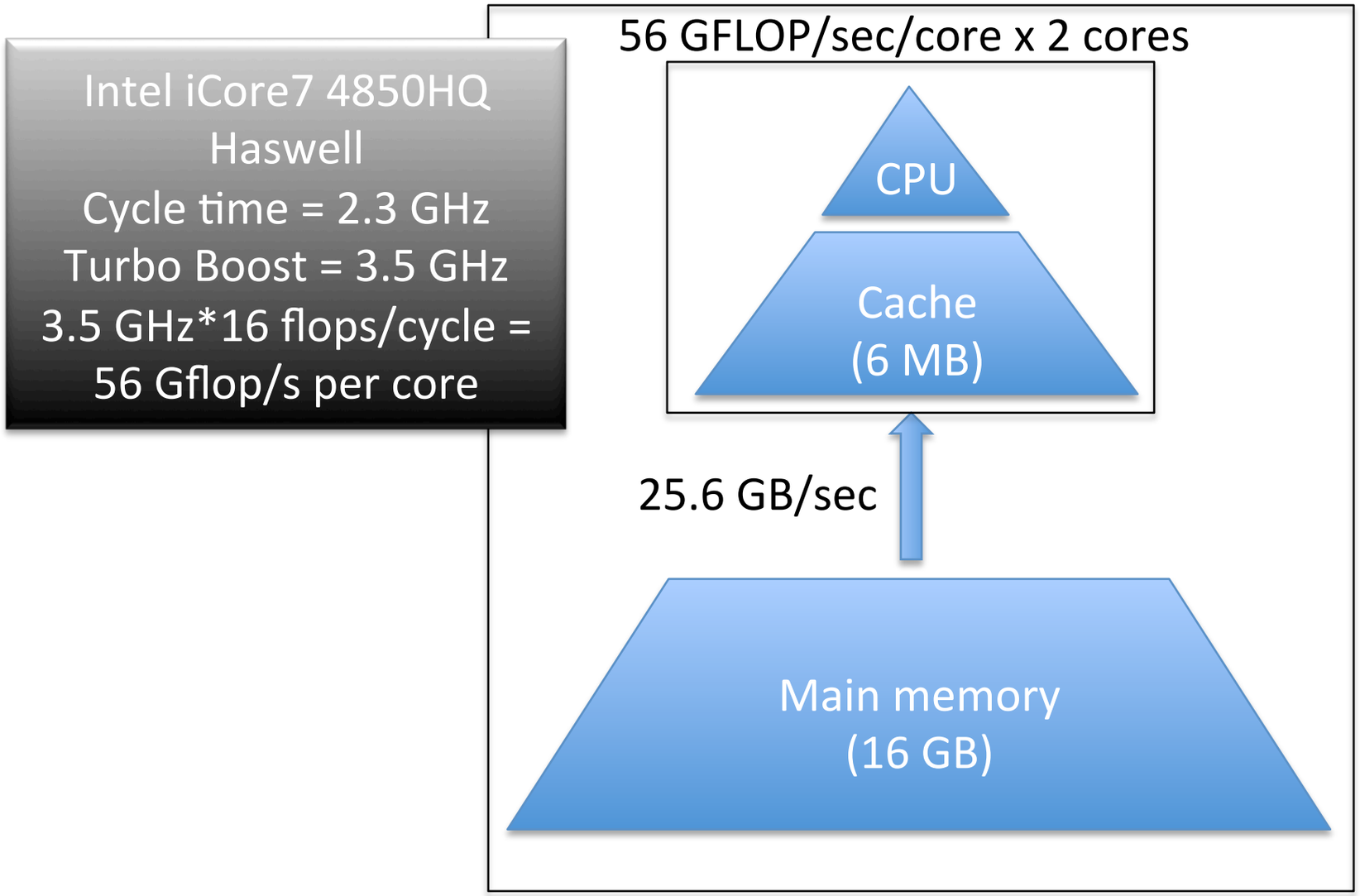
$2n^3$ FLOP
$3n^2$ memory references
$3n^2$ READ, n^2 WRITE

RATIO: 2/3 n

http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

Memory transfer

- One level of memory model on my laptop:

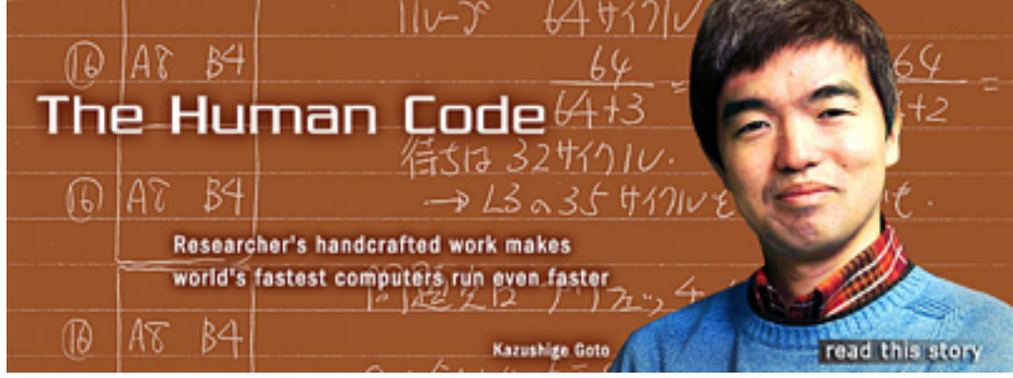


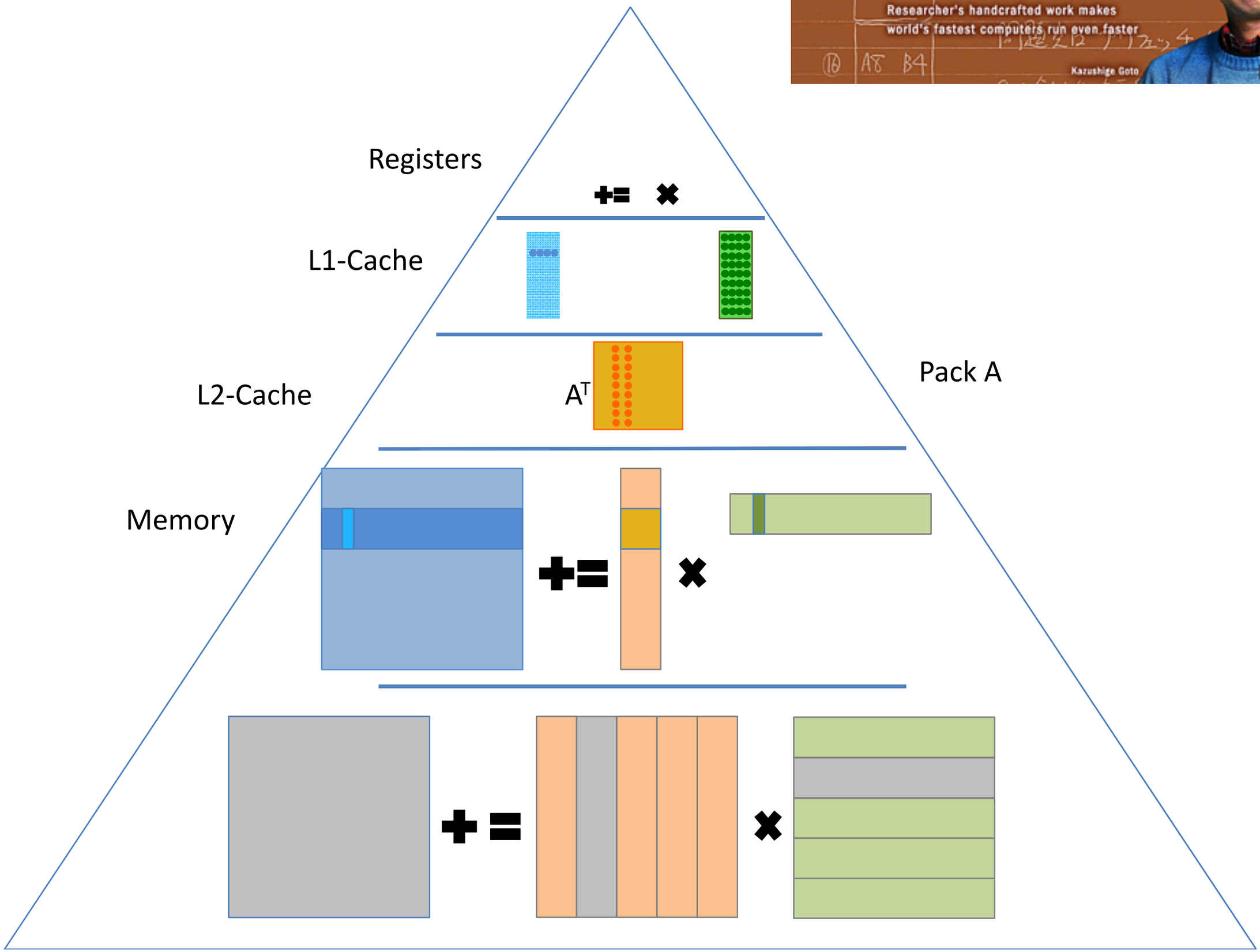
(Omitting latency here.)

The model IS simplified (see next slide) but it provides an upper bound on performance as well. I.e., we will never go faster than what the model predicts.
(And, of course, we can go slower …)

http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

Goto BLAS

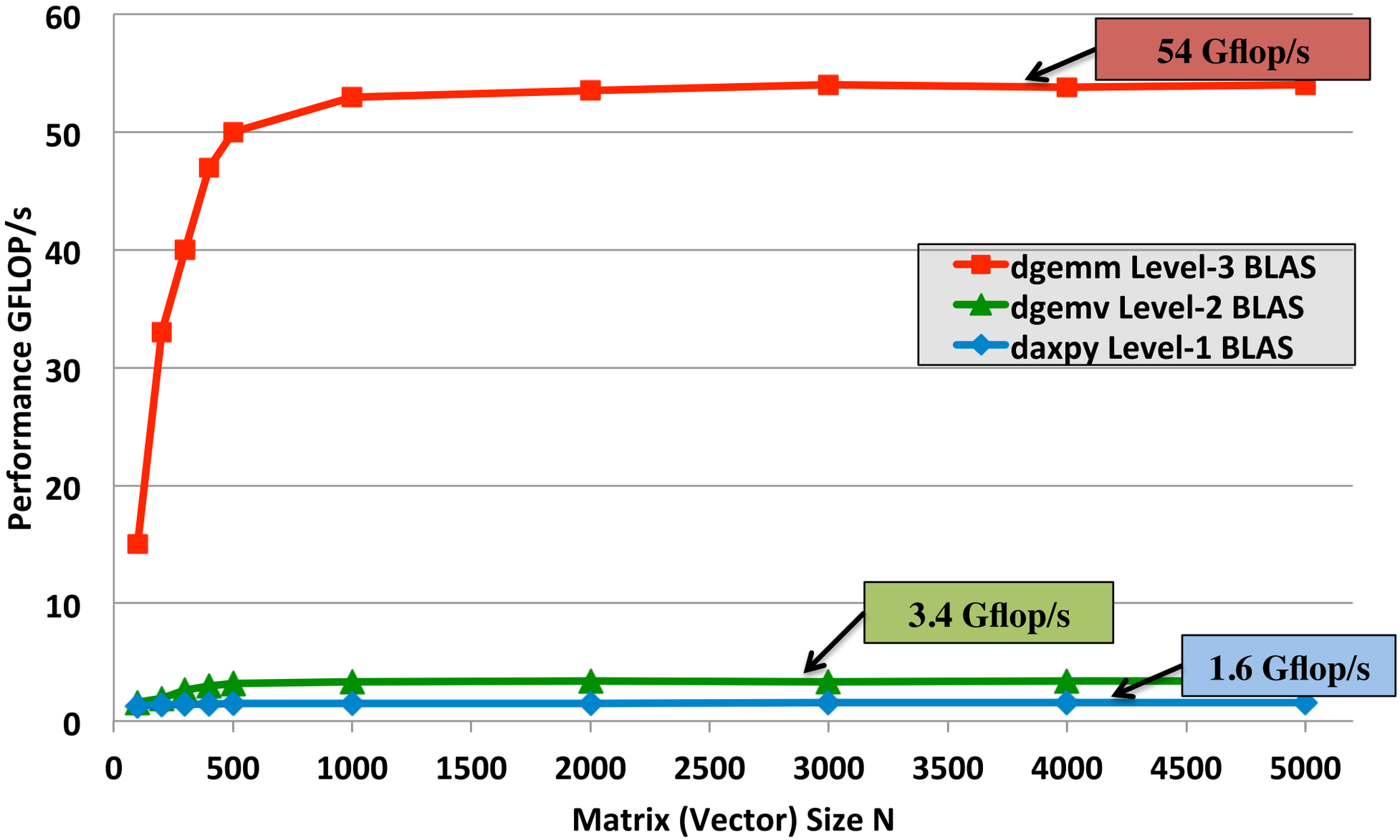




http://www.sc.informatik.tu-darmstadt.de/media/fg_bischof/material_2/02_van_de_Geijn.pdf

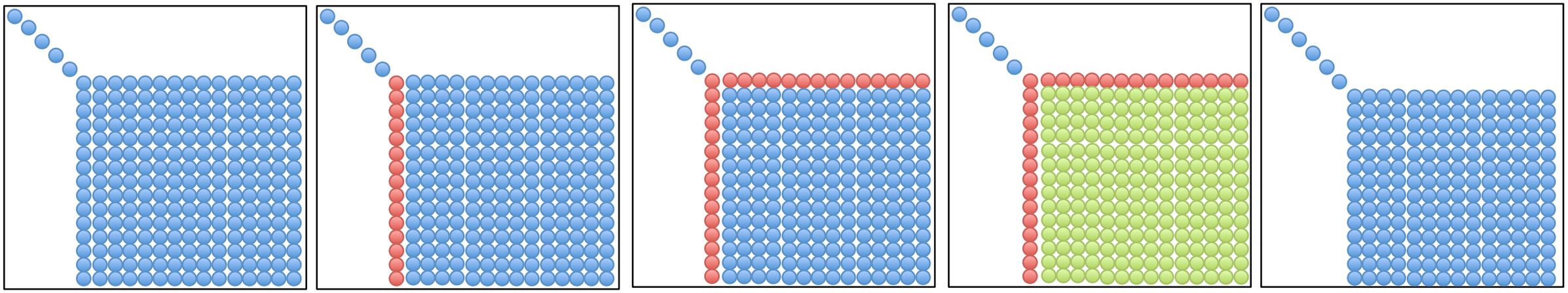
Level 1, 2 and 3 BLAS

1 core Intel Haswell i7-4850HQ, 2.3 GHz (Turbo Boost at 3.5 GHz);
Peak = 56 Gflop/s



1 core Intel Haswell i7-4850HQ, 2.3 GHz, Memory: DDR3L-1600MHz
6 MB shared L3 cache, and each core has a private 256 KB L2 and 64 KB L1.
The theoretical peak per core double precision is 56 Gflop/s per core.
Compiled with gcc and using Veclib

The Standard LU Factorization LINPACK 1970's HPC of the Day: Vector Architecture



Factor column with Level 1 BLAS

Divide by Pivot row

Schur complement update (Rank 1 update)

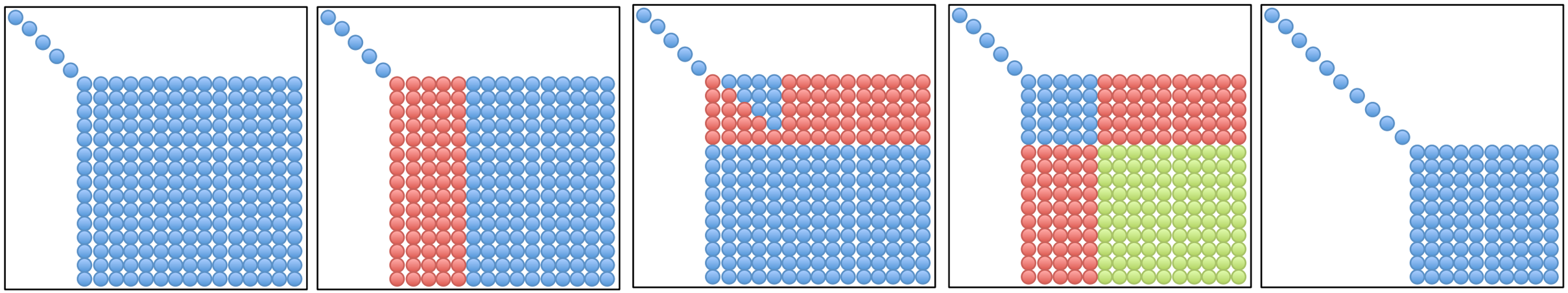
Next Step

Main points

- Factorization column (zero) mostly sequential due to memory bottleneck
- Level 1 BLAS
- Divide pivot row has little parallelism
- Rank -1 Schur complement update is the only easy parallelize task
- Partial pivoting complicates things even further
- Bulk synchronous parallelism (fork-join)
  - Load imbalance
  - Non-trivial Amdahl fraction in the panel
  - Potential workaround (look-ahead) has complicated implementation

http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

The Standard LU Factorization LAPACK
1980's HPC of the Day: Cache Based SMP



Factor panel with Level 1,2 BLAS

Triangular update

Schur complement update

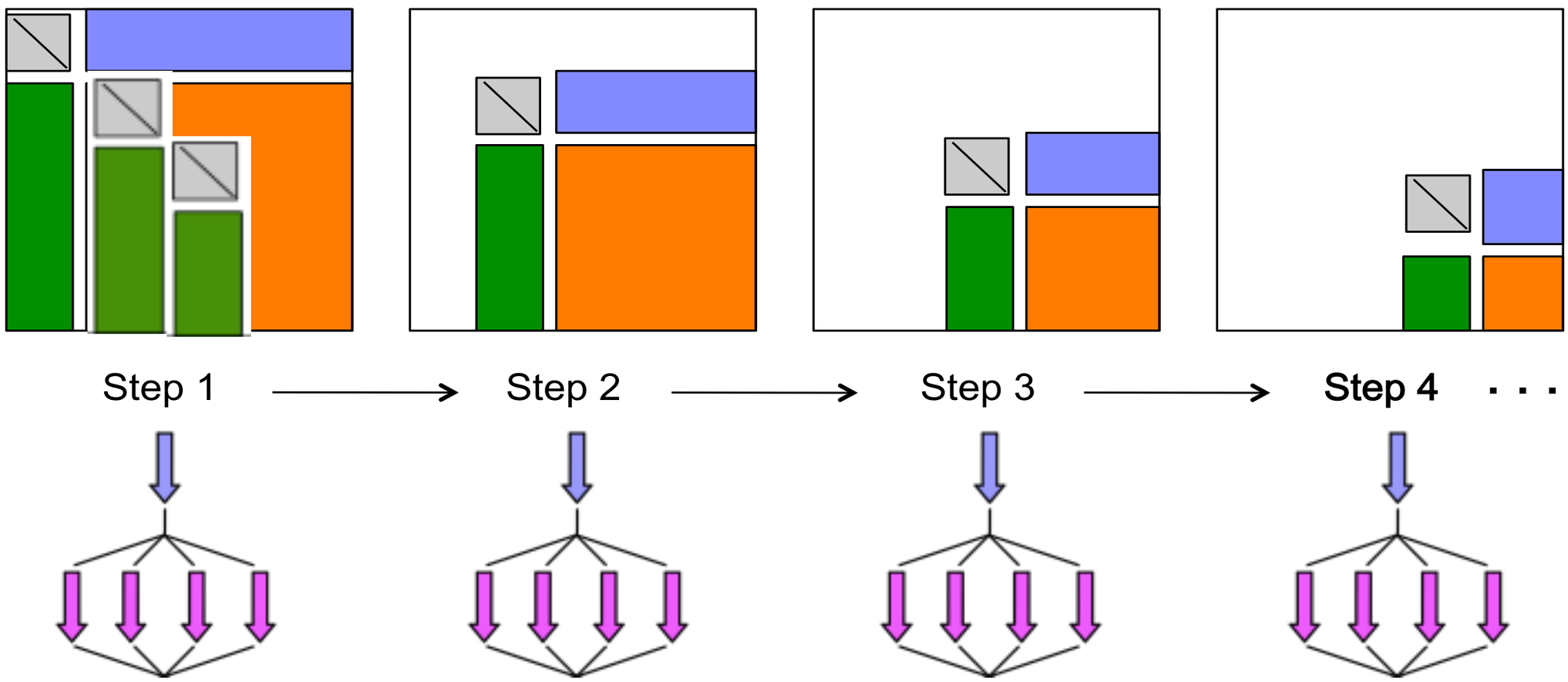
Next Step

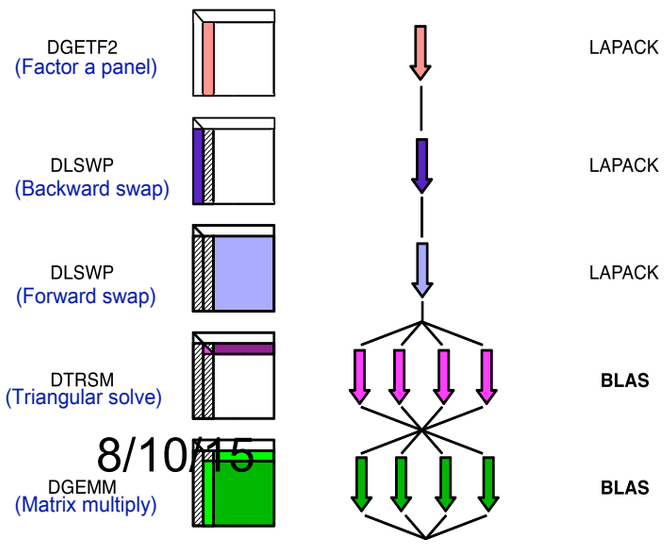
Main points

- Panel factorization mostly sequential due to memory bottleneck
- Triangular solve has little parallelism
- Schur complement update is the only easy parallelize task
- Partial pivoting complicates things even further
- Bulk synchronous parallelism (fork-join)
  - Load imbalance
  - Non-trivial Amdahl fraction in the panel
  - Potential workaround (look-ahead) has complicated implementation

http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

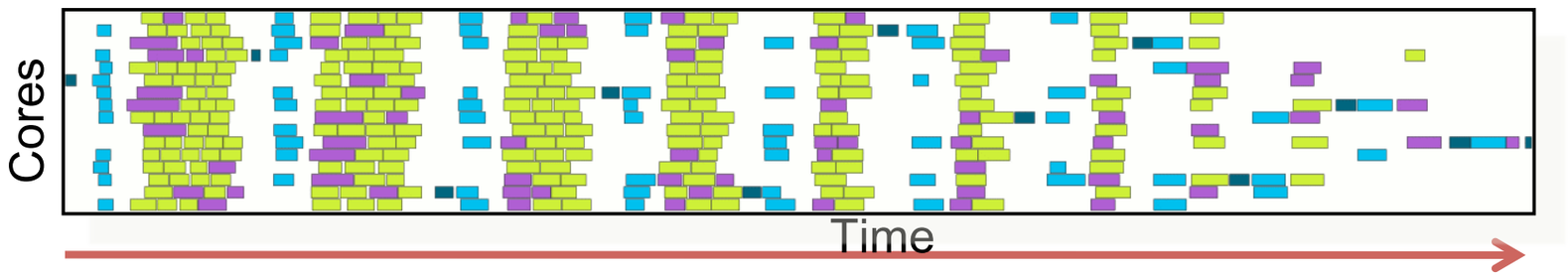
Synchronization (in LAPACK LU)





8/10/15

- fork join
- bulk synchronous processing

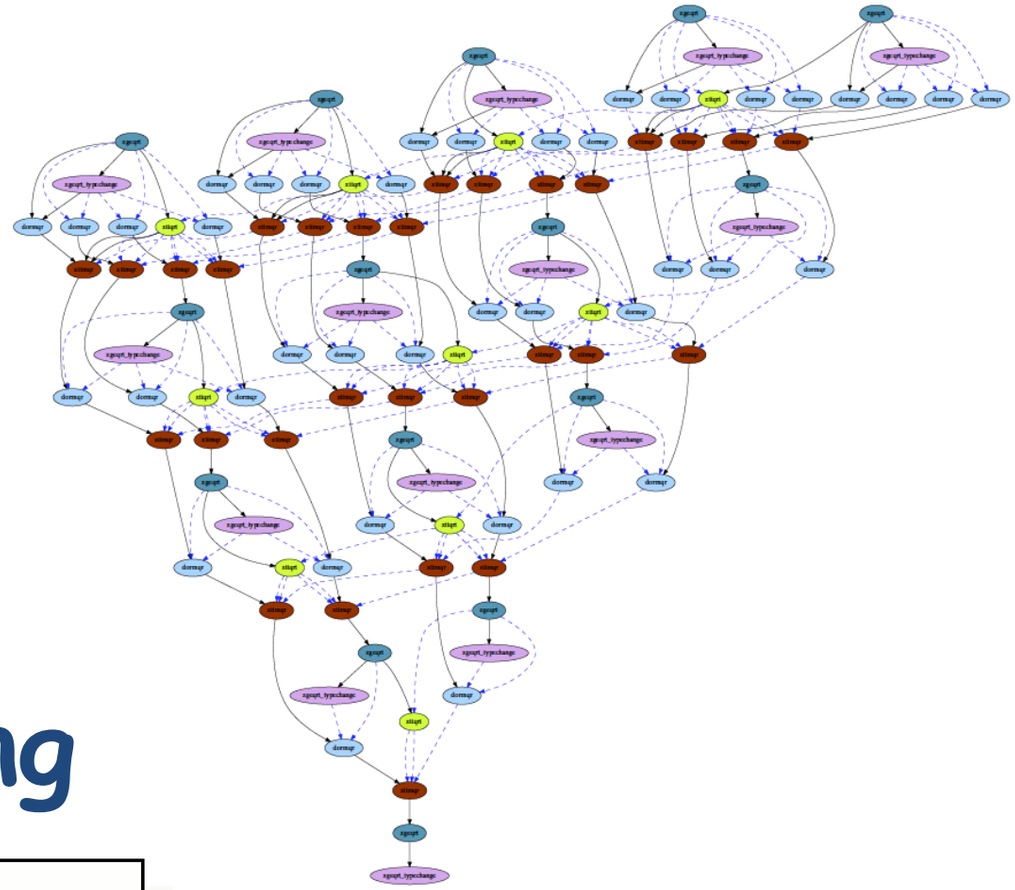


http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

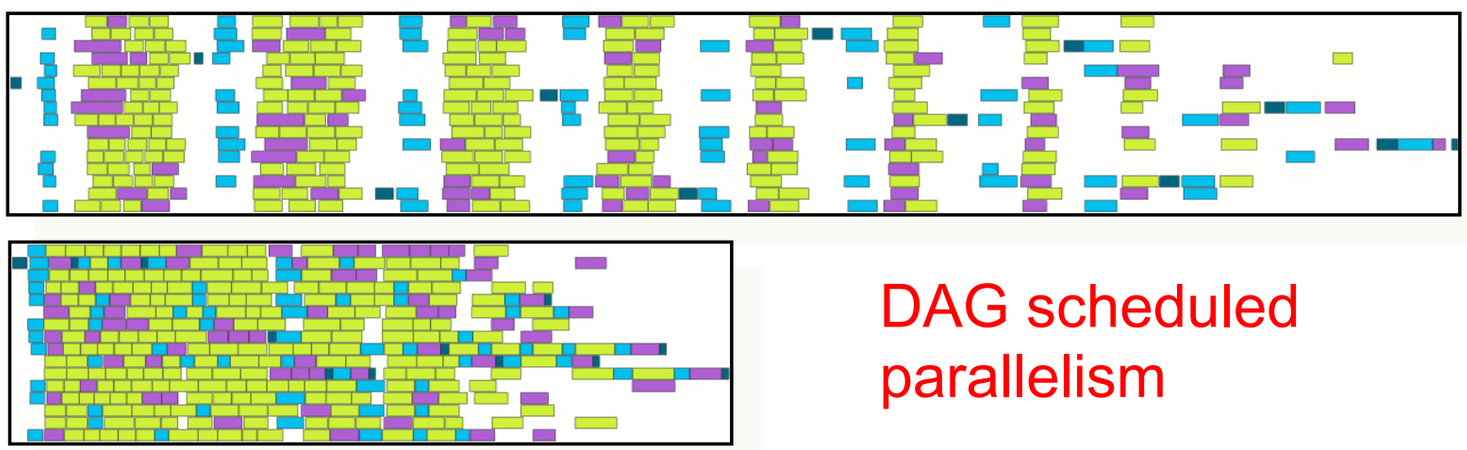
The Purpose of a QUARK Runtime

- Objectives
  - High utilization of each core
  - Scaling to large number of cores
  - Synchronization reducing algorithms
- Methodology
  - Dynamic DAG scheduling (QUARK)
  - Explicit parallelism
  - Implicit communication
  - Fine granularity / block data layout

- Arbitrary DAG with dynamic **scheduling**



Fork-join parallelism
Notice the synchronization penalty in the presence of heterogeneity.



DAG scheduled parallelism

http://extremecomputingtraining.anl.gov/files/2015/08/dongarra-anl-0815.pdf

05/09	Class 9	Dense direct solvers	Understand the principle of LU decomposition and the optimization and parallelization techniques that lead to the LINPACK benchmark.
05/12	Class 10	Dense eigensolvers	Determine eigenvalues and eigenvectors and understand the fast algorithms for diagonalization and orthonormalization.
05/16	Class 11	Sparse direct solvers	Understand reordering in AMD and nested dissection, and fast algorithms such as skyline and multifrontal methods.
05/19	Class 12	Sparse iterative solvers	Understand the notion of positive definiteness, condition number, and the difference between Jacobi, CG, and GMRES.
05/23	Class 13	Preconditioners	Understand how preconditioning affects the condition number and spectral radius, and how that affects the CG method.
05/26	Class 14	Multigrid methods	Understand the role of smoothers, restriction, and prolongation in the V-cycle.
05/30	Class 15	Fast multipole methods, H-matrices	Understand the concept of multipole expansion and low-rank approximation, and the role of the tree structure.