Sparse LU factorization of circuit simulation matrices

Tim Davis, Ken Stanley

davis@cise.ufl.edu, ken_s_stanley@yahoo.com

University of Florida, Sandia National Laboratory

Outline

- circuit matrix characteristics
- ordering methods considered
  - unsymmetric or symmetric pre-orderings
  - permutation to block-triangular form
- factorization methods considered
  - Kundert’s Sparse 1.3 (right-looking, Markowitz-style)
  - UMFPACK v4.3 (right-looking multifrontal)
  - Gilbert & Peierls’ left-looking sparse LU
  - SuperLU (supernodal left-looking sparse LU)
- KLU: sparse LU for circuit matrices (K: “Clark Kent”)
- experimental results
- parallel algorithm (future work)

Overview

KLU: An effective sparse matrix factorization algorithm for circuit matrices

- block triangular form
- approximate minimum degree (AMD)
- our implementation of Gilbert/Peierls' method
- very little fill-in for circuit matrices
- up to $\sim$1000 times faster than Kundert's Sparse 1.3
- faster than methods that use the BLAS

Sparse circuit matrix characteristics

- zero-free or nearly zero-free diagonal
- unsymmetric values, pattern roughly symmetric
- permutable to block triangular form (BTF)
- nonzero pattern of each block typically more symmetric than whole matrix (peculiar to circuit matrices)
- often have a few dense rows/columns - removed by BTF
- very, very sparse (BLAS not appropriate)
- if ordered properly: LU factors remain sparse (peculiar to circuit matrices)
- if ordered with typical strategies: can experience high fill-in

Ordering methods : BTF

Unsymmetric permutation to upper block triangular form.

- MATLAB `dmperm`, Duff and Reid's MC21+MC13, ...
- step 1: unsymmetric permutation for a zero-free diagonal (a maximal matching graph problem)
- step 2: symmetric permutation to block triangular form (find the strongly-connected components of a graph)
- factor/solve block k, block back substitution, factor/solve block $k-1$, ... No fill-in in off-diagonal blocks.

$$\mathbf{P}_b\mathbf{A}\mathbf{Q}_b = \begin{bmatrix} \mathbf{A}_{11} & \dots & \mathbf{A}_{1,k-1} & \mathbf{A}_{1k} \\ 0 & \dots & \dots & \dots \\ 0 & 0 & \mathbf{A}_{k-1,k-1} & \mathbf{A}_{k-1,k} \\ 0 & 0 & 0 & \mathbf{A}_{kk} \end{bmatrix}$$

Ordering methods : AMD

AMD finds $\mathbf{Q}$ to reduce fill-in for the Cholesky factorization of $\mathbf{Q}^\mathsf{T}\mathbf{AQ}$ (or of $\mathbf{Q}^\mathsf{T}(\mathbf{A}+\mathbf{A}^\mathsf{T})\mathbf{Q}$ if $\mathbf{A}$ unsymmetric). Limits fill-in for LU of $\mathbf{Q}^\mathsf{T}\mathbf{AQ}$ assuming no numerical pivoting.

- pre-ordering: find $\mathbf{Q}$ to control fill-in of Cholesky of $\mathbf{Q}^\mathsf{T}(\mathbf{A}+\mathbf{A}^\mathsf{T})\mathbf{Q}$
- numerical factorization: try to select $\mathbf{P}=\mathbf{Q}$ (constrained partial pivoting)
- Result: $\mathbf{PAQ}=\mathbf{LU}$
- AMD reduces an *optimistic* estimate of fill-in.
- other possibilities: nested dissection (recursive node separators) of $\mathbf{A}+\mathbf{A}^\mathsf{T}$
- unsuitable for circuit matrices *unless* applied to blocks of BTF form

Factorization methods : Kundert's Sparse 1

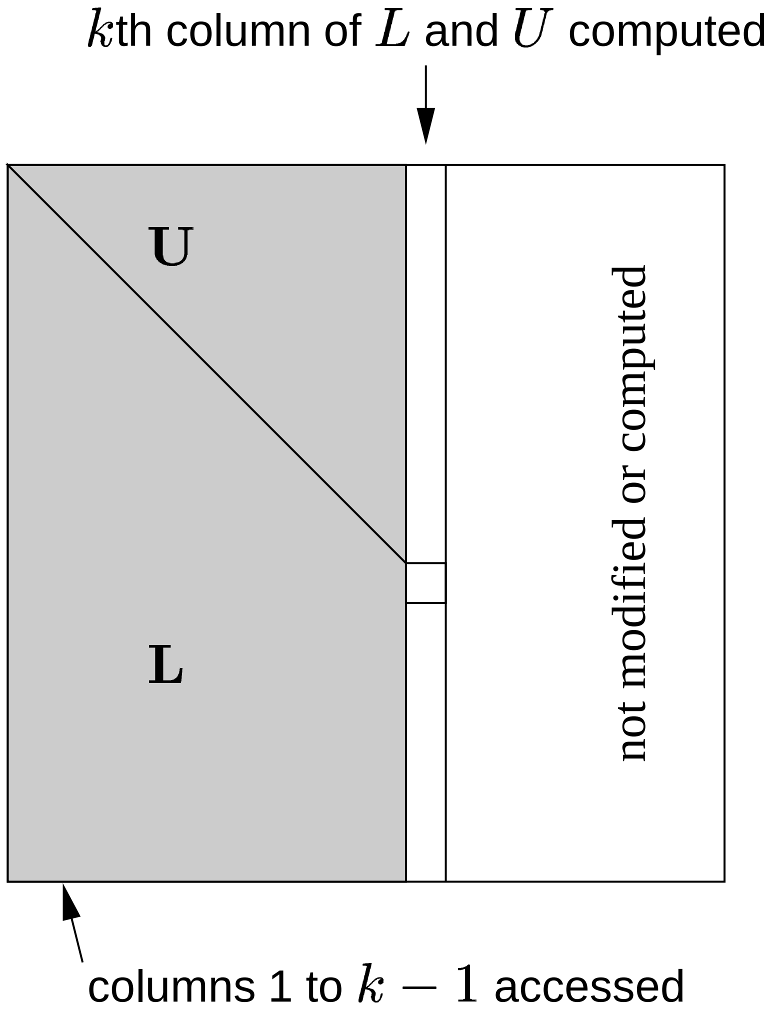
- First factorization
  - single-pass sparse Markowitz ordering + factorization
  - attempts to select pivots on diagonal
- Subsequent factorization
  - no changes to pivot ordering allowed
- no BTF pre-ordering
- low fill-in for circuit matrices, *unless* highly reducible to BTF
- very slow for large matrices, even when fill-in is low
- refactorization cannot modify pivot order

Factorization methods : Gilbert/Peierls

- left-looking. kth stage computes kth column of $\mathbf{L}$ and $\mathbf{U}$

```
L = I
U = I
for k = 1:n
    s = L \ A(:,k)
    (partial pivoting on s)
    U(1:k,k) = s(1:k)
    L(k:n,k) = s(k:n) / U(k,k)
end
```

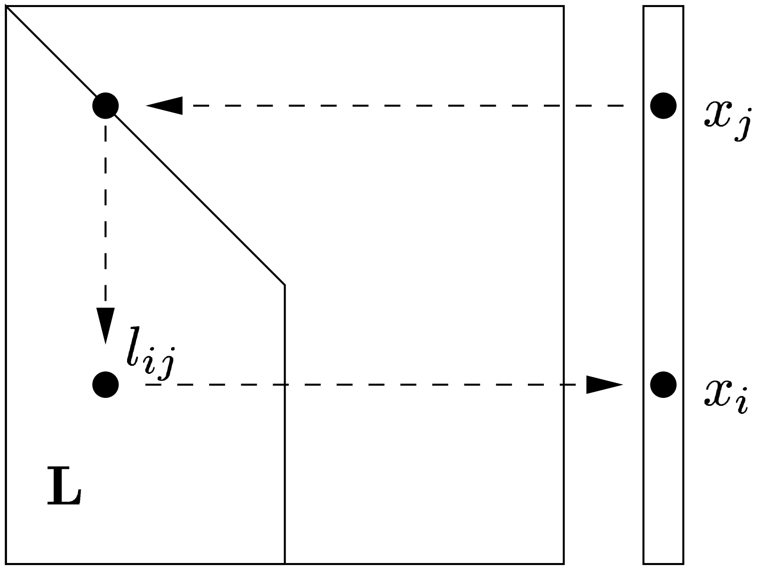
- key step: solve $\mathbf{L}\mathbf{x} = \mathbf{b}$, where $\mathbf{L}$ $\mathbf{x}$, and $\mathbf{b}$ are sparse


kth column of L and U computed
$\mathbf{U}$
$\mathbf{L}$
not modified or computed
columns 1 to $k-1$ accessed

Factorization methods : Gilbert/Peierls

Solving $\mathbf{Lx} = \mathbf{b}$, need nonzero pattern of $\mathbf{x}$ to compute $\mathbf{x}$

```
x = b
for j = 1:n
    if (x(j) ~= 0)
        x(j+1:n) = x(j+1:n) - ...
             L (j+1:n,j) * x(j)
    end
end
```



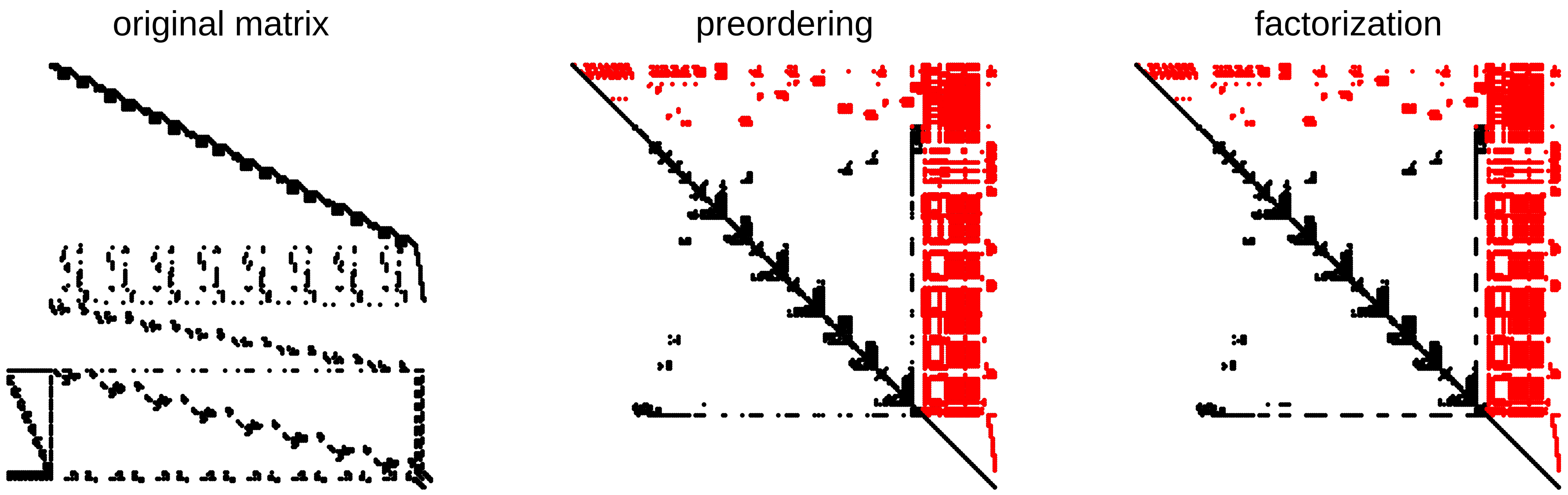
- $b_i \neq 0 \rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \rightarrow x_i \neq 0$

- let $G(L)$ have an edge $j \rightarrow i$ if $l_{ij} \neq 0$
- let $\mathcal{B} = \{i \,|\, b_i \neq 0\}$ and $\mathcal{X} = \{i \,|\, x_i \neq 0\}$
- then $\mathcal{X} = \text{Reach}_{G(L)}(\mathcal{B})$

KLU: a “Clark Kent” LU

- Gilbert/Peierl’s: used with unsymmetric preordering in MATLAB, so unsuitable for circuits
- SuperLU: Gilbert/Peierls’ algorithm, with supercolumns. Exploits the BLAS, but few supercolumns in circuit matrices
- KLU: a “Clark Kent” LU (what SuperLU was before it became Super, but also including our preordering strategies, which are not in SuperLU)
  - BTF ordering
  - AMD ordering of each block
  - Gilbert/Peierls’ LU factorization of each block
  - partial pivoting with diagonal preference
  - refactorization can allow for partial pivoting

Results: Grund/meg1, n: 2902, nz: 58,142

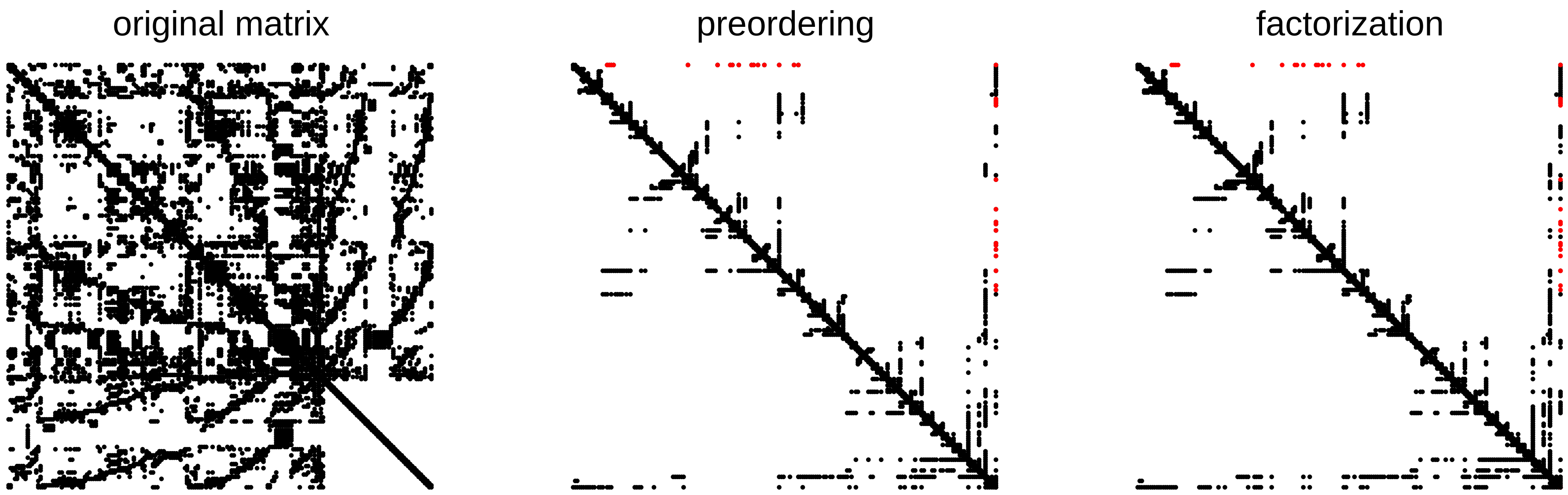


- largest block: 1906-by-1906
- total # nonzeros in diagonal blocks: 27,575
- one large block, the rest 1-by-1. BTF is very important

Results: Grund/meg1, n: 2902, nz: 58,142

method	factorization time (sec)			nz L+U	# flop
	1st	2nd,piv	no piv.	(10^3)	(10^6)
KLU	.02	.01	0.005	73.8	0.7
with BTF:					
MATLAB,GP+AMD	.06	.02	-	73.8	0.7
SuperLU+AMD	.03	.02	-	107.6	1.7
UMFPACK	.06	.04	-	77.1	0.7
Sparse 1.3	1.24	-	.10	84.9	2.1
no BTF:					
MATLAB,GP+AMD	.76	.65	-	522.7	61.9
SuperLU+AMD	.40	.32	-	518.4	61.9
UMFPACK	.06	.04	-	80.4	1.0
Sparse 1.3	24.28	-	1.49	199.1	16.7

Results: Hamm/scircuit, n: 170,988, nz: 958,936

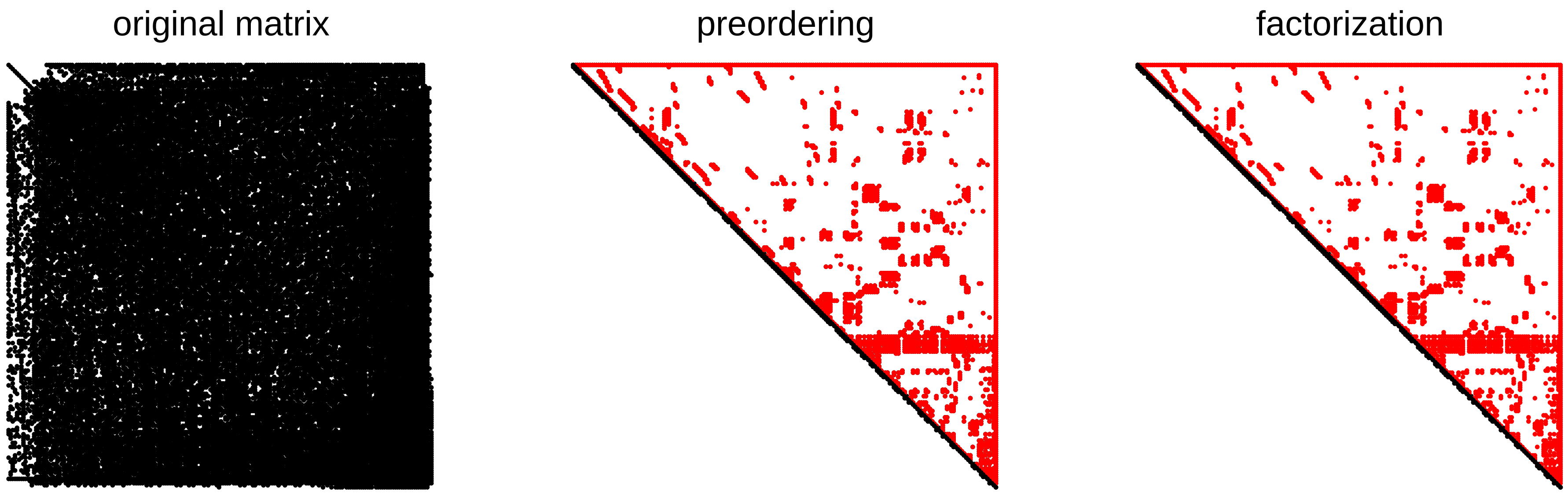


- largest block: 170,493-by-170,493
- total # nonzeros in diagonal blocks: 958,288
- essentially a single block, BTF has little effect

Results: Hamm/scircuit, n: 170,988, nz: 958,936

method	factorization time (sec)			nz L+U	# flop
	1st	2nd,piv	no piv.	(10^3)	(10^6)
KLU	1.67	.66	.35	2518	65.7
with BTF:					
MATLAB,GP+AMD	3.05	1.56	-	2518	65.7
UMFPACK	5.16	3.16	-	3734	214.0
Sparse 1.3	175.49	-	4.76	2453	60.9
no BTF:					
MATLAB,GP+AMD	2.93	1.58	-	2516	65.3
UMFPACK	5.12	3.16	-	3734	214.0
Sparse 1.3	207.40	-	5.01	2478	64.3

Results: Sandia/mult_dcop_03 , n: 25187, nz: 193,216



- largest block: 70-by-70
- total # nonzeros in diagonal blocks: 80,038
- many small blocks, BTF is very important

Results: Sandia/mult_dcop_03 , n: 25187, nz: 193,216

method	factorization time (sec)			nz L+U	# flop
	1st	2nd,piv	no piv.	(10^3)	(10^6)
KLU	.54	.06	.03	204	.17
with BTF:					
MATLAB,GP+AMD	.82	.24	-	202	.16
SuperLU+AMD	1.03	.29	-	202	.16
UMFPACK	1.19	.46	-	201	.16
Sparse 1.3	.56	-	.03	202	.16
no BTF:					
MATLAB,GP+AMD	.80	.42	-	446	4.81
SuperLU+AMD	.76	.51	-	445	4.83
UMFPACK	.73	.42	-	372	2.56
Sparse 1.3	131.22	-	22.29	294	.90

Results: Acar matrix , n: 11,341, nz: 97,193

- From Emrah Acar, IBM Research, Austin TX.
- “UMFPACK v4.0 failed, MA28 and Sparse 1.3 slow, UMFPACK V4.3 not tested”
- transient circuit, highly coupled
- UMFPACK v4.3 added an automatic pivot strategy (AMD is selected for circuits) and singleton removal that is far better than v4.0 for circuits All results in this presenation are for UMFPACK V4.3.
- one large block, 48 singletons (1-by-1 blocks): typical for transient analysis

Results: Acar matrix , n: 11341, nz: 97,193

method	factorization time (sec)			nz L+U	# flop
	1st	2nd,piv	no piv.	(10^3)	(10^6)
KLU	.23	.16	.12	362	2.37
with BTF:					
MATLAB,GP+AMD	.39	.27	-	362	2.37
UMFPACK	.33	.22	-	362	2.37
Sparse 1.3	16.83	-	2.48	396	3.17
no BTF:					
MATLAB,GP+AMD	.38	.28	-	368	2.43
UMFPACK	.31	.22	-	362	2.37
Sparse 1.3	17.14	-	2.34	392	3.00

Parallel algorithm (future work)

- BTF pre-ordering
- for each diagonal block:
  - coarse-grain nested dissection
  - pre-order with a constrained AMD
  - apply sequential sparse left-looking algorithm on each leaf
  - factorizing separators, three options:
    - right-looking: construct Schur complement and send to parent
    - left-looking: pass local $\mathbf{L}$ to parent
    - left-looking: parallel sparse triangular solve

Summary

KLU: An effective sparse matrix factorization algorithm for circuit matrices

- block triangular form
- approximate minimum degree (AMD)
- our implementation of Gilbert/Peierls' method
  - symmetric pruning
  - non-recursive depth-first-search
  - simple: primary kernel only 650 lines of code
  - faster than Gilbert/Peierls (MATLAB [L,U,P]=lu(B))
- up to ∼1000 times faster than Kundert's Sparse 1.3
- up to ∼100 times faster than BTF + Sparse 1.3
- faster than methods that use the BLAS

(additional slides)

- Factorization methods : UMFPACK
- Ordering methods : COLAMD

Factorization methods : UMFPACK

- Pre-ordering
  - automatically selects a symmetric strategy (AMD, possibly with a pre-ordering to decrease the number of zeros on the diagonal), or an unsymmetric strategy (COLAMD).
  - usually selects symmetric strategy for circuits
  - no true BTF preordering (finds 1-by-1 blocks only)
- Numerical factorization
  - right-looking, outer-product, multifrontal method
  - uses the BLAS (unsuitable for circuit matrices)
  - allows for partial pivoting

Ordering methods : COLAMD

COLAMD finds a column ordering $\mathbf{Q}$ to reduce fill-in for the Cholesky factorization of $(\mathbf{AQ})^\mathsf{T}\mathbf{AQ}$. Limits the worst-case fill-in for LU of $\mathbf{AQ}$.

- pre-ordering: find $\mathbf{Q}$ to control fill-in with worst-case $\mathbf{P}$
- numerical factorization: select $\mathbf{P}$ via partial pivoting. Diagonal not treated specially.
- Result: $\mathbf{PAQ} = \mathbf{LU}$
- COLAMD reduces a *pessimistic* upper-bound on fill-in.
- other possibilities: nested dissection (recursive node separators) of $\mathbf{A}^\mathsf{T}\mathbf{A}$
- unsuitable for circuit matrices