Department of Mathematics
CHEMNITZ UNIVERSITY OF TECHNOLOGY

Single-Pattern-Multi-Value LU Decomposition
Basic Ideas and Parallelization

Martin Köhler
joint work with Peter Benner and Jens Saak

Mathematics in Industry and Technology
Chemnitz University of Technology

Facing the Multicore-Challenge
Heidelberg Academy of Sciences

March 19, 2010

Outline

Lyapunov Equation

$$FX + XF^T = -GG^T \tag{1}$$

with $F \in \mathbb{R}^{n\times n}$, $G \in \mathbb{R}^{n\times p}$ and unknown $X \in \mathbb{R}^{n\times n}$, $X = X^T > 0$

Arises in:

- Optimal Control
- Model Order Reduction
- a Newton-step for Algebraic Ricatti Equations

$$R(X) = Q + A^T X + XA + XGX = 0$$

Solution methods:

- dense matrices: Bartel-Stewart alg., Hammarling's method, Sign-Function $\mathcal{O}(n^3)$
- sparse matrices: **Alternating-Directions-Implicit iteration** $\mathcal{O}(\mathrm{nnz}(F))$

Alternating Directions Implicit Iteration

Consider the LRCF-ADI algorithm to solve $FX + XF^T = -GG^T$:

Algorithm 1 Low-rank Cholesky factor ADI iteration (LRCF-ADI)

Input: F, G defining $FX + XF^T = -GG^T$ and
shift parameters $\{p_1, \ldots, p_{imax}\}$

Output: $Z = Z_{imax} \in \mathbb{C}^{n \times t_{imax}}$, such that $ZZ^H \approx X$

1: **Solve** $(A + p_1 I)V_1 = \sqrt{-2\,\mathrm{Re}\,(p_1)}G$ **for** V_1
2: $Z_1 = V_1$
3: **for** $i = 2, 3, \ldots, i_{max}$ **do**
4: **Solve** $(A + p_i I)\tilde{V} = (V_{i-1})$ **for** $\tilde{V}$
5: $V_i = \sqrt{\mathrm{Re}\,(p_i)/\,\mathrm{Re}\,(p_{i-1})}(V_{i-1} - (p_i + \overline{p_{i-1}})\tilde{V})$
6: $Z_i = [Z_{i-1} \;\; V_i]$
7: **end for**

Problem - Memory Usage

We need p_{imax} decompositions and have to hold them in memory. In case of a simple FDM semi-discretized PDE problem[1], we get with $p_{imax} = 16$

N	size of L+U in MB	16 LUs in MB
100	0.02	0.35
2 500	1.16	18.59
10 000	6.45	103.20
40 000	33.62	537.92
90 000	90.75	**1 452.00**
250 000	285.10	**4 561.30**
562 500	718.00	**11 488.00**
1 000 000	**1 379.00**	**22 064.00**

[1] instationary convection-diffusion equation on the unit square with homogeneous 1st kind boundary conditions

Problem - Memory Usage

We need p_{imax} decompositions and have to hold them in memory. In case of a simple FDM semi-discretized PDE problem[1], we get with $p_{imax} = 16$

N	size of L+U in MB	16 LUs in MB
100	0.02	0.35
2 500	1.16	18.59
10 000	6.45	103.20
40 000	33.62	537.92
90 000	90.75	**1 452.00**
250 000	285.10	**4 561.30**
562 500	718.00	**11 488.00**
1 000 000	**1 379.00**	**22 064.00**

impracticable on non HPC machines

[1] instationary convection-diffusion equation on the unit square with homogeneous 1st kind boundary conditions

Single-Pattern-Multi-Value Idea

Preparation

If we compute a LU factorization of a matrix A, we know

- the pattern of L and U including the number of non-zero entries
- sizes and values of all data structures

Remark: Numerically zero entries must not be rejected in the pattern.

Definition

Let $A \in \mathbb{R}^{n \times m}$ be a matrix. We call the set

$$\mathcal{P}(A) = \{(i,j) \mid A_{i,j} \neq 0\}$$

pattern of A. Furthermore we define

$$\mathcal{P}_R(A,i) = \{j \mid A_{i,j} \neq 0\}$$

as the **pattern of the i-th row** of A.

Single-Pattern-Multi-Value Idea

Preparation

We want to compute $\tilde{L}\tilde{U} = A + pI$ with knowledge of $LU = A$.

If $\mathcal{P}(A + pI) = \mathcal{P}(A)$ holds[2] and $\tilde{L}\tilde{U} = A + pI$:

- $\mathcal{P}(\tilde{L}) = \mathcal{P}(L)$ and $\mathcal{P}(\tilde{U}) = \mathcal{P}(U)$
- want to use $\mathcal{P}(L)$ and $\mathcal{P}(U)$ to compute $\tilde{L}\tilde{U} = A + pI$
- allocate all required memory in one step

[2] in our case: $A(i,i) \neq 0 \quad \forall i$

Single-Pattern-Multi-Value Idea

Preparation

We want to compute $\tilde{L}\tilde{U} = A + pI$ with knowledge of $LU = A$.

If $\mathcal{P}(A + pI) = \mathcal{P}(A)$ holds[2] and $\tilde{L}\tilde{U} = A + pI$:

- $\mathcal{P}(\tilde{L}) = \mathcal{P}(L)$ and $\mathcal{P}(\tilde{U}) = \mathcal{P}(U)$
- want to use $\mathcal{P}(L)$ and $\mathcal{P}(U)$ to compute $\tilde{L}\tilde{U} = A + pI$
- allocate all required memory in one step

Realization Idea

Reuse $\mathcal{P}(L)$ and $\mathcal{P}(U)$ in a row-wise LU decomposition of $A + pI$.

[2] in our case: $A(i,i) \neq 0 \quad \forall i$

Single-Pattern-Multi-Value Idea

Resulting Algorithm

Algorithm 2 Pattern-Reuse for $\tilde{L}\tilde{U} = \tilde{A}$

Input: $\tilde{A} := A + pI$, $\mathcal{P}(L)$ and $\mathcal{P}(U)$ with $LU = A$ and $\mathcal{P}(A) = \mathcal{P}(\tilde{A})$
Output: $\tilde{L}$, $\tilde{U}$ with $\tilde{L}\tilde{U} = \tilde{A}$

$\tilde{U}(1,:) = \tilde{A}(1,:)$
for $i = 2, \dots, n$ **do**
  $w = \tilde{A}(i,:)$ as sparse vector
  for all $j \in \mathcal{P}_R(L,i)$ **ordered do**
    $\tilde{L}(i,j) = \alpha = w(j)/\tilde{U}(j,j)$
    $w = w - \alpha \cdot \tilde{U}(j,:)$
  end for
  for all $j \in \mathcal{P}_R(U,i)$ **do**
    $\tilde{U}(i,j) = w(j)$
  end for
end for

Single-Pattern-Multi-Value Idea

“Single-Pattern-Multi-Value” Idea

Another way to reuse information from $\mathcal{P}(L)$ and $\mathcal{P}(U)$.

- the L and U pattern of all system $A + p_i I$ is the same
- only necessary to store them once
- read-only access on $\mathcal{P}(L)$ and $\mathcal{P}(U)$ $\rightarrow$ no problems with race conditions
- use multicore CPUs: compute $L_i U_i = A + p_i I$ in parallel for different p_i with Algorithm 2 ($\rightarrow$ OpenMP)

Single-Pattern-Multi-Value Idea

"Single-Pattern-Multi-Value" Idea

Another way to reuse information from $\mathcal{P}(L)$ and $\mathcal{P}(U)$.

- the L and U pattern of all system $A + p_i I$ is the same
- only necessary to store them once
- read-only access on $\mathcal{P}(L)$ and $\mathcal{P}(U)$ $\rightarrow$ no problems with race conditions
- use multicore CPUs: compute $L_i U_i = A + p_i I$ in parallel for different p_i with Algorithm 2 ($\rightarrow$ OpenMP)

$\rightarrow$ reduce the memory usage drastically

$\rightarrow$ use modern CPUs more efficiently

Numerical Results

Pattern-Reuse

Factorize the FDM matrix A from the example problem on an Intel®Xeon®5160. Computation times in seconds.

dimension	LU with CSparse	LU with known $\mathcal{P}(L)$, $\mathcal{P}(U)$	savings
10 000	0.06	0.02	57.0%
90 000	1.90	1.17	38.2%
250 000	9.55	7.57	20.7%
1 000 000	92.70	81.30	12.3%

Numerical Results

Memory Saving

Memory usage for $A + p_i I$ with $imax = 16$ on a 64bit machine:

N	size of L+U in MB	16 LUs in MB	SPMV[3] LU	savings
10 000	6.45	103.20	53.68	47.99%
90 000	90.75	1 452.00	760.91	47.60%
160 000	175.28	2 804.50	**1 471.50**	47.53%
250 000	285.10	4 561.30	**2 394.50**	47.50%
562 500	718.00	11 488.00	**6 038.00**	47.44%
1 000 000	1 379.00	22 064.00	**11 604.00**	47.41%

[3] single-pattern-multi-value

Numerical Results

Overall Results

We solve the Lyapunov-Equation arising from Problem 1 on an Intel®Xeon®5160 CPU, 16 GB RAM. With our implementation and MATLAB®.

N	LyaPack[4]	M.E.S.S.[5]	C.M.E.S.S.[6]
625	0.10	0.23	0.04
10 000	6.22	5.64	0.97
40 000	71.48	34.55	11.09
90 000	418.50	90.49	34.67
160 000	out of mem.	219.90	109.32
250 000	out of mem.	403.80	193.67
562 500	out of mem.	1 216.70	930.14
1 000 000	out of mem.	2 428.60	2 219.95

[4] current MATLAB toolbox
[5] upcoming MATLAB toolbox
[6] without CSparse → slower first decomposition

Numerical Results

Conclusions

- minimize the memory allocation effort (**no** reallocation needed)
- speedup depends on the cache size of the cpu, the `(re)malloc` implementation, the memory architecture and the matrix size
- data is read continuously from memory
- **reuse** of the pattern structure **can accelerate** factorizations **significantly** and reduce the memory usage
- memory bandwidth is the bottle neck for many cores

Outlook

Open Problems

- UMFPack (unsymmetric multifrontal LU) is faster than the reuse but requires more memory
  - → check if the reuse idea can be ported to UMFPack
  - → port the memory saving idea to UMFPack
  - → seems to be not thread safe
- MATLAB-interface with OpenMP support nearly impossible because of conflicting linker/compiler flags some older versions of gcc:
  `XLDFLAGS="$XLDFLAGS -Wl,-z,nodlopen"`
  or MATLAB crashes immediately
- shared memory parallel algorithms for sparse matrices

Outlook

Current Implementation: C.M.E.S.S.

C.M.E.S.S. is:

- upcoming C library for solving large scale matrix equations
- providing a uniform interface for iterative and direct linear system solvers
- supporting OpenMP where it is possible
- a front end for UMFPack, LAPACK, RRQR, CSparse, SLICOT,. . .
- dynamically converting between various sparse storage support
- handling sparse and dense matrices in a unified way

See our C.M.E.S.S. poster as well.

Outlook

Current Implementation: C.M.E.S.S.

C.M.E.S.S. is:

- upcoming C library for solving large scale matrix equations
- providing a uniform interface for iterative and direct linear system solvers
- supporting OpenMP where it is possible
- a front end for UMFPack, LAPACK, RRQR, CSparse, SLICOT,. . .
- dynamically converting between various sparse storage support
- handling sparse and dense matrices in a unified way

Thanks for your attention.