PARALLEL LOW RANK LU AND CHOLESKY REFACTORIZATION

Lung-Sheng Chien, NVIDIA
lchien@nvidia.com
OUTLINE

- What is low rank refactorization
- Formulation
- Experiment
- Conclusions
**Rank-1 LU refactorization**

- Initial LU factorization, \( A=LU \)
  \[
  A = \begin{pmatrix}
  21 & 2 & 3 & 4 \\
  5 & 26 & 7 & 8 \\
  9 & 10 & 31 & 12 \\
  13 & 14 & 15 & 36
  \end{pmatrix}
  = \begin{pmatrix}
  1.000 & & & \\
  0.238 & 1.000 & & \\
  0.429 & 0.358 & 1.000 & \\
  0.619 & 0.5 & 0.364 & 1
  \end{pmatrix}
  \begin{pmatrix}
  21.00 & 2.000 & 3.000 & 4.000 \\
  & & & \\
  25.52 & 6.286 & 7.048 & \\
  & & & \\
  27.46 & 7.761 & & \\
  & & & \\
  27.17 & & & 
  \end{pmatrix}
  \]

- Augment rank-1 product to \( A \)
  \[
  A_{\text{new}} = \begin{pmatrix}
  21 & 2 & 3 & 4 \\
  5 & 26 & 3 & 8 \\
  9 & 10 & 31 & 12 \\
  13 & 14 & 12 & 36
  \end{pmatrix}
  = \begin{pmatrix}
  21 & 2 & 3 & 4 \\
  5 & 26 & 7 & 8 \\
  9 & 10 & 31 & 12 \\
  13 & 14 & 15 & 36
  \end{pmatrix}
  + \begin{pmatrix}
  0 & & & \\
  -4 & & & \\
  0 & & & \\
  -3 & & & 
  \end{pmatrix}
  \begin{pmatrix}
  0 & 0 & 1 & 0
  \end{pmatrix}
  = A + xy^T
  \]

Redo LU or fast refactorization?
Rank-k LU refactorization

\[ A_{\text{new}} = A + \sum (dA)(; , j) \cdot e_j^T \]

- \( A_{\text{new}} = A + dA \)
- \( A_{\text{new}} \approx A + \text{svd}(dA) \)

\[ A_{\text{new}} = A + \sum_{j=1}^{k} \sigma_j x_j y_j \]

Our focus
Applications

**CIRCUIT SIMULATION**

- For \( M_1 \):
  - \( I_{31} = f_1(V_3 - V_2) \)
  - \( I_{21} = f_2(V_2 - V_3) \)
  - \( I_{32} = f_3(V_3 - V_2) \)

- For \( M_2 \):
  - \( I_{21} = f_4(V_2 - V_3) \)
  - \( I_{10} = f_5(V_1) \)
  - \( I_{20} = f_6(V_2) \)

**STRUCTURAL MECHANICS**

**LINEAR PROGRAMMING**

- [www.cs.duke.edu](http://www.cs.duke.edu)

**STRUCTURE FROM MOTION**

- [www.theia-sfm.org](http://www.theia-sfm.org)
Limitation of low rank refactorization

- $A_{new}$ reuses pivoting of $A$
  \[ PA = LU \]
  \[ A_{new} = A + \alpha xy^T \]
  \[ PA_{new} = LU + \alpha (Px)y^T = L_{new}U_{new} \]

- Numerical instability is possible

- Needs a $O(n^2)$ post-processing to check $\|L_{new}\|$ and $\|U_{new}\|$
OUTLINE

- What is low rank refactorization
- Formulation
- Experiment
- Conclusions
Rank-1 LU formula

Given LU factorization of \( A \)
\[
A = \begin{pmatrix} 1 & l_{21} \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12}^T \\ U_{22} \end{pmatrix}
\]
and vectors
\[
x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}
\]
refactorize \( A_{\text{new}} = A + \alpha xy^T \)

\[
A_{\text{new}} = \begin{pmatrix} \frac{1}{l_{21}} & 1 \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12}^T \\ u_{12} & B \end{pmatrix}
\]

\[
\begin{align*}
\bar{u}_{11} &= u_{11} + \alpha x_1 y_1 \\
\bar{x}_2 &= x_2 - x_1 l_{21} \\
\bar{y}_2 &= y_2 - \frac{y_1}{u_{11}} u_{12} \\
\bar{l}_{21} &= l_{21} + \left( \alpha \frac{y_1}{u_{11}} \right) \bar{x}_2 \\
\bar{u}_{12} &= \left( \frac{u_{11}}{u_{11}} \right) u_{12} + \left( \alpha \frac{u_{11}}{u_{11}} \right) \bar{x}_2 \cdot \bar{y}_2 \\
B &= L_{22} U_{22} + \left( \alpha \frac{u_{11}}{u_{11}} \right) \bar{x}_2 \cdot \bar{y}_2
\end{align*}
\]
Schur complement \( B \) has the same form as \( A \) except different alpha.
Pros and cons of rank-1 LU

- No need to compute Schur complement numerically, so the complexity is $O(n^2)$, instead of $O(n^3)$
  Complexity for rank-1: $(2n^2 + 2n)$
  Complexity for rank-k: $(2n^2 + 2n)k$

- All operations are BLAS-1, not good for GPU except for super large matrix
  K40 has 16 SMs, 2048 threads each. To full utilize a GPU, it needs 32,000 threads, i.e. dimension of the matrix must be at least 32,000
Rank-k LU formula

Given LU factorization of \( A \)

\[
A = \begin{pmatrix}
1 & \frac{u_{11}}{l_{21}} \\
l_{21} & L_{22}
\end{pmatrix}
\begin{pmatrix}
\frac{u_{11}}{U_{22}} & u_{12}^T \\
\end{pmatrix}
\]

and rank-k matrices

\[
X = \begin{pmatrix}
x_1^T \\
X_2
\end{pmatrix},
Y = \begin{pmatrix}
y_1^T \\
Y_2
\end{pmatrix}
\]

refactorize \( A_{new} = A + XY^T \)

\[
A_{new} = \begin{pmatrix}
\frac{1}{l_{21}} & I
\end{pmatrix}
\begin{pmatrix}
\frac{u_{11}}{u_{12}^T} \\
B
\end{pmatrix}
\]

\[
\begin{align*}
\bar{u}_{11} &= u_{11} + x_1^T y_1 & \text{dot, BLAS-1} \\
\bar{y}_1 &= \frac{y_1}{u_{11}} & \text{scal, BLAS-1} \\
\bar{X}_2 &= X_2 - l_{21} x_1^T & \text{ger, BLAS-2} \\
\bar{l}_{21} &= l_{21} + X_2 \cdot \bar{y}_1 & \text{gemv, BLAS-2} \\
\bar{u}_{12} &= u_{12} + Y_2 \cdot x_1 & \text{gemv, BLAS-2} \\
\bar{Y}_2 &= Y_2 - \bar{u}_{12} \cdot \bar{y}_1^T & \text{ger, BLAS-2} \\
B &= L_{22} U_{22} + \bar{X}_2 \bar{Y}_2^T 
\end{align*}
\]
Rank-1 versus rank-k

\[ \bar{u}_{11} = u_{11} + \alpha x_1 y_1 \]

\[ \bar{u}_{11} = u_{11} + x_1^T y_1 \]

BLAS-1
**Rank-1 versus rank-k**

- **BLAS-1**
  - $\overline{u_{11}} = u_{11} + ax_1y_1$
  - $\overline{x_2} = x_2 - x_1l_{21}$
  - $\overline{l_{21}} = l_{21} + \left(\alpha \frac{y_1}{u_{11}}\right)\overline{x_2}$

- **BLAS-2**
  - $\overline{u_{11}} = u_{11} + x_1^Ty_1$
  - $\overline{y_1} = \frac{y_1}{u_{11}}$
  - $\overline{X_2} = X_2 - l_{21}x_1^T$
  - $\overline{l_{21}} = l_{21} + \overline{X_2} \cdot \overline{y_1}$
Rank-1 versus rank-k

\[
\begin{align*}
\bar{u}_{11} &= u_{11} + \alpha x_1 y_1 \\
\bar{x}_2 &= x_2 - x_1 l_{21} & \text{BLAS-1} \\
\bar{l}_{21} &= l_{21} + \left( \alpha \frac{y_1}{u_{11}} \right) \bar{x}_2 & \text{BLAS-1} \\
\bar{y}_2 &= y_2 - \frac{y_1}{u_{11}} u_{12} & \text{BLAS-1} \\
\bar{u}_{12} &= \left( \frac{u_{11}}{u_{11}} \right) u_{12} + (\alpha x_1) \bar{y}_2 & \text{BLAS-1}
\end{align*}
\]

\[
\begin{align*}
\bar{u}_{11} &= u_{11} + x_1^T y_1 & \text{BLAS-1} \\
\bar{y}_1 &= \frac{y_1}{u_{11}} & \text{BLAS-1} \\
\bar{X}_2 &= X_2 - l_{21} x_1^T & \text{BLAS-2} \\
\bar{l}_{21} &= l_{21} + \bar{X}_2 \cdot \bar{y}_1 & \text{BLAS-2} \\
\bar{u}_{12} &= u_{12} + Y_2 \cdot x_1 & \text{BLAS-2} \\
\bar{Y}_2 &= Y_2 - \bar{u}_{12} \cdot \bar{y}_1^T & \text{BLAS-2}
\end{align*}
\]
## Complexity analysis

- Traditional rank-k is rank-1 k times
- Refactorization is memory-bound, complexity of I/O is better than flops
- 25% less I/O on LU and 50% more I/O on Cholesky

<table>
<thead>
<tr>
<th></th>
<th>FLOPS</th>
<th>MEMORY I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU, rank-1 k times</td>
<td>$(2n^2k + 2nk)$</td>
<td>$(4n^2k)$</td>
</tr>
<tr>
<td>LU, ours rank-k</td>
<td>$2n^2k + n$</td>
<td>$n^2(3k + 3) + n(5k - 1)$</td>
</tr>
<tr>
<td>Cholesky, rank-1 k times</td>
<td>$(n^2k + 3nk)$</td>
<td>$(2n^2k + 3nk)$</td>
</tr>
<tr>
<td>Cholesky, ours rank-k</td>
<td>$(2n^2k + n(-k + 2))$</td>
<td>$n^2(3k + 3) + n(3k - 1)$</td>
</tr>
</tbody>
</table>
Pros and cons of rank-k LU

- BLAS-2 operations (gemv, ger) on n-by-k sub-matrices
- 25% less data transfer compared to rank-1 LU update k times if k is large. The saving comes from low cost for update of X and Y
- Need to tune the kernels (gemv, ger) for different configuration of dimension and rank
- less rounding error
OUTLINE

- What is low rank refactorization
- Formulation
- Experiment
- Conclusions
Experimental Setup

- CPU: Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz
- GPU: K40
- random generate dense matrices X and Y (not consider sparsity pattern to reduce the computation)
Advantage of BLAS-2

- For rank-k refactorization, gemv is $O(nk)$ and axpy is $O(n)$
- “shmoo plot” shows potential gain of BLAS-2 based on $k=128$. It ranges from 23 up to 80.
Rank-k LU (GPU) versus rank-1 LU (GPU) k times

- Compared to rank-1, rank-k is much better, ranging from 10x to 240x
- When n is large, the speedup converges to 20x
- The speedup comes from GPU under-utilization of BLAS-1
Rank-k LU (GPU) versus rank-1 LU (CPU) k times

- GPU needs rank 64 and above to outperform CPU single thread
- For large matrix and not small rank, GPU can be 10x faster than CPU (match bandwidth ratio)
Rank-k Chol (GPU) versus rank-1 Chol (GPU) \( k \) times

- Compared to rank-1, rank-k is much better, ranging from 10x to 231x
- When \( n \) is large, the speedup converges to 18x
Rank-k Chol (GPU) versus rank-1 Chol (CPU) \(k\) times

- GPU needs rank 64 and above to outperform CPU single thread
- For large matrix and not small rank, GPU can be 6x faster than CPU, this is worse than LU because of 50% overhead on I/O
Conclusions

- Rank-k factorization can be done efficiently by BLAS-2 operations
- Our rank-k LU is up to 10x faster than traditional rank-1 LU k times; Our rank-k Chol is up to 6x faster than traditional rank-1 Chol k times
- Although low rank LU does not do pivoting, we can check if refactorization succeeds or not by $O(n^2)$ operations
- So far, low rank LU is not appealing in dense LU, but it is a key for sparse LU.
- Future work: rank-k LU/Cholesky on sparse matrices
THANK YOU

JOIN THE CONVERSATION
#GTC16  

JOIN THE NVIDIA DEVELOPER PROGRAM AT developer.nvidia.com/join
Reference

