Characteristics

(operations) \( \Omega = \frac{n^2}{2} \)

(data) \( \delta = \frac{n^2}{2} + \frac{3n}{2} \)

(transfers) \( \Theta_{\min} = \frac{n^2}{2} + \frac{5n}{2} \)

(temporal locality) \( \mu_{\min} = \frac{1}{2} + \frac{5}{2n} \)

In the best case this is similar to the matrix-vector primitive. Very little data locality, i.e., the primitive runs at best at 2 operations per memory transfer and therefore its performance is determined by the bandwidth of the memory in which the data resides.
Vector or fine grain parallel algorithms

The inner loops of the standard sequential versions vectorize trivially yielding the row-sweep and column-sweep algorithms respectively. These two algorithms represent a decomposition into BLAS1 primitives

- the row sweep is a series of dotproducts, \( \mu_{row} \approx 1 \)
- the row sweep requires row accesses of the array
- the row sweep requires an efficient dotproduct
- the column sweep is a series of triads, \( \mu_{column} \approx 3/2 \)
- average vector length \( n/2 \)
- Not suitable for register-based vector machines
Local Memory or Register-based vector processors

Assume one memory port and vector registers of length $R$

Fact: If $n \leq R$ then $\mu_{\text{row}} = \mu_{\text{column}} = \mu_{\text{min}}$

This is clearly the case for these small systems. $f$ can be loaded once into a register and $x$ can be created in register and written once. Each row or column of $L$ can be read exactly once.

Some details about the use of a vector mask vs. manipulating the vector length must be considered when implementing the lower triangular solver via triads.

As done in the recitation notes, we can create a version of each routine by blocking and using these locally optimal primitives to achieve an algorithm that achieves near optimality overall.
Block Column Sweep

Let $L^{(0)} = L$, $f^{(0)} = f$, and let each of $L^{(j)}$, $x^{(j)}$, and $f^{(j)}$ be of order $(n - j\nu)$, $j = 0, \ldots, \frac{n}{\nu} - 1$ where

$$L^{(j)} = \begin{pmatrix} L^{(j)}_{11} & 0 \\ L^{(j)}_{21} & L^{(j)}_{22} \end{pmatrix}, L^{(j+1)} = L^{(j)}_{22},$$

$$x^{(j)} = \begin{pmatrix} x^{(j)}_1 \\ x^{(j)}_2 \end{pmatrix}, x^{(j+1)} = x^{(j)}_2, f^{(j)} = \begin{pmatrix} f^{(j)}_1 \\ f^{(j)}_2 \end{pmatrix}$$

with $L^{(j)}_{11}$, $x^{(j)}_1$, and $f^{(j)}_1$ being each of order $\nu$ (we assume that $\nu$ divides $n$). The block column-sweep algorithm may then be described as:

$$B_{\text{Col\_Sweep}} :$$

$\quad p = \frac{n}{\nu}$

$\quad$ do $j = 0, p - 2$

$\quad$ solve $L^{(j)}_{11} x^{(j)}_1 = f^{(j)}_1$ via Col\_Sweep or Row\_Sweep

$\quad$ $f^{(j+1)} = f^{(j)}_2 - L^{(j)}_{21} x^{(j)}_1$

$\quad$ end do

$\quad$ solve $L^{(p-1)} x^{(p-1)} = f^{(p-1)}$ via Col\_Sweep or Row\_Sweep.
Uses two primitives: a small triangular system solver and a matrix-vector product both with BLAS2 complexity on register-based vector processors.

If $\nu = 1$ the algorithm is the column sweep and if $\nu = n$ the algorithm is the column sweep or whatever is used to solve the smaller triangular systems.

It follows that

$$
\mu_{bcs} = \frac{1}{2} + \frac{1}{\nu} + \eta
$$

where $\eta$ is bounded in magnitude by $1/n$. 
Block Row Sweep

Partition $L$ so that each block row is of the form $[C_i, L_i, 0]$ where $C_i \in \mathbb{R}^{\nu \times (i-1)\nu}$ and $L_i \in \mathbb{R}^{\nu \times \nu}$. Let $x = (x_1^T, \ldots, x_p^T)^T$, $x^{(j)} = (x_1^T, \ldots, x_j^T)^T$, and $f = (f_1^T, \ldots, f_p^T)^T$, where $x_i, f_i \in \mathbb{R}^\nu$. The block algorithm is:

\[ B\_Row\_Sweep : \]
\[ p = \frac{n}{\nu} \]
\[ \text{solve } L_1 x_1 = b_1 \text{ via Col\_Sweep or Row\_Sweep} \]
\[ \text{do } j = 2, p \]
\[ f_j = f_j - C_j x^{(j-1)} \]
\[ \text{solve } L_j x_j = f_j \text{ via Col\_Sweep or Row\_Sweep} \]
\[ \text{end do.} \]

Uses the same two primitives as the block column sweep but the matrix-vector product operates on short, wide matrices rather than tall narrow matrices.

It follows that

\[ \mu_{brs} = \frac{1}{2} + \frac{1}{2\nu} + \eta \]

where $\eta$ is bounded in magnitude by $1/n$. 
Block Column Sweep Partitioning

\[ \begin{align*}
L & \quad x & = & f \\
\_ & \quad x & = & f \\
\_ & \quad x & = & f
\end{align*} \]

Block Row Sweep Partitioning

\[ \begin{align*}
L & \quad x & = & f \\
\_ & \quad x & = & f \\
\_ & \quad x & = & f
\end{align*} \]

\( x_i \in \mathbb{R}^n \) \hspace{1em} \( f_i \in \mathbb{R}^n \) for both partitionings
Parallel algorithms

Consider $L$ partitioned into submatrices $L_{ij} \in \mathbb{R}^{\nu \times \nu}$, $i,j = 1, \ldots, k = n/\nu$. The column and row oriented algorithms respect the data dependence graph seen below for $k = 6$.

The squares represent solve tasks $L_{ii}x_i = f_j$ and the circles represent matrix-vector tasks $f_i \leftarrow f_i - L_{ij}x_j$ (we will assume that this can be done indivisibly via a lock/unlock for now). We will refer to these as S tasks and M tasks respectively. Note that an S task is half the operations of an M task.
Consider what can happen in parallel for this task graph (operations at each level can be done in parallel)

Level 1  the S task for node (1,1)

Level 2  M tasks for nodes (5,1), (4,1), (3,1), and (2,1)

Level 3  the S task for node (2,2)

Level 4  M tasks for nodes (5,2), (4,2), and (3,2)

Level 5  the S task for node (3,3)

Level 6  M tasks for nodes (5,3), and (4,3)

Level 7  the S task for node (4,4)

Level 8  M tasks for nodes (5,4)

Level 9  the S task for node (5,5)

The maximum width is $k - 1$ therefore at most $k - 1$ processors are needed. The height of the task graph, the length of the critical path, is $k$ S tasks and $k - 1$ M tasks.
If $T_s$ and $T_m$ represent the time on one processor for an S and M task respectively we have a total time on $k - 1$ processors of

$$T_{k-1} = kT_s + (k - 1)T_m \approx \frac{3k}{2}T_m$$

where $T_s \approx T_m/2$ is used and a time on one processor of

$$T_1 \approx \frac{k^2}{2}T_m$$

which yields a speedup of $\sigma_{k-1} = k/3$ and an efficiency of $\epsilon_{k-1} \approx 1/3$. In the limit as $k \to n$ we have $\pi_{max} = n - 1$, $\pi_{ave} \approx n/3$, and $\Lambda = 3n$. Note that $\pi_{max}$ and $\pi_{ave}$ are computed under the assumption that the parallel algorithm does not have any redundant operations compared to the sequential algorithm. We will see later that we can go faster if we have more processors and we perform some redundant work.
Shared memory synchronization strategies

The block row sweep and block column sweep also provide the basic parallel shared memory versions of the solver. A simple total primitive approach with one processor performing a small triangular system solution followed by a barrier and then all processors cooperating on a matrix-vector product with all of its synchronization handled within the matrix vector.

The block size used for these algorithms must take into account the sequential bottleneck created by the single processor triangular solve. This must be considered at the same time blocking for register use is considered. It is possible, therefore, that the single processor triangular solve will be of one order $\nu_1$ and that it will be solved using a block row or column sweep with another register-length-based block size $\nu_2$.

More aggressive synchronization approaches are possible, e.g., protecting each subvector $f_i$ with a lock and update it indivisibly. Such strategies can remove some of the inefficiencies of the barrier-based approach.
The most aggressive in terms of adding local synchronizations adds dependences to the graph. It can be used on a number of architectures.

The following task graph is a retimed version of the triangular solve task graph shown earlier. Note that it has only nearest neighbor synchronization and does not violate any of the dependences of the original task graph. It could be used to create a ring implementation (with two level pipelining if desired), a shared memory doacross implementation (which needs a fixed dependence distance vector), or a two-dimensional systolic array which would be useful if multiple right-hand side vectors were to be used to define multiple systems.

The ripple of dependences in each row of matrix vector tasks can be replaced by synchronizations controlled by locks on pieces of the solution vector (1 per matrix block row)
The source of the $O(n)$ critical path for the standard algorithms is clear. The computation graph for the column sweep follows. Each node corresponds to $N_i$ multiplied by a vector, i.e., a triad. The row sweep graph is identical with $N_i$ replaced by $M_j$ and the nodes correspond to dotproducts.
Column Sweep Dependence Graph
Product form algorithm

The critical path can be reduced by applying associativity to the matrix operations. For example the column sweep operations could be regrouped for \( n = 8 \)

\[
(((N_7N_6)(N_5N_4))((N_3N_2)(N_1f))).
\]

Note that the height is now \( O(\log n) \) matrix operations. (We must still consider the complexity of each node compared to a triad or dotproduct.)

Define \( N(0,i) = N_i \) and \( f(0) = f \). Now recursively define \( N(j,i) = N(j - 1,2i + 1)N(j - 1,2i) \) and \( f(j) = N(j - 1,1)f(j - 1) \) for \( j = 0, \ldots, \log n \) and \( i = 1, \ldots, (n/2^j) - 1 \). The product form produces \( x = f(\log n) \) and has the following dependence graph.
The dependence graph has \( k = \log n \) levels.

There are \( 2^{k-j} - 1 \) matrices at level \( j \).

\( N(j, i) \) has \( q_j = 2^j \) nontrivial columns which start at column \( i2^j \).

In order to compute a matrix on level \( j \) in minimum time, inner products must be computed involving the \( q_{j-1} \) columns in two matrices from level \( j - 1 \) and their main diagonal. This requires \( j + 1 \) levels of scalar computation (recall a dotproduct of length \( l \) requires \( \log(l) + 1 \) scalar levels). Therefore,

\[
T = \sum_{j=1}^{k} 1 + \log q_j
\]

\[
= \sum_{j=1}^{k} j + 1
\]

\[
= \frac{k^2}{2} + \frac{3k}{2}
\]
• More operations are required than the sequential algorithm: approximately \( n^3/10 + O(n^2) \). This is an \( O(n) \) redundancy factor.

• The algorithm can achieve the \( O(\log^2 n) \) time if \( n^3/68 + O(n^2) \) processors are available.

• Due to the redundant computations the algorithm is less stable than the \( O(n^2) \) algorithms.

• This algorithm is of theoretical interest only for large dense systems but a modified form can be used for sparse \( L \). The associativity is applied in a way that keeps the sparsity of the partial inverse high.

• The algorithm is also of interest in vector spaces where the basic entity is very simple and therefore \( O(n^3) \) processors might not be unreasonable, e.g., when dealing with Boolean recurrences for circuits such as multipliers and adders. In that case a processor is essentially a logic gate.