Gerald Polland, American Math Monthly, March 2000

there were no other restaurant in town.

... eating at a McDonalds: you can find one anywhere, and the food will keep you going, but it would be sad if the palatability of the results, but experience evolutionarily brings satisfaction. Using Microsoft Windows is like Linix is like preparing your own meals from recipes in the Joy of Cooking: the effort involved initially exceeds Kosher: the believers would not live any other way, but they cannot eat with members of other religions. Using an Apple is like keeping different computer operating systems as being like different cuisines. Using an Apple is like keeping.
Notes 4

Page 2

with variables $x_1, \ldots, x_n$ and $q$, for which a common solution is sought.

$$1 - u^q = 1 - u_x \cdot u_{x'} = u_D + \cdots + x_{1} \cdot \cdot \cdot + u_{D} + 0 \cdot x_{1} \cdot \cdot \cdot + u_{D}$$

... 

$$1 = 1 \cdot u_x \cdot u_{x'} + \cdots + 1 \cdot u_{D} + 0 \cdot u_{D}$$

$$0 = 1 \cdot u_x \cdot u_{0} + \cdots + 1 \cdot u_{0} + 0 \cdot u_{0}$$

A system of linear equations also called just a linear system, is a collection of equations.

In which $a_0, a_1, \ldots, a_n$ are constants.

$$q = 1 \cdot u_x \cdot u_{D} + \cdots + 0 \cdot u_{D} + 1 \cdot u_{0} + 0 \cdot u_{0}$$

A linear equation in variables $x_0, x_1, \ldots, x_n$ is an equation $u_x \cdot \cdot \cdot \cdot u_{x'} = 0$.

Mathematically and computationally, we tend to try to convert other problems into linear problems. Because we can solve linear systems mathematically and computationally in efficient ways, systems computationally efficient ways.

First, we know mathematically how to solve linear systems, and we have worked out ways to solve linear systems using matrices and linear algebra. But two reasons...
A matrix is called **upper triangular** (lower triangular) if the only locations with nonzero entries lie on or above (on or below) the main diagonal.

A matrix is called **tridiagonal** if the only locations with nonzero entries lie on the main diagonal or on the two diagonals immediately above and below the main diagonal.

A matrix is called **banded** (with semibandwidth $m$) if the only locations with nonzero entries lie on the main diagonal or on the $m$ diagonals adjacent either above and below and below the main diagonal.

\[
\begin{pmatrix}
1-uq & 1-u_x & 1-u_p & 1-u_p & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots \\
1_q & 1_x & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
\end{pmatrix} =
\begin{pmatrix}
1 & 1-u & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
\end{pmatrix}
\]

\[B = X \cdot V\]

We usually write such systems as
A matrix $A$ is called **positive definite** if for all vectors $x$ we have $x^T A x > 0$.

A matrix is called **symmetric** if for all $i$, $j$ we have $a_{ij} = a_{ji}$.

A matrix is called **diagonally dominant** if for all $i$, we have

$$u > 0 \quad \sum_{i 
eq j} |a_{ij}| < |a_{ii}|$$
Now, in outer loop iteration, the task responsible for row \( \ell \) has the \( \ell \)-th row of the \( a \) matrix, but must

\[ [\tau] \] assign a task to each row of the matrix, including the \( x \) and the \( q \) values.

But we could do the inner loop in parallel, we can back solve for \( x' \), so we can't do the outer loop in parallel.

There are some inherently sequential parts to this, since we clearly need to have the value for \( x' \) before

Naively, this is \( \Theta(u) \).

\[
\begin{align*}
\tau_0 &= 0.0 \\
\tau_0 \cdot [\tau] x - [\tau] q &= [\tau] q \\
\text{for } \ell = 0 \text{ to } \ell-1 \text{ do} \\
\text{for } \ell = \ell-1 \text{ down to } 1 \text{ do}
\end{align*}
\]

Back Substitution by Rows.

Page 17
number of data items moved.

Since each message is a single value, this is also the
costs $\ell$ each time, for a total of $u$ in messages.

In each iteration of the outer loop, the task that just computed the new $x^*$ has to do a broadcast, which
has rows of average length $u$ and thus has to do $d/u$ operations.

We fix this by strife by shifting the matrix across tasks instead of allocating blocks. On average, each task/processor
at the bottom have mostly zeros.

The matrix would have most of the work, since the rows near the top have mostly nonzeros and the rows
of the matrix would have only a little work to do and then would be idle, while the tasks at the top of
another issue: If we blocked off rows and handed blocks of rows to the tasks, then some tasks at the bottom
So we have the task for row $i$ compute $x^*$ and then broadcast it.

be delivered the value of $x^*$ from the task for row $i$.  
the output of the multiplier could be streamed into the adder directly.

had separate multiplication and addition arithmetic units, this could be done very efficiently because

call, referred to as a \textsc{sxpy} (scalar \times X \text{ plus } Y), on machines like the Cray vector machines that

scientific computing that vendors of hardware and software usually include this as an optimized function

a vector (in this case a column vector $\mathbf{a}^T$) and adding in a vector $\mathbf{b}$] [\textsuperscript{T}]. This happens so often in

Note the $\mathbf{q} \cdot \mathbf{a} = [\textsuperscript{T}] [\textsuperscript{T}] - [\textsuperscript{T}] [\textsuperscript{T}] \mathbf{q}$ in the code above. We are multiplying a vector (x) times
For the row-stripped method,

\[ d \leq n \] instead of \( u \leq n \) and hence \( (u(\Theta)^{\Theta})^2 \) overall with each iteration, and hence \( (u(\Theta)^{\Theta})^2 \) of the sequential algorithm.

We get no computational parallelism, so the compute time is still the VEC of \( x \).

In this case, in iteration \( t \), process \( i \) will compute \( x^t \). This process will then update the columns and send

The previous back substitution assigned tasks to rows of the matrix. What about assigning tasks to columns?
worth noting to the effect of the parallel computer in the first place.

The columnar method is better only on matrices of less equal about 10 rows and columns. This makes it not

Basicallly, this is in order for the columnar method to be better. This is unlikely for 10 1 processors,

\[ d \frac{25000}{1007} > \left( \frac{d}{1} - \frac{25000}{1007} \right) \cdot u \]

\[ d \frac{25000}{1007} + \frac{d}{u} > \frac{25000}{u} + u \]

\[ d \frac{u}{9} 10 + d \frac{u}{e} 10 > \frac{u}{9} 10 + \frac{250}{e} + \frac{u}{e} 8 \]

Let \( X = 250 \) microseconds and \( Y = 10 \) nanoseconds. The columnar method will be better than the row

communication of the column strip method?

What is the difference between the row-stripped time and the \( d \frac{u}{e} 10 \) time and the time and \( d \frac{u}{e} 8 \) time and the \( u \)?
unnecessary, but by keeping an array of indices that permutes the physical order into a logical order, which pivoting will occur. This is done not by actually exchanging rows, which would be costly and is

- The naive algorithm pivots on the diagonal. For numerical reasons, in an adult version of Gauss, we usually

- The basic op in Gauss is a SAXPY

\[ 2n^3 \]
The inner two loops (the row loop and the S$\text{AXPY}$) are independent operations once the pivot row has been

\begin{verbatim}
broadcast. end for
end for
end for
do the S$\text{AXPY}$
for each row
choose a pivot row
for each column
\end{verbatim}

20.1 Gauss in Parallel
This is even worse that matrix-vector or matrix-matrix mult, so it does not scale well.

\[ d \leq d \sum_i \frac{\alpha}{\gamma \cdot u} < u \]

\[ d \leq d \sum_i \frac{\alpha}{\gamma \cdot u} < u \]

The total communication overhead is \( d \sum_i \frac{\alpha}{\gamma \cdot u} \). For isoefficiency, we have \( u \geq d \), as desired.

If we stripe the rows across processors, then we get full computational parallelism and the compute time is

\[ \text{latency broadcast} \cdot u \leq d \sum_i \frac{\alpha}{\gamma \cdot u} \leq d \sum_i \frac{\alpha}{\gamma \cdot u} \]

\[ \text{rows/cols} \]

Communication: On average we broadcast \( \frac{\gamma}{u} \) elements of the pivot row, and we do this for each of \( u \).
Once again, the row method is better for large, \( d \) small, and the column method for small, \( u \) large. Again, we have computation time balanced, so \( u^3 \), but this still scales as badly as the row striped version.

Again, we have computation time balanced, so \( u^3 \), but this still scales as badly as the row striped version.

Broadcast the \( z/\sqrt{u} \) elements of the pivot row.

Find the pivot row.

Be sent each iteration to the next process.

This task needs to have the current index/permutation vector to know how to scan, so this vector needs to

The task with column \( i \) in iteration \( i \), needs to find the pivot row. This is \( u \) time to scan down the column.

Each column goes to a different processor.

\[ \text{Column-striped Parallel Gauss} \]
and communication are totally overlapped, is up.

**Communication:** Each process must send \( n - 1 \) messages, so \( n \) communication takes parallel overhead, if compute

And so forth.

Process 2.

When process 1 finishes, it can determine the second pivot column and send that with its row to

Process 0 and the pivot column to Process 2, and then begins to do its row reduction on its rows.

Process 1 receives row 0 and the pivot column.

Process 0 searches row 0 for a pivot column.

Pivoting on choice of rows is not magic. We could just as easily pivot on columns instead.

What we want to do is overlap the computation and the message passing.
So memory (\( M_u \)) per processor stays constant as number of processors grows.

\[
\begin{align*}
d u &< u \\
d C &< u \\
d u C &< u
\end{align*}
\]
\[
\left(\sum_{(0)} x^f \frac{\bar{\tau}'1\neq f}{(1)} - \frac{1}{(1)} x^0 \bar{\tau}D - \frac{0}{(1)} x^0 \bar{\tau}D - \bar{\varepsilon}_q}{1} \right) \frac{\bar{\varepsilon}D}{1} = \frac{\bar{\varepsilon}x}{(1)}
\]

and so forth.

\[
\left(\sum_{(0)} x^f \frac{1\neq \tau}{(1)} - \frac{0}{(1)} x^0 \tauD - \tau_q}{1} \right) \frac{\tauD}{1} = \frac{\tau}{(1)}
\]

\[
\left(\sum_{(0)} x^f \frac{0\neq \tau}{(1)} - \frac{0}{(1)} \bar{\tau}D\right) \frac{\tauD}{1} = \frac{\bar{\tau}x}{(1)}
\]

Choose an initial solution vector \(x^{(0)}\).
Do the same as above, but update all of $x^{(t+1)}$ using the values of $x^{(t)}$ in parallel.
In practice, we don't have exact arithmetic and we will need to stop when we get within the usual sorts of
system in insufficient iterations.

If we assume that exact arithmetic is in effect, then the Conjugate Gradient will find the solution to an $n \times n$ system.

The Conjugate Gradient method solves $A x$ by finding this minimum.

\[ q = x^T A x \]

is the unique minimum solution to

\[ c + q ||x||^2 x^T A^{-1} x = (x^T b) x \]

Let $A$ be an $n \times n$, symmetric, positive definite matrix, then the solution vector $x$ to

20.4 Conjugate Gradient
1. Compute the gradient

\[ q = (1-i)x_v = (i)b \]

2. Compute the direction vector

\[ q - (1-i)x_v = (i)b \]

3. Compute the step size

\[ \frac{(i)pV_L((i)p)}{(i)b_L((i)b)} = (i)s \]

4. Compute the new approximation of the x vector

\[ (i)p(i)^s + (1-i)x = (i)x \]

Conjugate Gradient Algorithm
\[
\begin{align*}
\begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} \frac{\begin{pmatrix} \Pi \\ \Sigma \end{pmatrix}}{\begin{pmatrix} \Pi \\ \Sigma \end{pmatrix}} (\Pi - \Sigma) + \begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} = (1)p \\
\begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} \begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} (3 \ 1) = (1)b \\
\begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} &= (0)b \begin{pmatrix} 0 \\ 0 \end{pmatrix} (0)p \begin{pmatrix} 0 \\ 0 \end{pmatrix} = (0)x \\
\begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} &= \begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} \begin{pmatrix} \Pi \\ \Sigma \end{pmatrix} (1, 2) \cdot (3, 1, 2)
\end{align*}
\]
\[
\begin{pmatrix}
0.40 \\
1.93
\end{pmatrix}
\approx
\begin{pmatrix}
11 \\
7
\end{pmatrix}
\begin{pmatrix}
615 \\
0.17
\end{pmatrix}
+ \begin{pmatrix}
0 \\
0
\end{pmatrix}
= (I)^x
\]

\[
\begin{align*}
&\frac{615}{0.17} \\
&\approx 0.2764
\end{align*}
\]

\[
\begin{pmatrix}
11 \\
7
\end{pmatrix}
\begin{pmatrix}
3 & 1 \\
1 & 2
\end{pmatrix}
\begin{pmatrix}
11 \\
7
\end{pmatrix}
= (I)^s
\]