Systolic Arrays and Algorithms
Systolic Arrays

- This is a form of **pipelining**, sometimes in more than one dimension.

- The term “systolic” was first used in this context by H.T. Kung, then at CMU; it refers to the “pumping” action of a heart.

- Machines have been constructed based on this principle, notably the iWARP, fabricated by Intel in the 80’s.
Early Systolic Operation

- Colossus Mark 2 at Bletchley Park, 1944 (http://en.wikipedia.org/wiki/Colossus_computer)

- “Colossus included the first ever use of shift registers and systolic arrays, enabling five simultaneous tests, each involving up to 100 Boolean calculations, on each of the five channels on the punched tape (although in normal operation only one or two channels were examined in any run).”

In 1994, a team led by Tony Sale (right) began a reconstruction of a Colossus at Bletchley Park. Here, in 2006, Sale supervises the breaking of an enciphered message with the completed machine.
Systolic Matrix Multiplication

- Processors are arranged in a 2-D grid.
- Each processor accumulates one element of the product.
- The elements of the matrices to be multiplied are “pumped through” the array.
- We illustrate for individual elements, but can be applied to block decompositions.
Block Multiplication

one block of product

columns of blocks

rows of blocks
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices

alignments in time

rows of a

columns of b

\[
\begin{align*}
a_{0,2} &\quad a_{0,1} &\quad a_{0,0} \\
a_{1,2} &\quad a_{1,1} &\quad a_{1,0} \\
a_{2,2} &\quad a_{2,1} &\quad a_{2,0}
\end{align*}
\]
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices

alignments in time

\[
\begin{align*}
& a_{0,2} \quad a_{1,2} \quad a_{2,2} \\
& + a_{1,1} \quad a_{2,1} \\
& a_{0,1} \quad a_{1,0} \quad a_{2,0}
\end{align*}
\]

\[
\begin{align*}
& b_{2,0} \quad b_{1,1} \quad b_{0,2} \\
& b_{2,1} \quad b_{1,2} \\
& b_{0,1} \quad b_{0,0}
\end{align*}
\]
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices
Systolic Matrix Multiplication
Illustrated with two 3x3 matrices
Analysis

- Assuming $n \times n$ matrices ($n^2$ elements)
- $O(n^3)$ operations
- $O(n)$ time steps on $n^2$ processors
A Related Algorithm: Cannon’s Method

- Let’s take another view of systolic multiplication: Consider the rows and columns of the matrices to be multiplied as *strips* that slide past each other.

- The strips are first *pre-skewed* so that the correct elements are multiplied at each time step.
Cannon’s Method

- Rather than have some processors idle,

- wrap the array rows and columns so that every processor is doing something on each step.

- In other words, rather than feeding in the elements, they are rotated around,

- starting in an initially staggered position as in the systolic model.

- We also change the order of products slightly, to make it correspond to more natural storage by rows and columns.
Cannon’s Matrix Multiply (“Pipe-Roll” method)

Note that the a diagonal is in the left column and the b diagonal is in the top row.

Products computed at each step:

Step 1

<table>
<thead>
<tr>
<th>a00*b00</th>
<th>a01*b11</th>
<th>a02*b22</th>
</tr>
</thead>
<tbody>
<tr>
<td>a11*b10</td>
<td>a12*b21</td>
<td>a10*b02</td>
</tr>
<tr>
<td>a22*b20</td>
<td>a20*b01</td>
<td>a21*b12</td>
</tr>
</tbody>
</table>

Step 2

<table>
<thead>
<tr>
<th>a02*b20</th>
<th>a00*b01</th>
<th>a01*b12</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10*b00</td>
<td>a11*b11</td>
<td>a12*b22</td>
</tr>
<tr>
<td>a21*b10</td>
<td>a22*b21</td>
<td>a20*b02</td>
</tr>
</tbody>
</table>

Step 3

<table>
<thead>
<tr>
<th>a01*b10</th>
<th>a02*b21</th>
<th>a00*b02</th>
</tr>
</thead>
<tbody>
<tr>
<td>a12*b20</td>
<td>a10*b01</td>
<td>a11*b12</td>
</tr>
<tr>
<td>a20*b00</td>
<td>a21*b11</td>
<td>a22*b22</td>
</tr>
</tbody>
</table>

Example sum: \( c_{0,2} = a_{0,2} \cdot b_{2,2} + a_{0,1} \cdot b_{1,2} + a_{0,0} \cdot b_{0,2} \)
Cannon’s Method for **Block Multiplication**

- At each step, entire blocks are transmitted down and to the left of neighboring PE’s.

- Memory space is conserved.
Fox’s Algorithm

- Also for block matrix multiplication, it has a resemblance to Cannon’s algorithm.

- The difference is that on each cycle:
  - A row block is broadcast to each other processor in the row.
  - The column blocks are rolled cyclically.
Fox’s Algorithm

Step 1

<table>
<thead>
<tr>
<th>a00*b00</th>
<th>a00*b01</th>
<th>a00*b02</th>
</tr>
</thead>
<tbody>
<tr>
<td>a11*b10</td>
<td>a11*b11</td>
<td>a11*b12</td>
</tr>
<tr>
<td>a22*b20</td>
<td>a22*b21</td>
<td>a22*b22</td>
</tr>
</tbody>
</table>

A different row element of $a$ is broadcast in each step.

Step 2

<table>
<thead>
<tr>
<th>a01*b10</th>
<th>a01*b11</th>
<th>a01*b12</th>
</tr>
</thead>
<tbody>
<tr>
<td>a12*b20</td>
<td>a12*b21</td>
<td>a12*b22</td>
</tr>
<tr>
<td>a20*b00</td>
<td>a20*b01</td>
<td>a20*b02</td>
</tr>
</tbody>
</table>

b column elements are rolled each step

Step 3

<table>
<thead>
<tr>
<th>a02*b20</th>
<th>a02*b21</th>
<th>a02*b22</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10*b00</td>
<td>a10*b01</td>
<td>a10*b02</td>
</tr>
<tr>
<td>a21*b10</td>
<td>a21*b11</td>
<td>a21*b12</td>
</tr>
</tbody>
</table>
Fox vs. Cannon

- Fox does not require pre-skewing.
- Broadcast may be cheaper than multiple block transfers.
Other Uses of Systolic Arrays

- Cholesky Factorization of Matrices
- Digital filters (FIR)
- Genome Matching (Smith-Waterman Algorithm, Paracel Inc.)

Analysis of a Class of Parallel Matrix Multiplication Algorithms
http://www.cs.utexas.edu/users/plapack/papers/ipps98/ipps98.html

Two-Dimensional **Block Cyclic** Data Distribution as a Key to Load Balancing and Software Reuse (part of ScaLAPACK User’s Guide)
http://www.netlib.org/scalapack/slug/node110.html