Signal Processing on Raw

Hank Hoffmann

MIT LCS/LL

hank@mit.edu

joint work with

Volker Strumpen, Anant Agarwal, and the Raw Team

March 2003

Outline

1. Running Example: Matrix Multiplication

2. From streams to systolic arrays
   (a) streaming inner product
   (b) connecting streams

3. From systolic arrays to the moon
   (a) Drawbacks of systolic arrays
   (b) How to do better

Matrix Multiplication

$C = AB$, **all** $N \times N$ **matrices**

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0, c[i][j]=0.0; k<N; k++)
      c[i][j] += a[i][k]*b[k][j];
```

Computational complexity: $\Theta(N^3)$

N^2 **inner products:** $c_{ij} = \sum_{k=0}^{N-1} a_{ik} \cdot b_{kj}$

Focus on computing inner products

Inner Product Streams

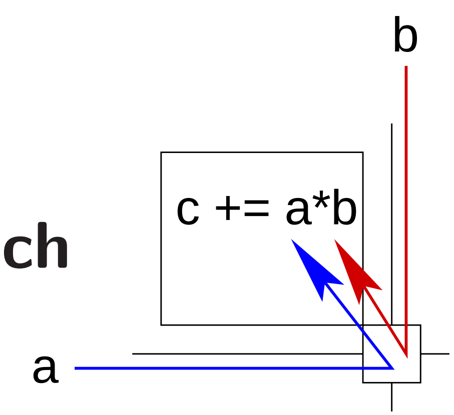
Heart of matrix multiply is inner product:

```
for (k=0, c=0.0; k<N; k++)
  c += a[i][k]*b[k][j];
```

Use streams to compute the inner product of a and b:

Store c locally in a register

a and b stream through the switch



We now have a $100\,\%$ efficient inner product

Connecting Streams

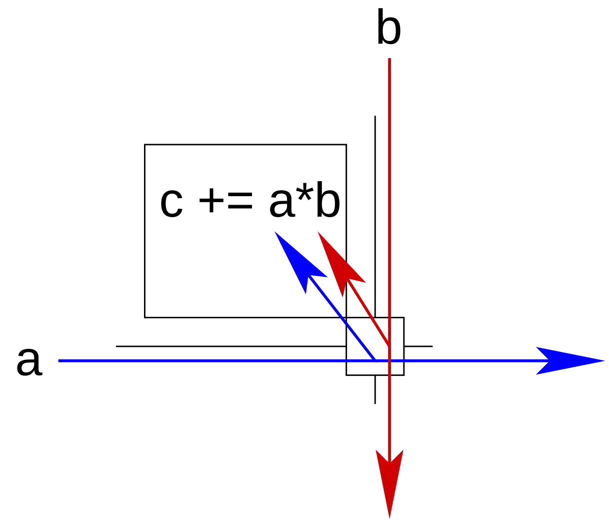
To compute $C = AB$ requires N^2 inner products

Each row of A and column of B used N times

Communicate each row and column while computing

Switch code:

```
nop route $cEi->$csti,
          $cEi->$cWo,
          $cNi2->$csti2,
          $cNi2->$cSo2
```

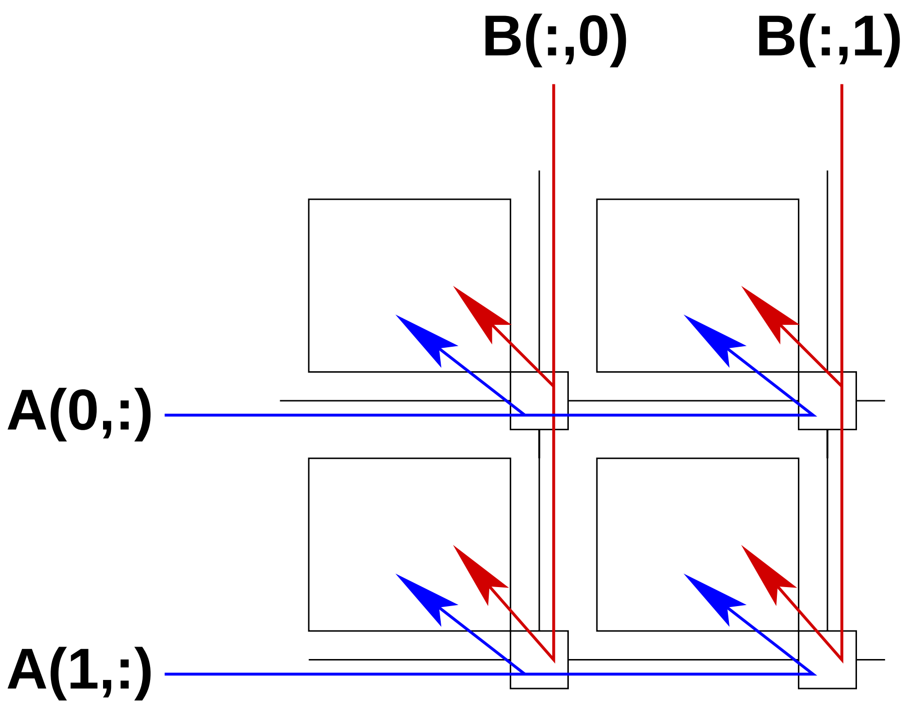


Now we can form a systolic array for matrix multiplication.

Raw's networks allow "free" corner-turns

Systolic Matrix Multiplication

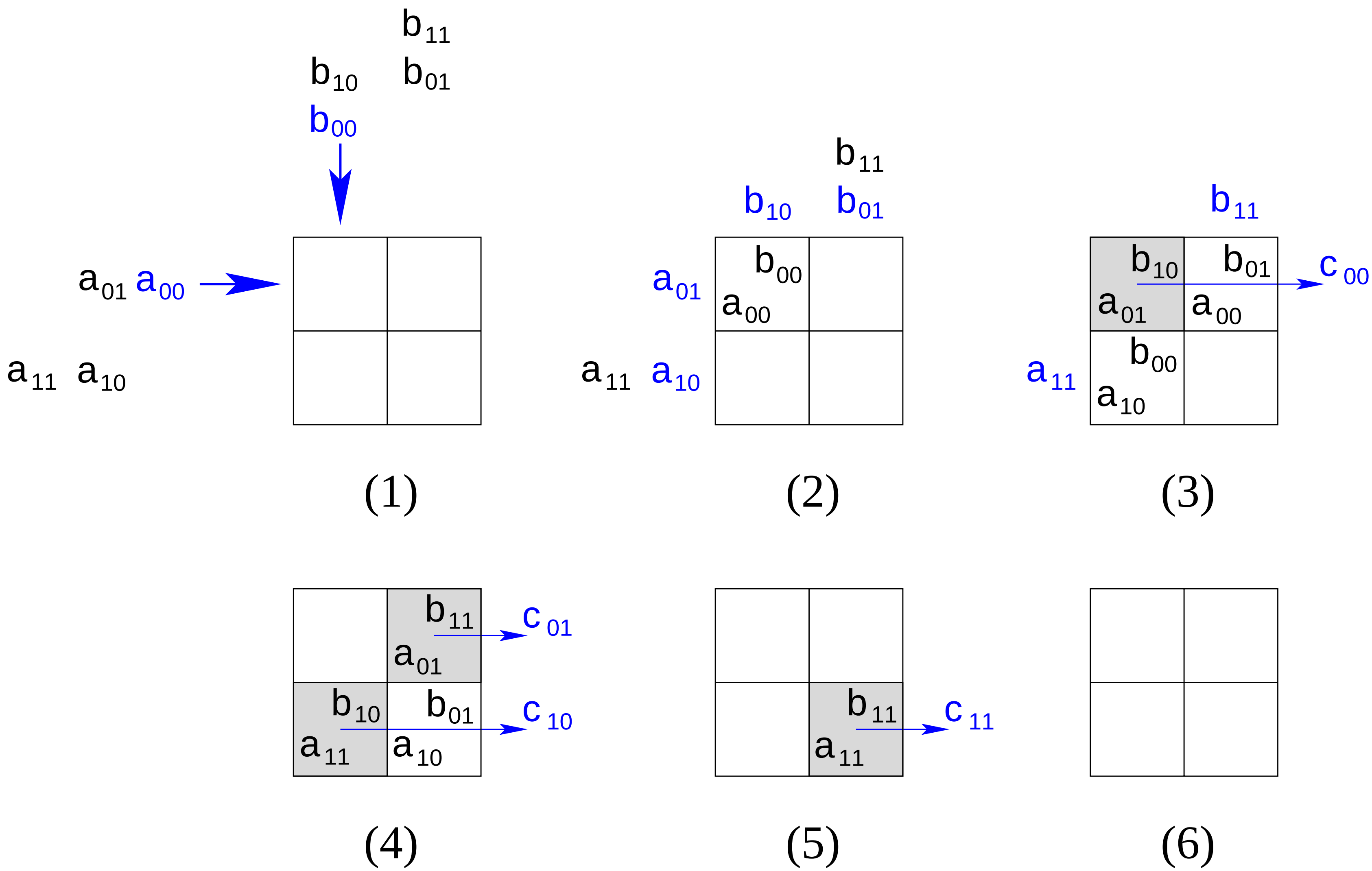
We stream the row vectors of A and the column vectors of B through N^2 inner product tiles. (here: $N = 2$)



Size of inner dimension does not matter...

Systolic Matrix Multiplication

Here individual elements stream through the array.



Systolic Array Summary

1. **Systolic matrix multiplication is $100\,\%$ efficient**

2. **What if $N > 4$ (we can't fit the problem onto Raw)?**
   (a) **Simulate a larger raw fabric/larger array**
   (b) **Use 1 Raw tile to simulate many virtual tiles**
   (c) **Use local memory to store intermediate results**

3. **Problem: Cost of `load`'s and `store`'s for inner product**

Data Type	FLOPs	Memory Ops	Total Ops	Max Efficiency
Real	2	4	6	$33\,\%$
Complex	8	8	16	$50\,\%$

Need to move `loads` and `stores` off critical path...

Improving our Matrix Multiplication

We want the efficiency of a systolic array on large problems

1. **Partition into $(N/R)^2$ problems by recursively applying:**

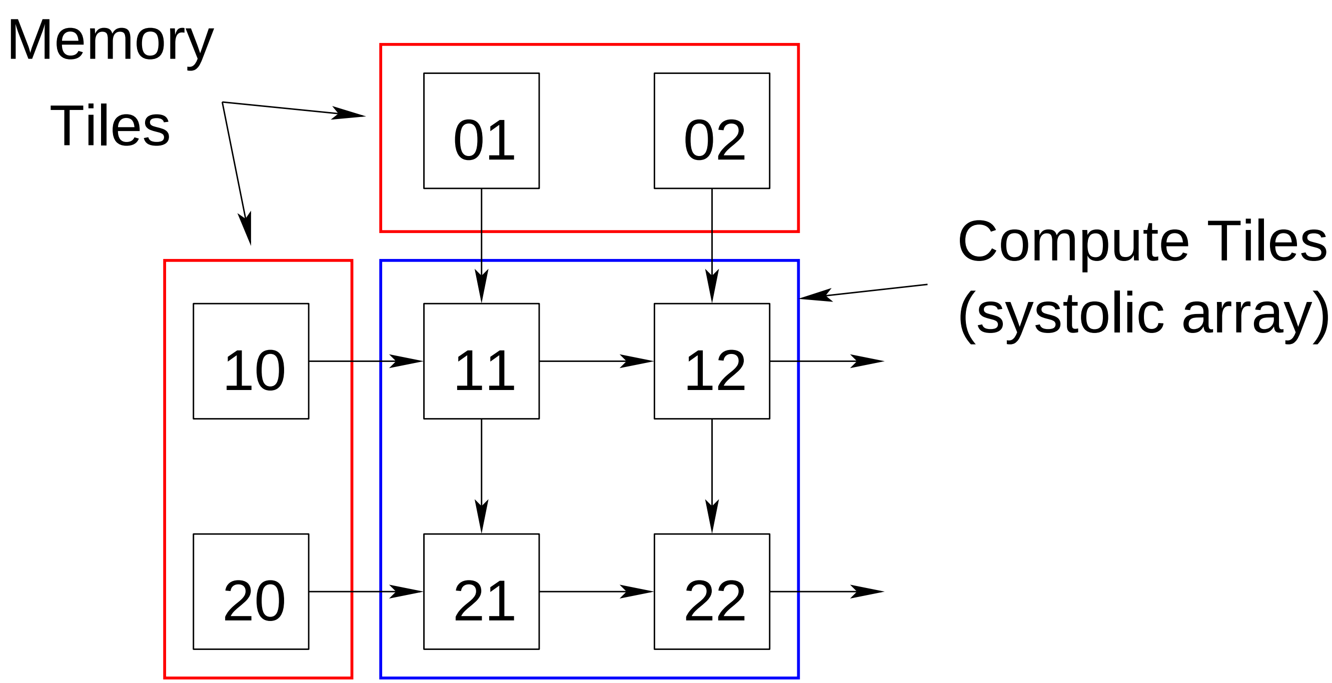
$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \end{pmatrix}$$

2. **Each submatrix of A is $R \times N$, those of B are $N \times R$**
   (a) **Each submatrix of C is computed on $R \times R$ tiles**
   (b) **No simulation required!**

3. **Decouple memory access - store A_{ij}, B_{kl} until necessary**
   (a) **R processors store rows of A**
   (b) **R processors store columns of B**
   (c) **$2R$ memory processors**

Decoupled Matrix Multiplication

Memory tiles implement the data access to stream the rows and columns into the systolic array of compute tiles.



Decoupled Systolic Matrix Multiplication

	B(:,2) B(:,0)	B(:,3) B(:,1)
A(0,:) A(2,:)		
A(1,:) A(3,:)		

(1)

Decoupled Systolic Matrix Multiplication

	B(:,2) B(:,0)	B(:,3) B(:,1)
A(0,:) A(2,:)	b_{00} a_{00}	
A(1,:) A(3,:)		

(2)

Decoupled Systolic Matrix Multiplication

	B(:,2) B(:,0)	B(:,3) B(:,1)
A(0,:) A(2,:)	b_{10} a_{01}	b_{01} a_{00}
A(1,:) A(3,:)	b_{00} a_{10}	

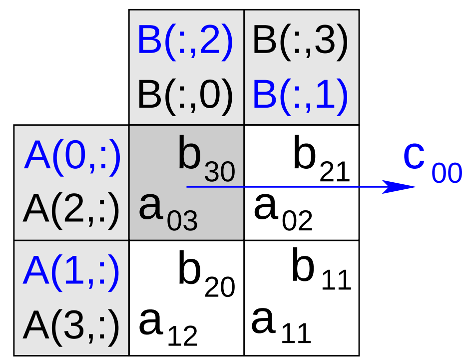
(3)

Decoupled Systolic Matrix Multiplication

	B(:,2) B(:,0)	B(:,3) B(:,1)
A(0,:) A(2,:)	b_{20} a_{02}	b_{11} a_{01}
A(1,:) A(3,:)	b_{10} a_{11}	b_{01} a_{10}

(4)

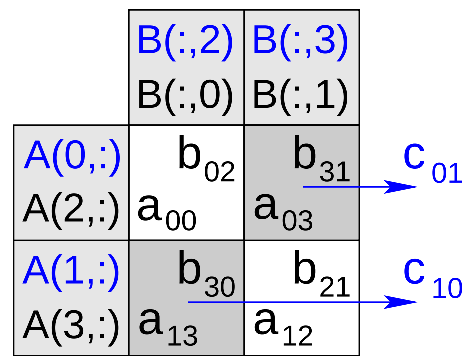
Decoupled Systolic Matrix Multiplication



(5)

- **We have reached steady state**
- **Systolic array tiles executing one op per clock cycle**

Decoupled Systolic Matrix Multiplication



(6)

- **Starting second submatrix multiplication**

Calculating Matrix Multiply Efficiency

$$
\begin{aligned}
\textbf{FLOPs, } F &= 2N^3 \\
\textbf{Cycles, } C &= 2(N/R)^3 R + 6R \\
\textbf{Tiles, } T &= R^2 + 2R \\
\textbf{ratio of } N \textbf{ to } R\textbf{, } \sigma &= N/R
\end{aligned}
$$

$$
\begin{aligned}
E(N,R) &= \frac{F}{C \cdot T} \\
E_{mm}(N,R) &= \frac{2N^3}{(2(N/R)^3 R + 6) \cdot (R^2 + 2R)} \\
E_{mm}(\sigma,R) &= \frac{\sigma^3}{\sigma^3 + 3} \cdot \frac{R}{R+2}
\end{aligned}
$$

$$
\lim_{\sigma,R\to\infty} E(\sigma,R) = 1, \qquad E_{mm}(\infty,2) = 50\,\%
$$

Efficiency has gone from $33\,\%$ to $50\,\%$.
Design is scalable - more processors gives more efficiency!

What just happened?

We now can increase efficiency as we increase tiles

How? Move `load`'s and `store`'s off critical path

We made use of:

1. Systolic algorithms [Kung and Leiserson, 1978]
   (a) provide solution when Problem Size(N) $=$ Network Size(R)
2. Decoupled Access Execute Architectures [Jim Smith, 1982]
   (a) separate memory accesses from computation
3. Out of Core Algorithms [Sivan Toledo, 1999]
   (a) work on large problems with limited space
4. Fewer memory tiles than compute tiles

Stream Algorithms

We call an algorithm that meets these four criteria a Stream Algorithm

1. **Stream algorithms are efficient on small Raw fabrics**
2. **Stream algorithms scale as Raw fabric scales**
3. **Stream algorithms approach $100\,\%$ efficiency**

Existing Stream Algorithms:

Convolution	DFT
Matrix Multiplication	Triangular Solver
LU Factorization	QR Factorization

Conclusion

1. We converted streams into efficient matrix multiplication

2. We derived a 4 step method to convert other algorithms

3. Using this method we write DSP code for Raw that is:

   (a) Efficient

   (b) Scalable

4. Stream algorithms tech report out soon...

5. Ask me about complex data