Mapping Signal Processing Kernels to Tiled Architectures

Henry Hoffmann
James Lebak [Presenter]
Massachusetts Institute of Technology
Lincoln Laboratory

Eighth Annual High-Performance Embedded Computing Workshop (HPEC 2004)
28 Sep 2004

This work is sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Credits

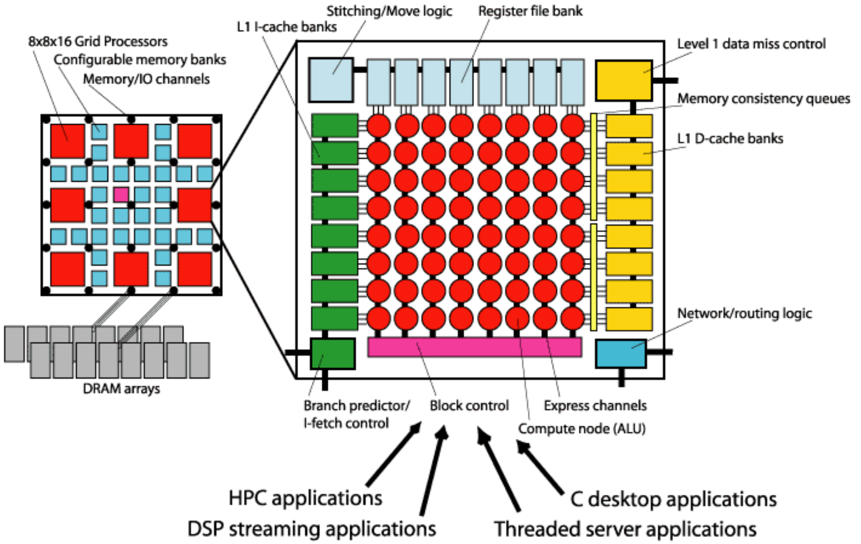
- **Implementations on RAW:**
  - – **QR Factorization: Ryan Haney**
  - – **CFAR: Edmund Wong, Preston Jackson**
  - – **Convolution: Matt Alexander**
- **Research Sponsor:**
  - – **Robert Graybill, DARPA PCA Program**

Tiled Architectures

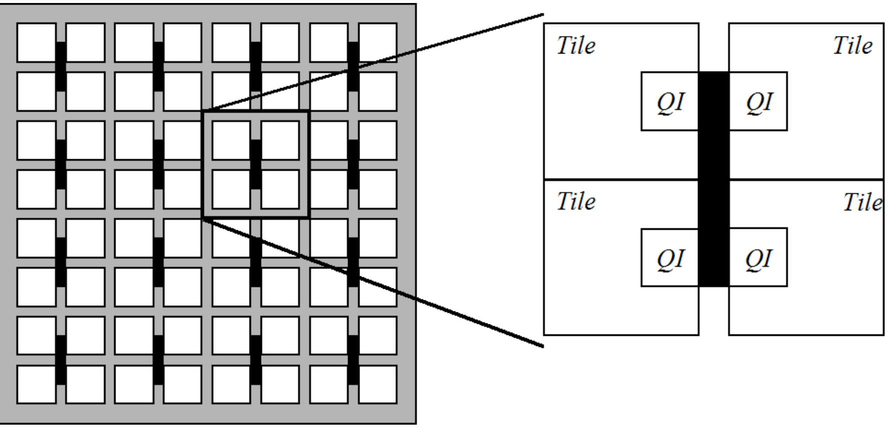
- Monolithic single-chip architectures are becoming rare in the industry
  - Designs become increasingly complex
  - Long wires cannot propagate across the chip in one clock
- Tiled architectures offer an attractive alternative
  - Multiple simple tiles (or “cores”) on a single chip
  - Simple interconnection network (short wires)
- Examples exist in both industry and research
  - IBM Power4 & Sun Ultrasparc IV each have two cores
  - AMD, Intel expected to introduce dual-core chips in mid-2005
  - DARPA Polymorphous Computer Architecture (PCA) program

PCA Block Diagrams

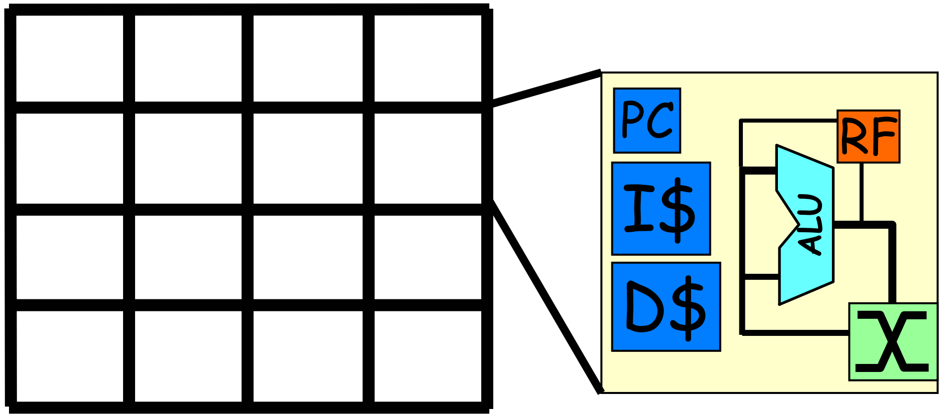
TRIPS (University of Texas)



Smart Memories (Stanford)



RAW (MIT)



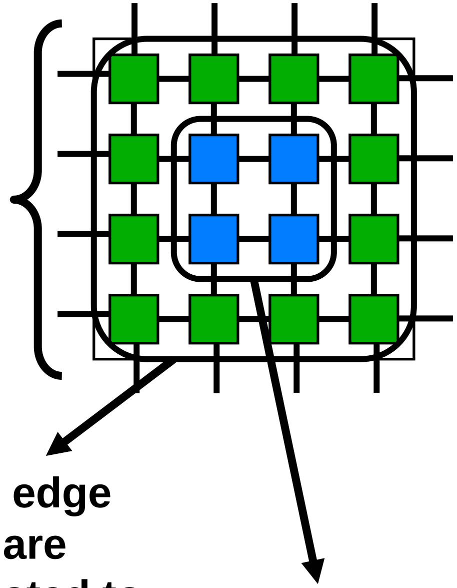
- All of these are examples of tiled architectures
- In particular, RAW is a 4x4 array of tiles
  - Small amount of memory per tile
  - *Scalar operand network* allows delivery of operands between functional units
  - Plans for a 1024-tile RAW fabric
- This research aims to develop programming methods for large tile arrays

Outline

- Introduction
- Stream Algorithms and Tiled Architectures
- Mapping Signal Processing Kernels to RAW
- Conclusions

Stream Algorithms for Tiled Architectures

Decoupled Systolic Architecture



M(R) edge tiles are allocated to memory management

P(R) inner tiles perform computation systolically using registers and static network

Stream Algorithm Efficiency:

$$E(N,R) = \frac{C(N)}{\overbrace{T(N,R)}^{\text{Time}} * \overbrace{(P(R) + M(R))}^{\text{Space}}}$$

where

N = problem size

R = edge length of tile array

C(N) = number of operations

T(N,R) = number of time steps

P(R) + M(R) = total number of tiles

Compute Efficiency Condition: $\lim_{\sigma,R \to \infty} E(\sigma,R) = 1$

where $\sigma = N/R$

Stream algorithms achieve high efficiency by:

–Partitioning the problem into sub-problems

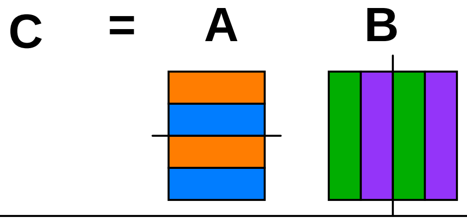
–Decoupling memory access from computation

–Hiding communication latency

Example Stream Algorithm: Matrix Multiply

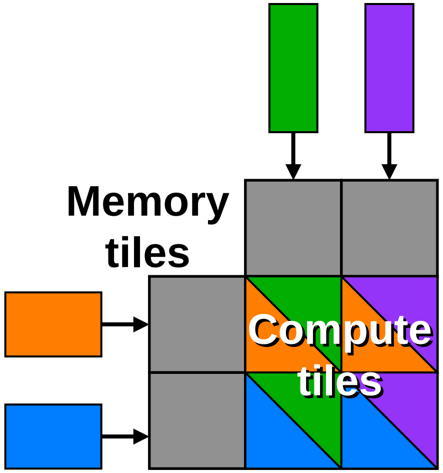
- **Calculate C=A B**
  - **Partition A into N/R *row blocks*, B into N/R *column blocks***

N = problem size
R = edge length of tile array



- **Computations can be pipelined**
  - **Cost is 2R cycles to start and drain the pipeline**
  - **R cycles to output the result**

- **In each phase, compute R^2 elements of C**
  - **Involves 2N operations per tile**
  - **N^2/R^2 phases**



Efficiency Calculation:

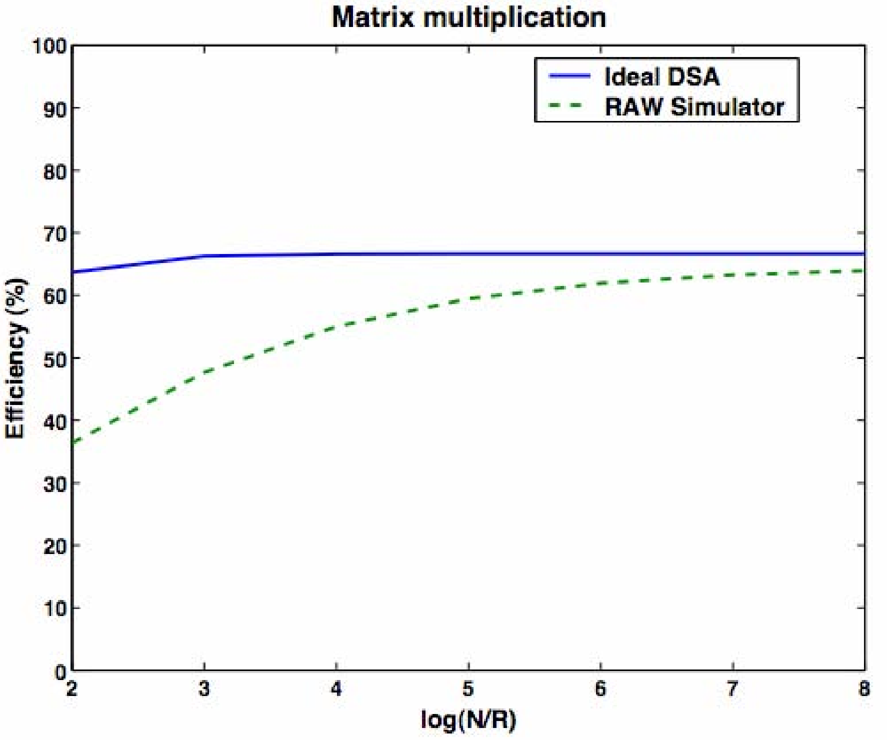
$$E\,(N,R) = \frac{2N^3}{(2N(N^2/R^2)+3R)(R^2+2R)}$$

$$= \frac{2\sigma^3}{2\sigma^3+3}\;\frac{R}{R+2} \qquad \text{for } \sigma = N/R$$

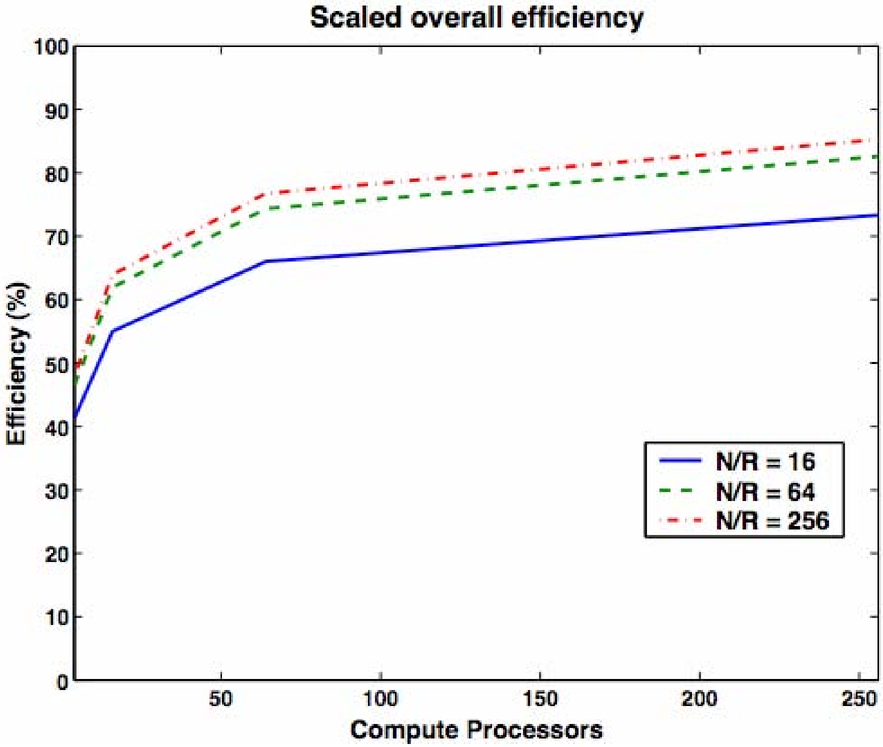
$$\lim_{\sigma,R\to\infty} E(\sigma,R) = 1$$

Achieves high efficiency as array size (N) & data size (R) grow

Matrix Multiply Efficiency



Assume a 4x4 decoupled systolic architecture or RAW surrounded by memory tiles (max efficiency=66%)

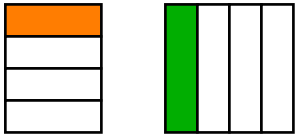


Scale the number of overall tiles
Smaller percentage of tiles devoted to memory leads to higher efficiency

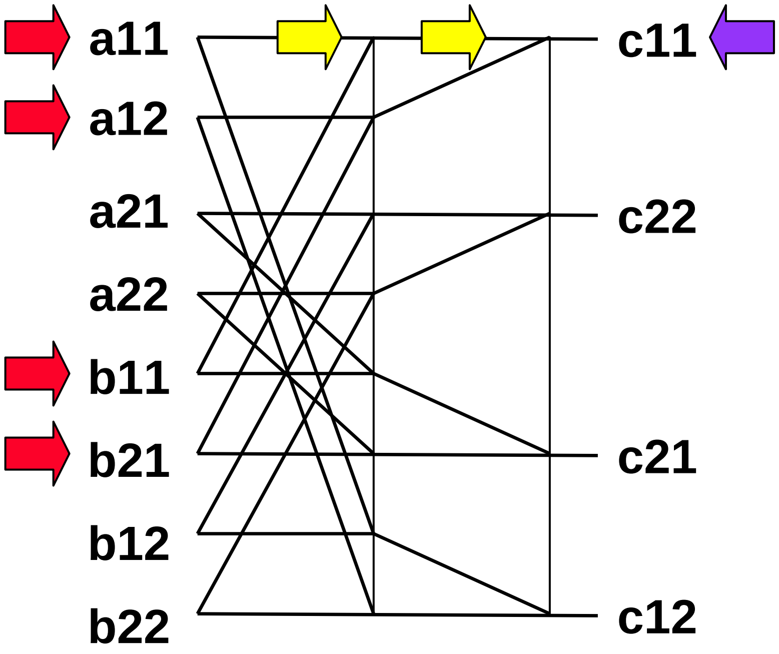
- Stream algorithms achieve high efficiency on large tile arrays
- We need to identify algorithms that can be recast as stream algorithms

Analyzing the Matrix Multiply

Consider the matrix multiply computation in more detail



To compute cij, row i of A is multiplied by column j of B
- 2N inputs required
- 2N operations required



- Examine the directed acyclic graph (DAG) for the matrix multiply
- For each output ⇦ produced
- There are W inputs ➡ required (O(N))
- The input i is used ➡ Q_i times (O(N))
  - These are intermediate products
- The matrix multiply is an example of an algorithm with a *constant ratio* of input data (W) to intermediate products (Q)

A constant W/Q implies a degree of *scale-invariance*:
- Communication and computation maintain the same ratio as N increases
- Therefore the implementation can efficiently use more tiles on large problems

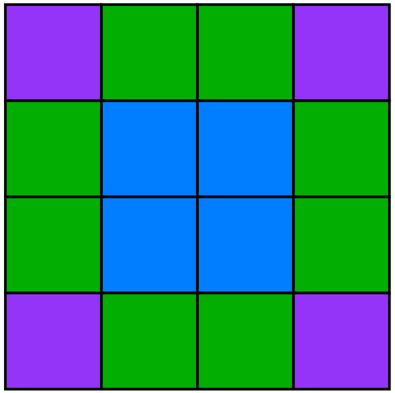
Outline

- Introduction
- Stream Algorithms and Tiled Architectures
- Mapping Signal Processing Kernels to RAW
  - QR Factorization
  - Convolution
  - CFAR
  - FFT
- Conclusions

RAW Test Board

- Write kernels to run on prototype RAW board
  - 4x4 RAW chip, 100 MHz
- MIT software includes cycle-accurate simulator
  - Code written for the simulator easily runs on board
  - Initial tests show good agreement between simulator and board
- Expansion connector allows direct access to RAW static network
  - Firmware re-programming required
  - External FPGA board streams data into and out of RAW
  - Design streams data into ports on corner tiles
  - Interface is not yet complete so present results are from simulator

Typical RAW configuration for a stream algorithm on prototype board:



I/O tiles
- Stream data to and from outside world

Memory tiles
- Store intermediate values
- Stream data to and from computation tiles

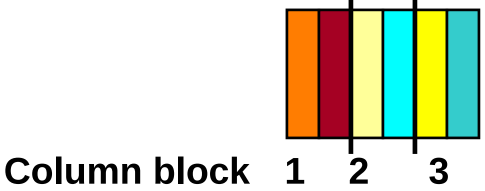
Computation tiles
- Perform computation systolically
- Use static network and registers

QR Factorization Mapping

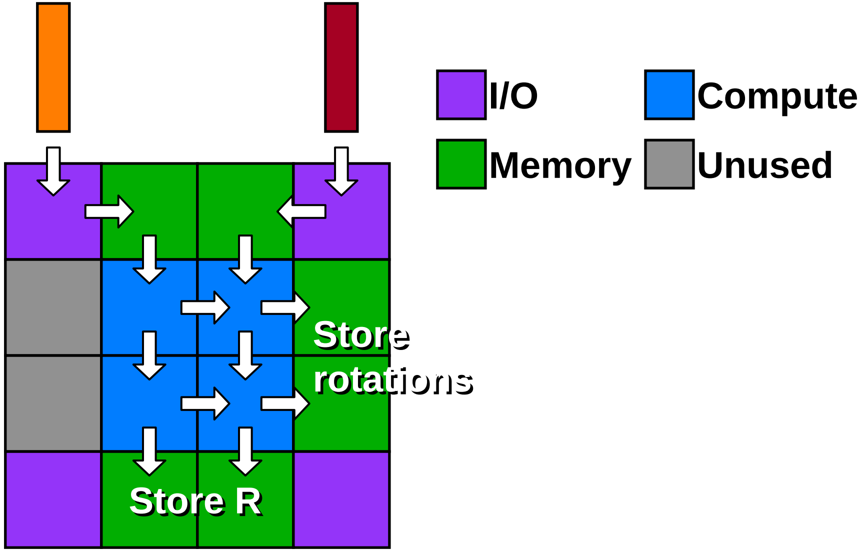
Algorithm to compute A=QR:

```
For each block of columns
  compute Givens rotations
  apply Givens rotation to A
```

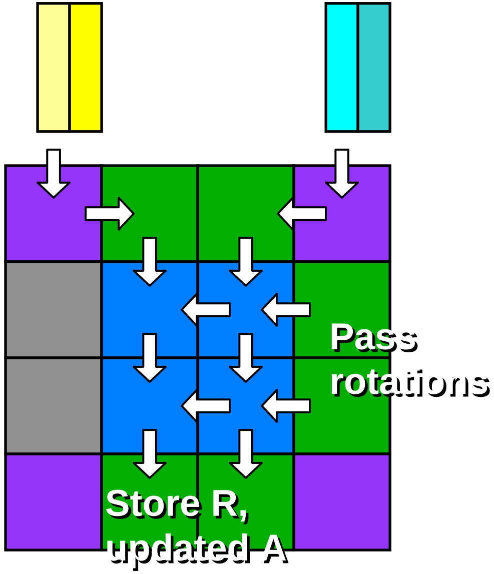
For a matrix A with six columns:



Data flow during rotation computation

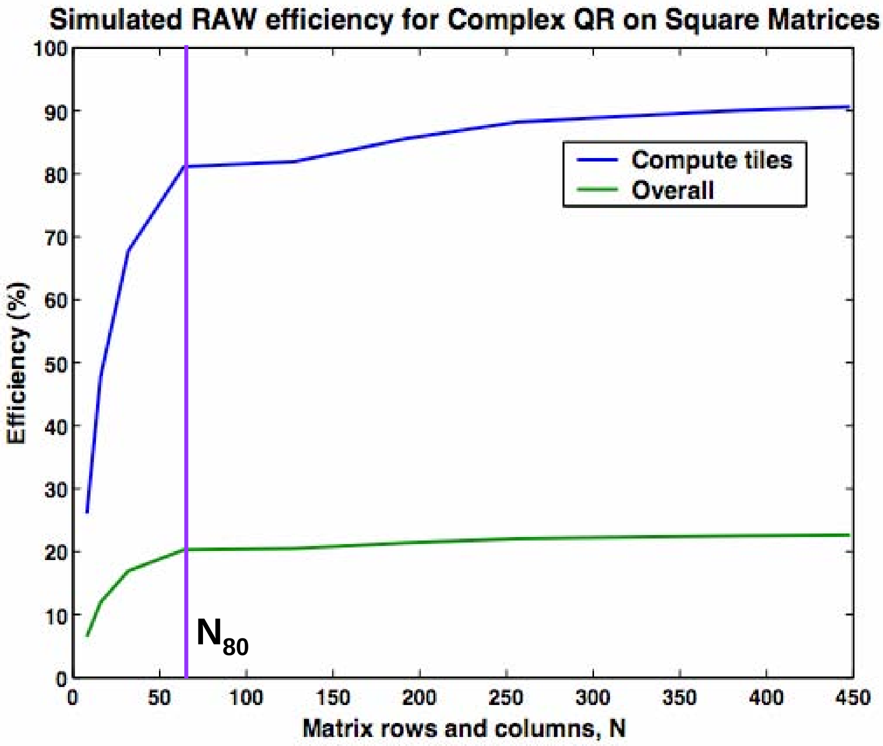


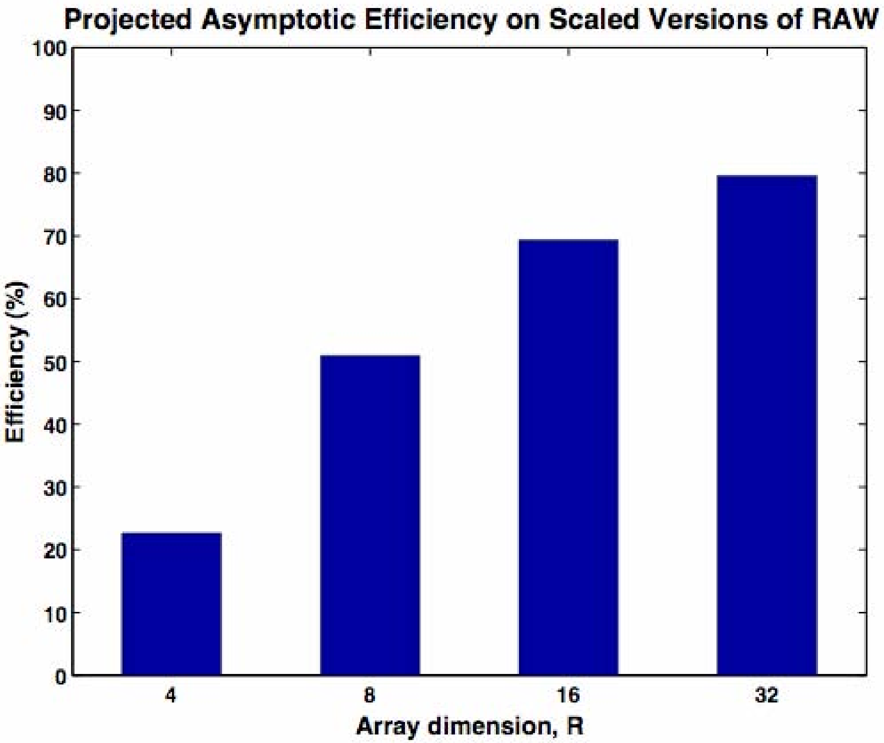
Data flow during rotation application



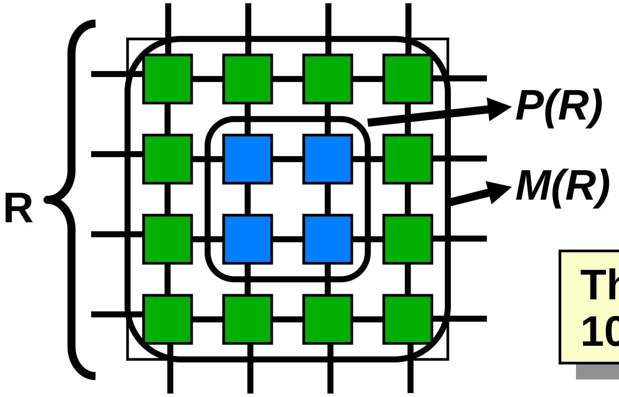
- I/O tiles are only used at start and end of process
  - In-between, data is stored in memory tiles
- This shows the flow for odd-numbered column blocks
  - For even-numbered blocks of columns, data flows from bottom memory tiles to the top of the array

Complex QR Factorization Performance





The QR factorization has a *constant ratio* of input data (W) to intermediate products (Q)

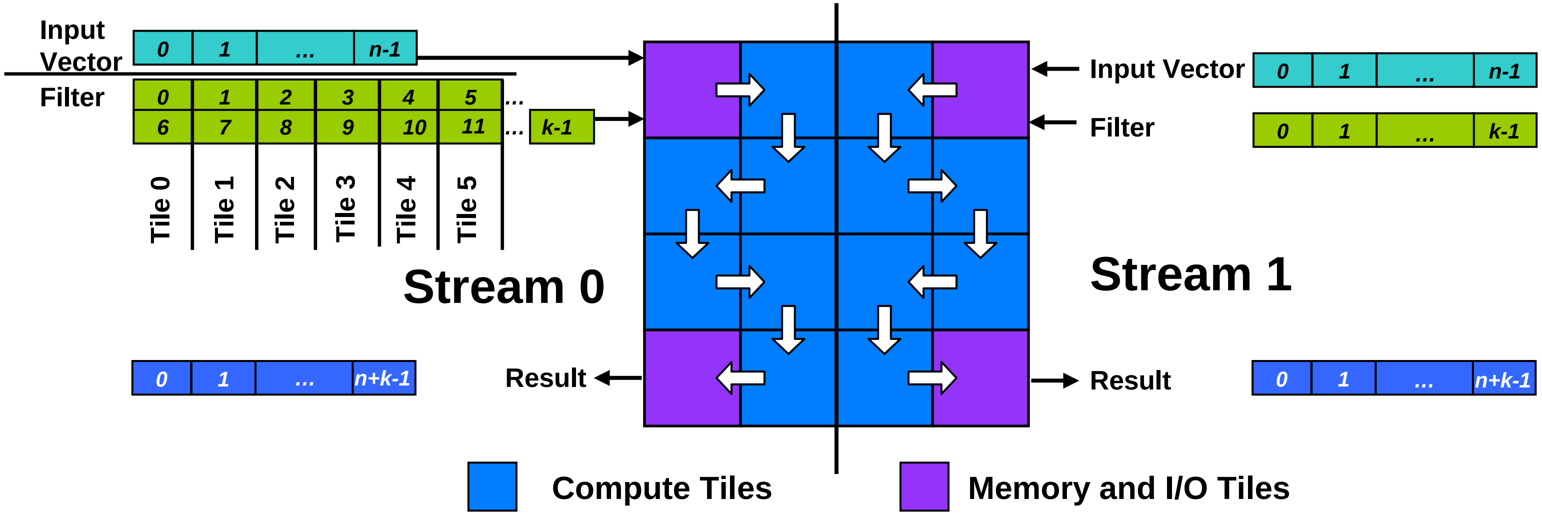


Projected matrix size N_{80} to achieve 80% efficiency on compute tiles *P(R)*:

R	N_{80}
4	64
8	128
16	256
32	512

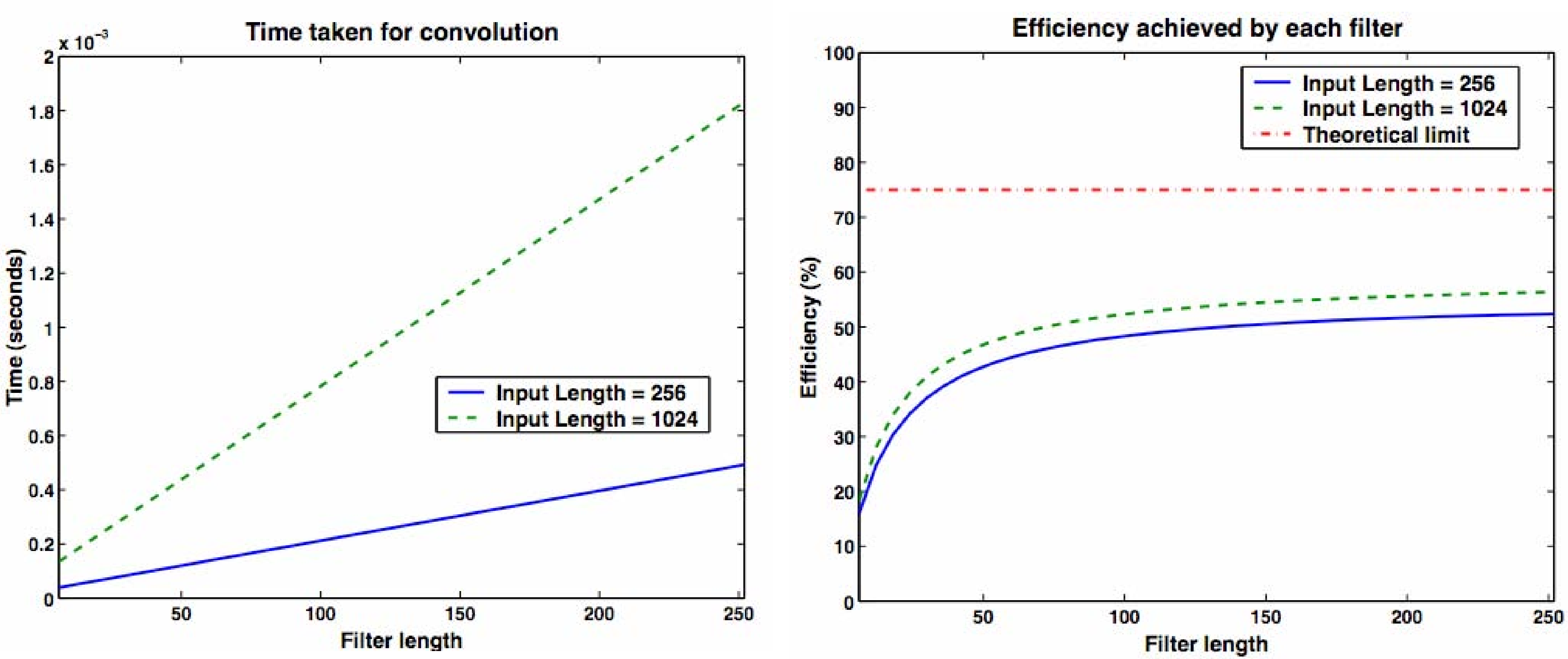
The QR factorization efficiency scales to 100% as array and data size increase

Convolution (Time Domain) Mapping



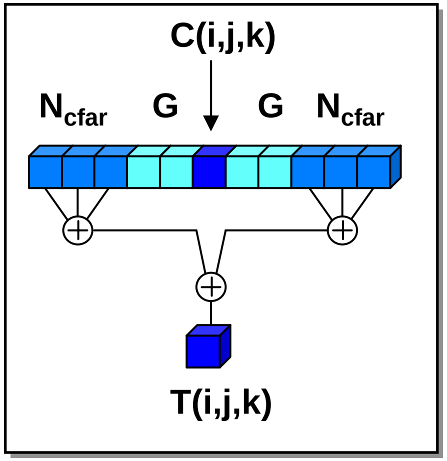
- **Filter coefficients distributed cyclically to tiles**
  - Each compute tile convolves the input with a subset of the filter
  - Assume *n* (data length) > *k* (filter length)
- **Each stream is a different convolution operation**
  - In multichannel signal processing applications we rarely perform just one convolution
- **12 of 16 tiles used for computation**
  - Maximum 75% efficiency

Convolution Performance



- Convolution achieves good performance in RAW simulator
- Longer filters and input vectors are more efficient
- Longer input vectors are also more easily mapped to more processors

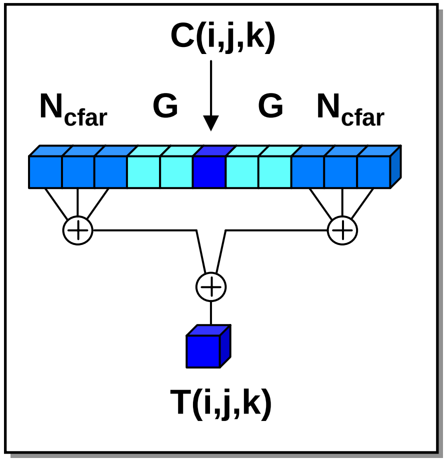
CFAR Mapping



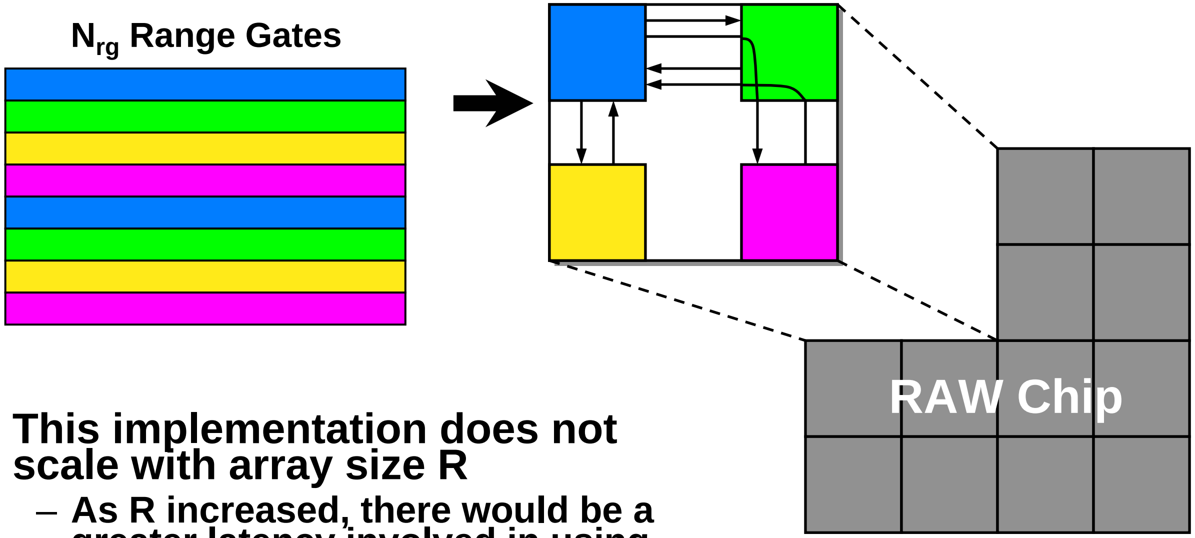
- Constant False-Alarm Rate (CFAR) Detection
- For each output:
  - There are W = O(N_{cfar}) inputs required
  - The input i is used Q_i = O(1) times

- For a long stream, CFAR requires 7 ops/cell
- Consider dividing up a stream over *R* tiles
  - 7/R operations per tile
  - N communication steps per tile
  - Communication quickly dominates computation
- Instead consider parallel processing of streams

CFAR Mapping



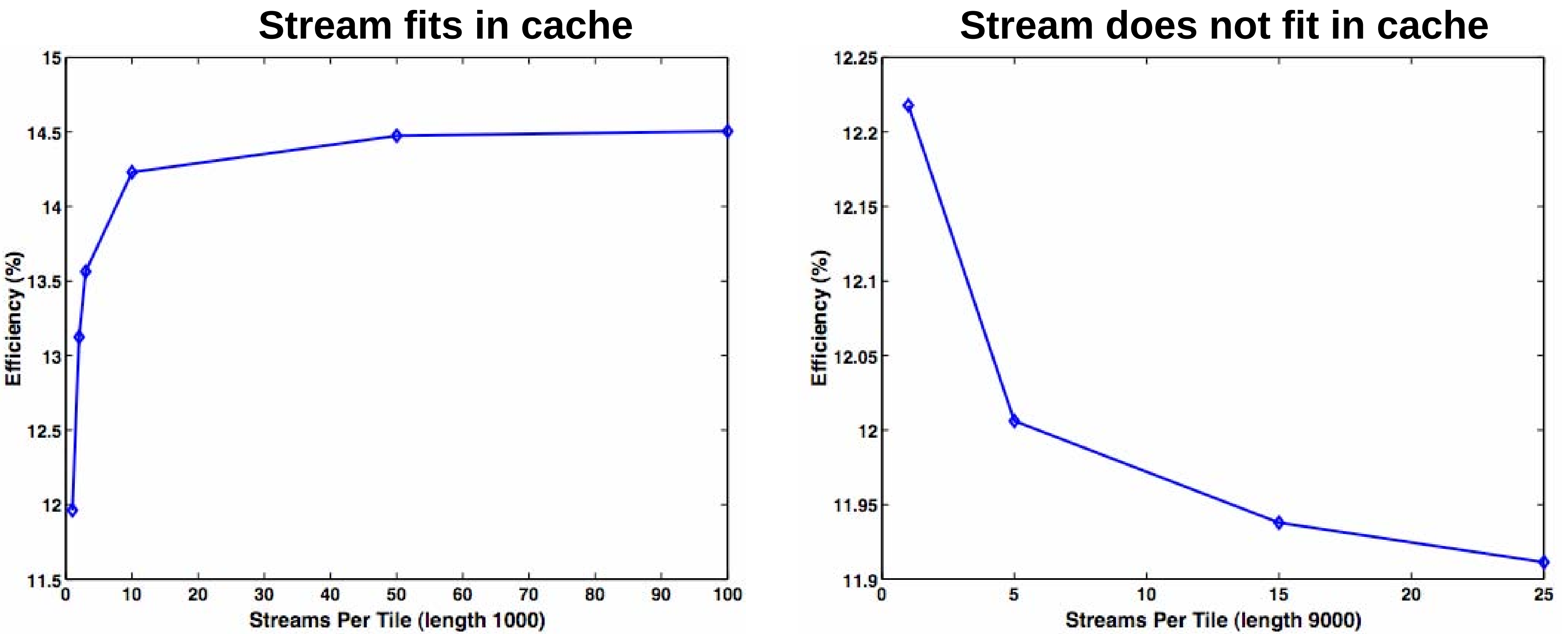
- Constant False-Alarm Rate (CFAR) Detection
- For each output:
  - There are W = $O(N_{cfar})$ inputs required
  - The input i is used $Q_i = O(1)$ times
- Goal is to move data through the chip as fast as possible



- Data cube is streamed into RAW using the static network
- Corner input ports receive data
- Each quadrant processes data from one port
- One row of range data ("one stream") is processed by a single tile
- Results gathered to corner tile and output

- This implementation does not scale with array size R
  - As R increased, there would be a greater latency involved in using tiles in the center of the chip

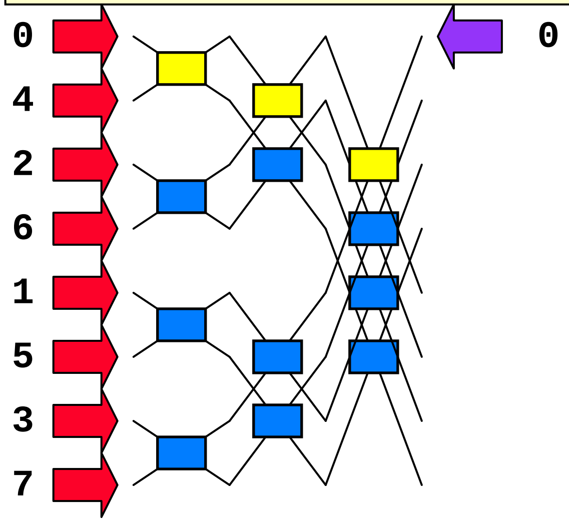
CFAR Performance

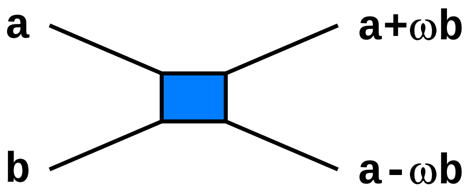


- **CFAR achieves an efficiency of 11-15%**
  - **Efficiency on conventional architectures = 5-10%, similarly optimized**
  - **RAW implementation benefits from large off-chip bandwidth**
- **Compute tile efficiency does not scale to 100% as for Stream Algorithms (matrix multiply, convolution, QR)**

Data Flow for the FFT

Cooley-Tukey Radix-2 FFT:
For each of ($\log_2 N$) stages
compute N/2 "butterflies"





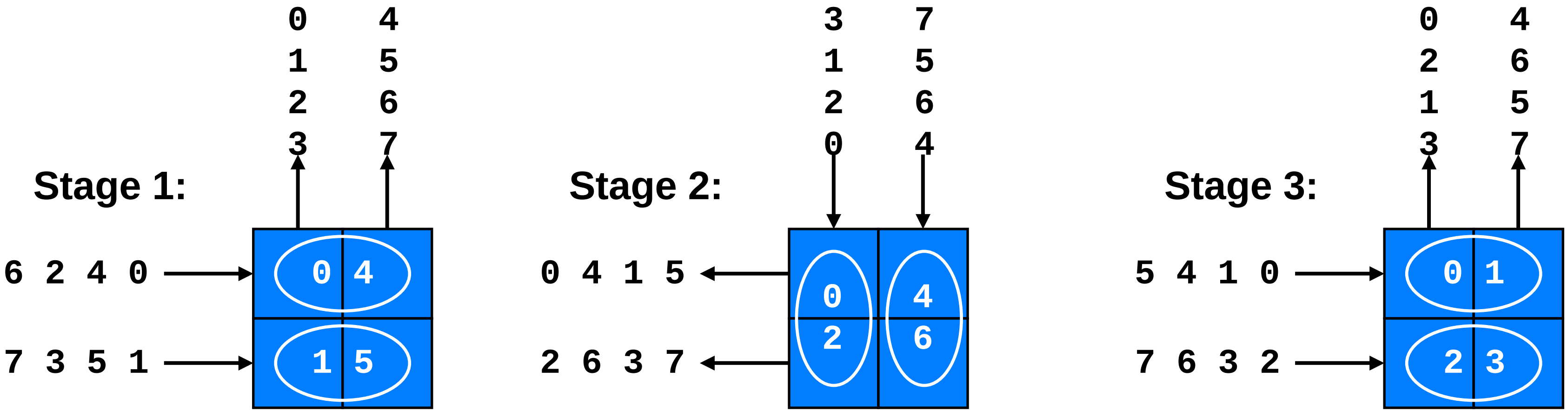
Radix-2 butterfly:

- **2 complex inputs**
- **precomputed weight ω**
- **10 real operations**

- **For each output ⬅ produced**
- **There are W inputs ➡ required (O(N))**
- **The input i is used 🟨 Q_i times ($O(\log_2 N)$)**
  - **These are intermediate computations**

- **W/Q is $O(N/\log_2 N)$**
  - **As N increases, communication requirements grow faster than computation**
  - **Therefore we expect that the Radix-2 FFT cannot efficiently scale**

Mapping the Radix-2 FFT to a Tile Array



- For each butterfly:
  - 4 + (R-1) cycles to clock inputs across the array
  - 10/R computations per tile
  - When R=2, tiles are used efficiently

    Can overlap computation (5 cycles) and communication (5 cycles)
  - When R>2, cannot use tiles efficiently

    Latency to clock inputs > number of ops per tile
- For each stage:
  - Pipeline N/2 butterflies on R rows or columns
- Overall efficiency limited to 50%
  - 2x2 compute tiles + 4 memory tiles

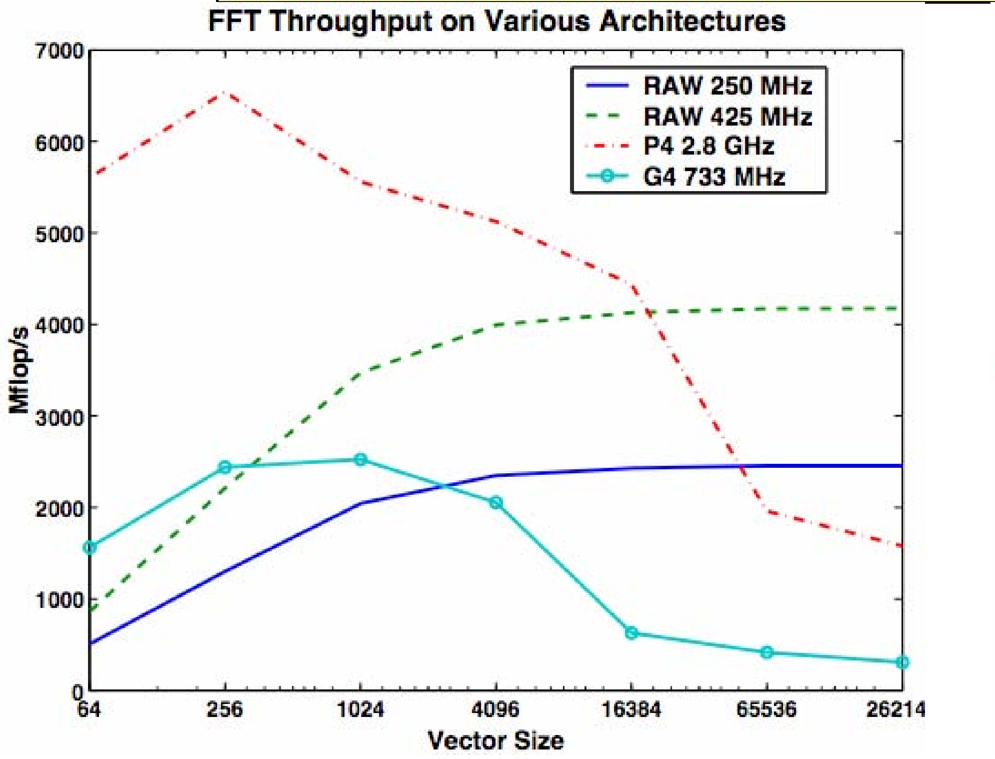
Mapping the Radix-R FFT to a Tile Array

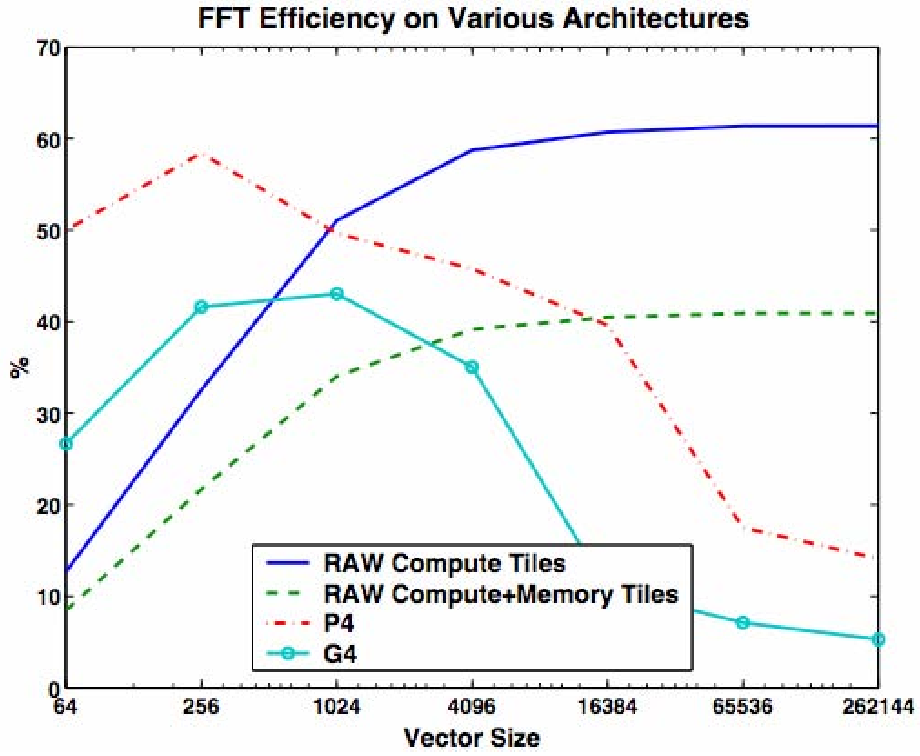
Idea: use a Radix-R FFT algorithm on an R by R array

- A Radix-R FFT algorithm
  - Uses $\log_R N$ stages
  - Compute N/R Radix-R butterflies per stage
- Implement the radix-R butterfly with an R-point DFT
  - W, Q both scale with R for a DFT
  - Allows us to use more processors for each stage
  - Still becomes inefficient as R gets “too large”
  - Efficiency limit for radix-4 algorithm = 56%
  - Efficiency limit for radix-8 algorithm = 54%
- Radix-4 implementation:
  - Distribute a radix-4 butterfly over 4 processors in a row or column
  - Perform 4 butterflies in parallel
  - 8 memory tiles required

Radix-4 FFT Algorithm Performance

Simulated Radix-4 FFT on 4x4 RAW plus 8 memory tiles





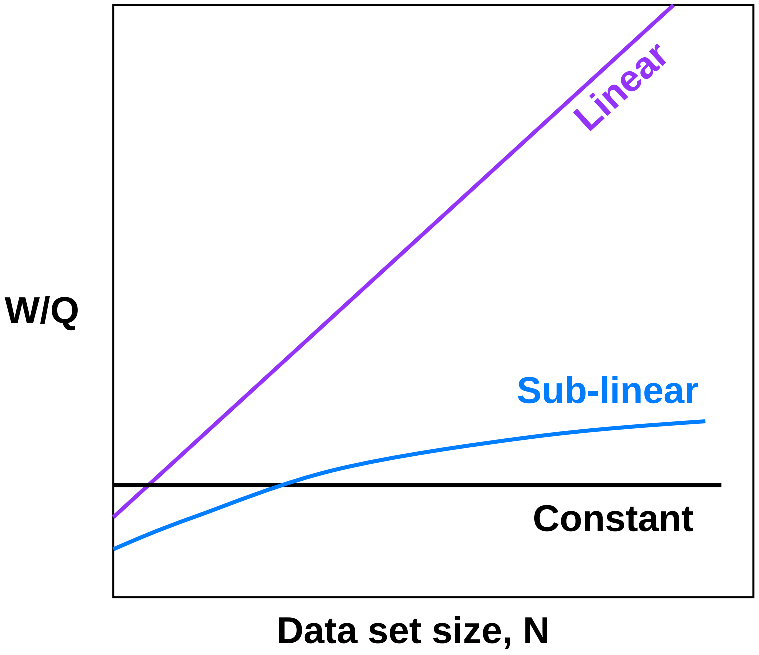
- **Example: Radix-4 FFT algorithm achieves high throughput on 4x4 RAW**
  - **Comparable efficiency to FFTW on G4, Xeon**
- **Raw efficiency stays high for larger FFT sizes**

G4, Xeon FFT results from
`http://www.fftw.org/benchfft`

Classifying Kernels

Kernels may be classified by the ratio W/Q

- Constant Ratio: W = O(N), Q_i = O(N)
  - e.g., Matrix Multiply, QR, Convolution
  - Stream algorithms: efficiency approaches 1 as R, N/R increase
- Sub-Linear Ratio: W=O(N), Q_i < O(N);
  - e.g., FFT
  - Require trade-off between efficiency and scalability
- Linear Ratio: W = O(N), Q_i = O(1);
  - e.g., CFAR
  - Difficult to find efficient or scalable implementation



Examining W/Q gives insight into whether a stream algorithm exists for the kernel

Conclusions

- Stream algorithms map efficiently to tiled arrays
  - Efficiency can approach 100% as data size and array size increase
  - Implementations on RAW simulator show the efficiency of this approach
  - Will be moving implementations from simulator to board
- The communication-to-computation ratio W/Q gives insight into the mapping process
  - A constant W/Q seems to indicate a stream algorithm exists
  - When W/Q is greater than a constant it is hard to efficiently use more processors
- This research could form the basis for a methodology of programming tile arrays
  - More research and formalism required