Mapping Signal Processing Kernels to Tiled Architectures

Henry Hoffmann and James Lebak (presenter)

MIT Lincoln Laboratory[1]

05 May 2004

A key feature of many new computer architectures is that they are composed of multiple tiles, each of which is a fully capable processor. Tiled architectures are attractive alternatives to monolithic computer architecture designs because they allow a larger design to be built from smaller modules and limit the number of wires that need to span the entire chip. Examples of tiled architectures include many under development for the DARPA/IPTO Polymorphous Computer Architectures (PCA) program, including the MIT Raw machine [13], the Stanford Smart memories project [8], and the University of Texas TRIPS machine [9].

The ***decoupled systolic architecture*** (DSA) represents a canonical abstract machine that encompasses many of the key features of single-chip tiled architectures [4] including the PCAs and other emerging architectures such as Scale [6], Wavescalar [12], and Synchroscalar [10]. ***Stream algorithms*** are defined as the family of algorithms which can achieve 100% computational efficiency on the DSA. The DSA and stream algorithms provide a rigorous analytical framework for reasoning about the performance of algorithms on modern architectures. This framework is unique in that it explicitly penalizes algorithmic implementations that make use of long wires and/or large local memories while rewarding those algorithms that can efficiently execute using only a small, bounded amount of local memory and a next-neighbor interconnect network. Thus, this framework makes an excellent match for evaluating architectures faced with the growing physical concerns of wire delay [3] and the energy dissipation of on-chip memory [2, 5].

Stream algorithms are therefore important because the existence of a stream algorithm for a particular problem implies a scalable, computation, energy, and area efficient solution to that problem on many real-world architectures. Stream algorithms decouple memory access from computation, performing memory access on tiles on the periphery of the chip and performing computation in a systolic fashion on the tiles in the interior of the chip. For a problem of size N on an $R \times R$ array of tiles, the efficiency of the problem is the total number of operations $C(N)$ divided by the product of the number of cycles $T(N, R)$ and the total number of memory tiles $M(R)$ and compute tiles $P(R)$,

$$E(N, R) = \frac{C(N)}{T(N, R) * (M(R) + P(R))}. \tag{1}$$

For a conventional architecture, the total number of tiles is equal to 1. A necessary condition for $E(N, R)$ to scale with the size of the array is that $M(R)$ be asymptotically smaller than $P(R)$. An algorithm that meets the requirement that $P(R) = o(M(R))$ is *decoupling efficient*, because it efficiently decouples memory accesses from computation [11]. An algorithm is *computation efficient* if $\lim_{\sigma, R \to \infty} E(\sigma, R) = 1$ where $\sigma = N/R$. Computation efficient algorithms implemented on an array of fixed size scale toward an asymptotic limit on performance as data size increases, and this asymptotic limit becomes larger as the array size R increases. Stream algorithms are therefore those algorithms that meet the computation efficiency condition. Stream algorithms for matrix multiplication, QR factorization, convolution, and other applications have been discovered and implemented on the Raw cycle accurate simulator [4]. Comparison of these algorithms with conventional implementations on conventional architectures such as the PowerPC G4 [7] shows that stream algorithms have the potential to achieve higher efficiency on many different problems.

This presentation focuses on understanding when a stream algorithm exists for a given kernel. We do so by considering the directed acyclic graph (DAG) for a particular implementation of the kernel. Nodes in the DAG represent inputs, outputs, or intermediate products of the algorithm, and edges from node A to node B in the DAG show that A is used to compute B. We can characterize the DAG for an algorithm by the ratio of inputs, W, to the number of intermediate products, Q, for which any one value is directly required. For example, in an algorithm to multiply two $N \times N$ matrices A and B, element i, j of the output matrix C is computed as $c_{i,j} = \sum_{k=1}^{N} a_{ik} b_{kj}$. That is, for each output, there are $W = 2N$ inputs used and a total of $Q = N$ intermediate products (the partial sums) computed. The stream algorithm implementation of matrix multiply meets the compute efficiency condition. Matrix multiplication is an example of a kernel with a *constant ratio* of W to Q. All known algorithms – including QR, SVD, convolution – with a constant ratio of W to Q have implementations that meet the compute efficiency condition.

In contrast, consider an algorithm to compute the FFT of a length-N vector. To compute any particular output of the FFT, all N inputs are required, and (as is well known) each input directly contributes to $\log_2(N)$ intermediate products.

Report Documentation Page

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 FEB 2005	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Mapping Signal Processing Kernels to Tiled Architectures	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
MIT Lincoln Laboratory1	

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

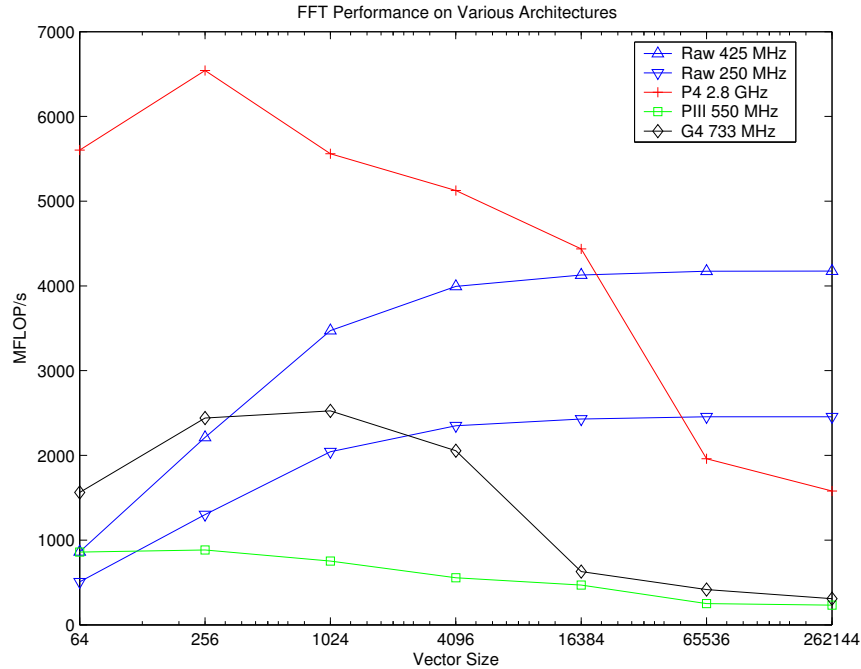| 16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**27**	

Figure 1: Comparison of Raw FFT throughput, measured in the Raw cycle-accurate simulator, to FFTW throughput on the PowerPC G4 and Xeon.

For the radix-2 FFT algorithm, the ratio of $W = O(N)$ to $Q = O(\log_2(N))$ is asymptotically *greater* than a constant. Because any stream algorithm for the FFT must meet the decoupling efficiency condition, we cannot use local memory to buffer the large number of inputs. Instead inputs must reside in the network while compute tiles are working. For the FFT, with a $W/Q = O(N)/O(\log_2(N))$, this implies that the maximum distance that any piece of data must travel is greater than the number of intermediate calculations in which the data is used. Therefore, communication costs cannot be effectively amortized in the systolic implementation on a tiled architecture. The factor T in the denominator of the efficiency expression (1) will have a lower bound that is limited by the size of the array, meaning that the efficiency cannot approach a limit of 1 as the array size R increases. A stream algorithm implementation for the FFT is still an open research problem. Stream algorithm techniques can be used to implement an efficient implementation of the radix-4 FFT for a 4x4 tile array, but this implementation is not scalable and performance will be worse on larger Raw systems. Simulated throughput of this algorithm is compared to the throughput of FFTW [1] on the 2.8 GHz Pentium 4 and 733 MHz G4 in Figure 1. The Raw FFT achieves high performance for large data sizes, and offers performance that is more stable across a range of data sizes.

In this talk, we will describe the implementation of FFT, QR factorization, and CFAR kernels on the Raw simulator and Raw board. We examine the performance of these kernels and compare to conventional implementations on the Pentium and G4 architectures. Finally, we characterize the DAG of each kernel and discuss how the DAG influences the implementation on Raw and on tiled architectures in general.

References

[1] Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1381–1384. IEEE Signal Processing Society, May 1998.

[2] Ricardo Gonzalez and Mark Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, September 1996.

[3] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[4] Henry Hoffmann. Stream Algorithms and Architecture. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2003.

[5] Jason Sungtae Kim, Michael B. Taylor, Jason Miller, and David Wentzlaff. Energy Characterization of a Tiled Architecture Processor with On-Chip Networks. In *ACM Symposium on Low Power Electronics and Design*, pages 424–427, Seoul, Korea, August 2003.

[6] Ronny Krashinsky, Christopher Batten, Mark Hampton, Steve Gerding, Brian Pharris, Jared Casper, and Krste Asanovic. The Vector-Thread Architecture. In *31st International Symposium on Computer Architecture*, München, Germany, June 2004. (publication pending).

[7] James Lebak, Hector Chan, Ryan Haney, and Edmund Wong. Polymorphous computing architectures (PCA) kernel benchmark measurements on the PowerPC G4. Project Report PCA-Kernel-2, MIT Lincoln Laboratory, Lexington, MA, January 2004.

[8] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, and Mark Horowitz. Smart Memories: A Modular Reconfigurable Architecture. In *28th Annual International Symposium on Computer Architecture*, pages 161–171, June 2000.

[9] Ramdass Nagarajan, Karthikeyan Sankaralingam, Doug C. Burger, and Steve W. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *34th Annual International Symposium on Microarchitecture*, pages 40–51, December 2001.

[10] John Oliver, Ravishankar Rao, Paul Sultana, Jedidiah Crandall, Erik Czernikowski, Leslie W. Jones IV, Dean Copsey, Diana Keen, Venkatesh Akella, and Frederic T. Chong. Synchroscalar: A Multiple Clock Domain Power-Aware Tile-Based Embedded Processor. In *31st International Symposium on Computer Architecture*, München, Germany, June 2004. (publication pending).

[11] James E. Smith. Decoupled Access/Execute Computer Architectures. *ACM Transactions on Computer Systems*, 2(4):289–308, November 1984.

[12] Steven Swanson, Ken Michelson, Andrew Schwerin, and Mark Oskin. WaveScalar. In *36th International Symposium on Microarchitecture*, pages 291–302, San Diego, CA, December 2003. IEEE Computer Society.

[13] Michael B. Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro*, 22(2):25–36, March/April 2002.

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

MIT Lincoln Laboratory

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)

8x8x16 Grid Processors
Configurable memory banks
Memory/IO channels

L1 I-cache banks
Stitching/Move logic
Register file bank

Level 1 data miss control
Memory consistency queues
L1 D-cache banks

Network/routing logic

DRAM arrays

Branch predictor/ I-fetch control
Block control
Express channels
Compute node (ALU)

HPC applications
DSP streaming applications
Threaded server applications
C desktop applications

RAW (MIT)

PC
I $
D$
ALU
RF
X

- **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**

Smart Memories (Stanford)
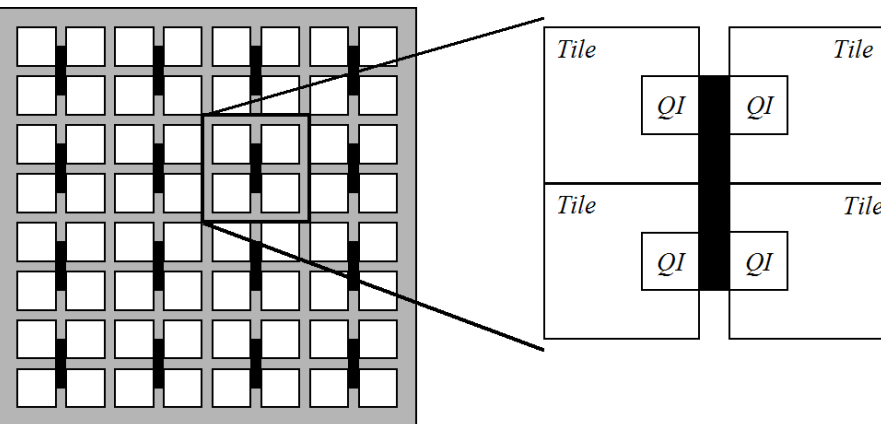
Tile
QI
QI
Tile
Tile
QI
QI
Tile

Outline

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

Stream Algorithms for Tiled Architectures

Decoupled Systolic Architecture

Time **Space**

Stream Algorithm Efficiency:

$$E(N,R) = \frac{C(N)}{T(N,R)*(P(R) + M(R))}$$

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: $\lim\limits_{\sigma, R \to \infty} E(\sigma, R) = 1$
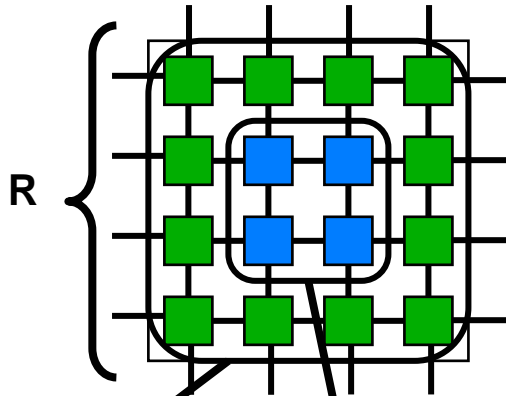
where $\sigma = N/R$

M(R) edge tiles are allocated to memory management

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

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

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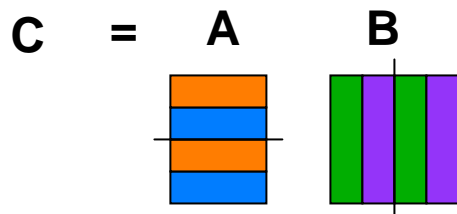
- **In each phase, compute R^2 elements of C**
  - **Involves 2N operations per tile**
  - **N^2/R^2 phases**

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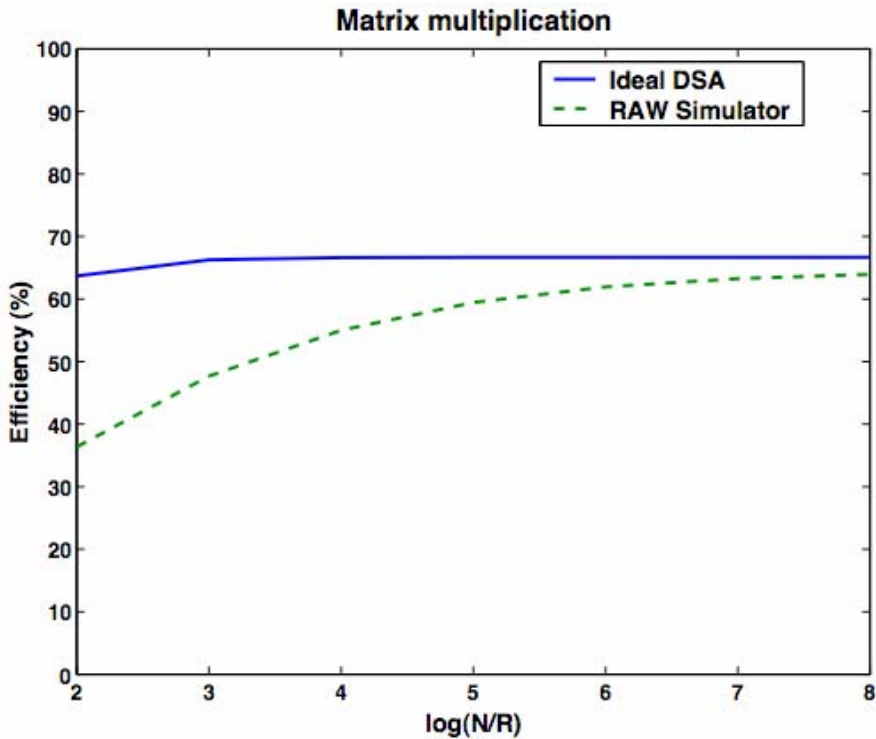 - **R cycles to output the result**

Memory tiles

Compute tiles
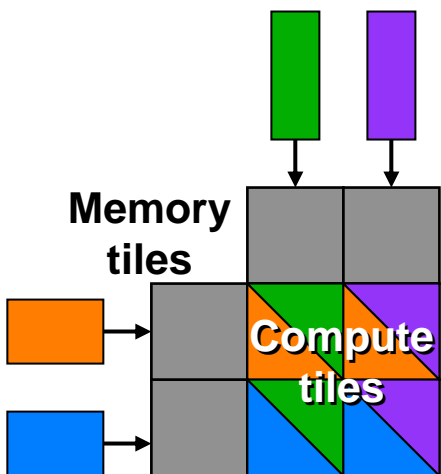
Efficiency Calculation:

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

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

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

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

Matrix Multiply Efficiency

Matrix multiplication

Legend:
- Ideal DSA
- RAW Simulator

X-axis: log(N/R), Y-axis: Efficiency (%)

Scaled overall efficiency

Legend:
- N/R = 16
- N/R = 64
- N/R = 256

X-axis: Compute Processors, Y-axis: Efficiency (%)

Assume a 4x4 decoupled systolic architecture or RAW surrounded by memory tiles (max efficiency=66%)
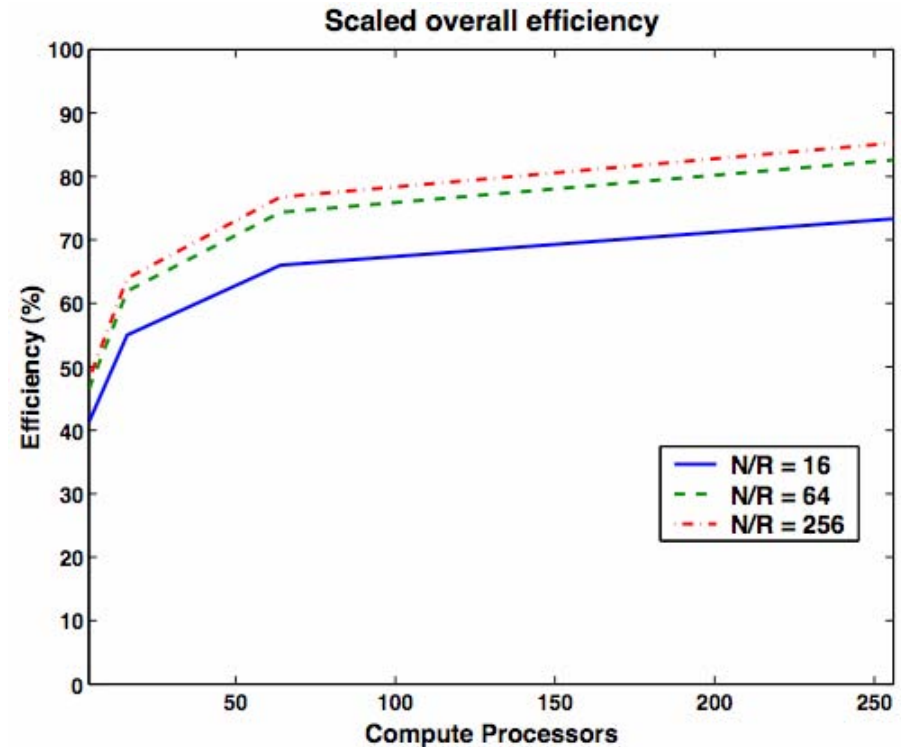
Scale the number of overall tiles Smaller percentage of tiles devoted to memory leads to higher efficiency

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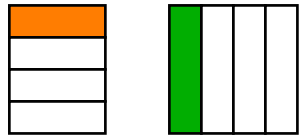
To compute cij, row i of A is multiplied by column j of B
- **2N inputs required**
- **2N operations required**

a11
a12
a21
a22
b11
b21
b12
b22

c11
c22
c21
c12
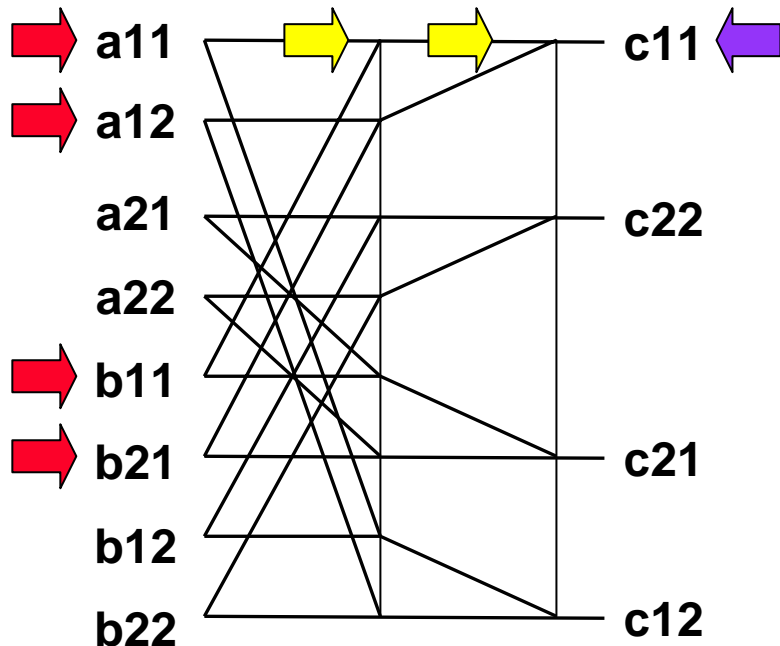
- **Examine the directed acyclic graph (DAG) for the matrix multiply**
- **For each output** ⬅ **produced**
- **There are W inputs** ➡ **required (O(N))**
- **The input i is used** ➡ **Q_i times (O(N))**
  - **These are intermediate products**
- **The matrix multiply is an example of an algorithm with 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:
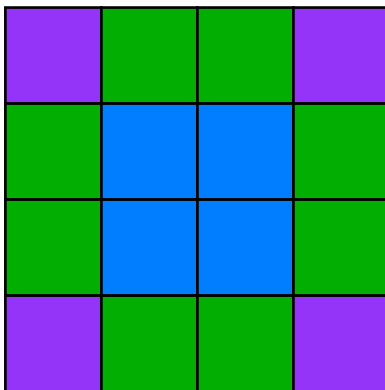
I/O tiles
- **Stream data to and from outside world**

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
```
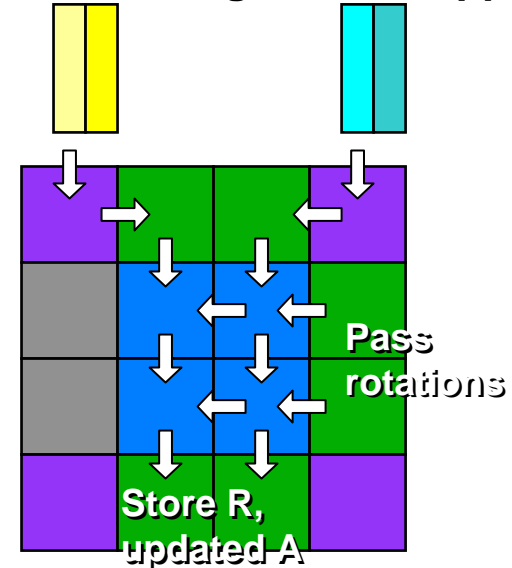
For a matrix A with six columns:

Column block   **1**   **2**   **3**

Data flow during rotation computation
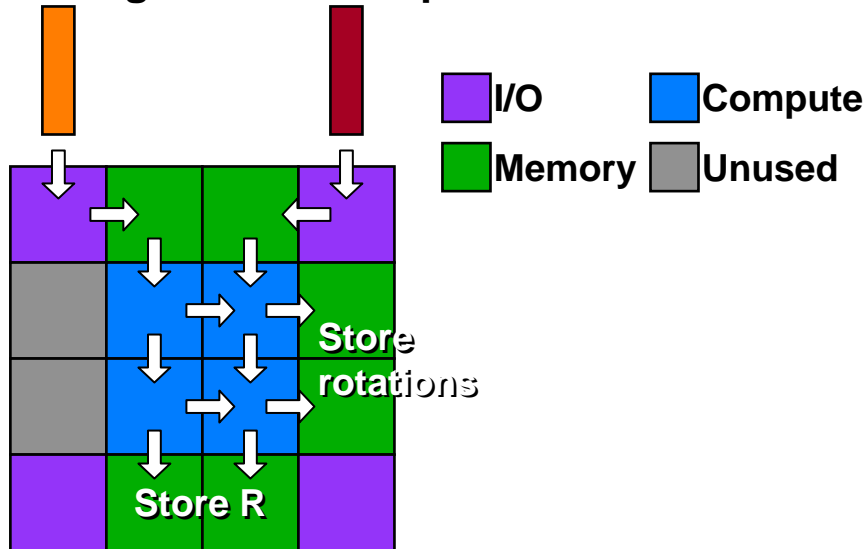
Data flow during rotation application

- **I/O**
- **Memory**
- **Compute**
- **Unused**

Store rotations

Store R

Pass rotations

Store R, updated A

- **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

Simulated RAW efficiency for Complex QR on Square Matrices

Efficiency (%) vs Matrix rows and columns, N

- Compute tiles
- Overall

N_{80}
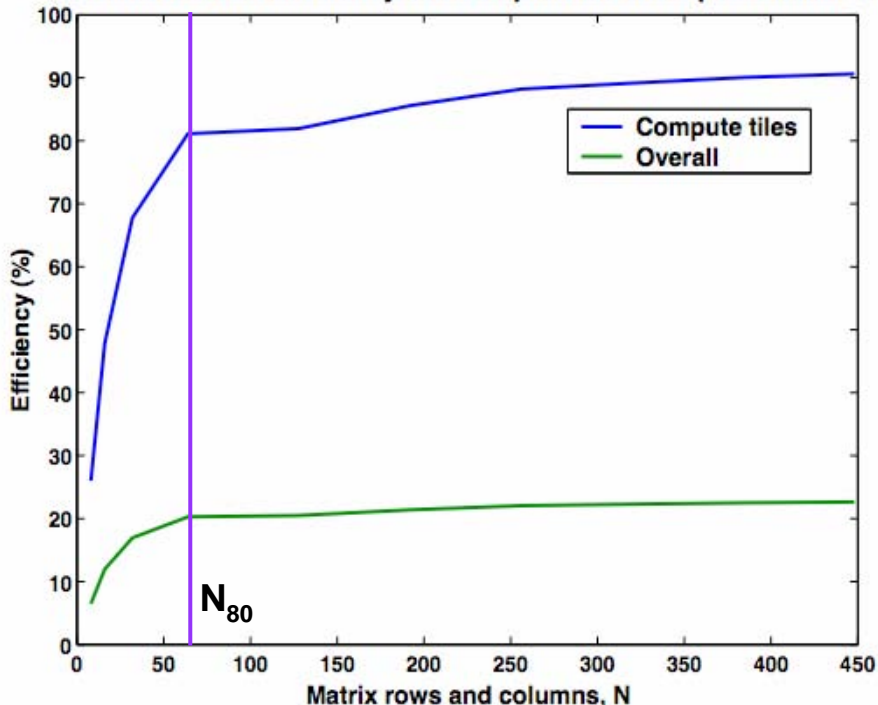
Projected Asymptotic Efficiency on Scaled Versions of RAW

Efficiency (%) vs Array dimension, R
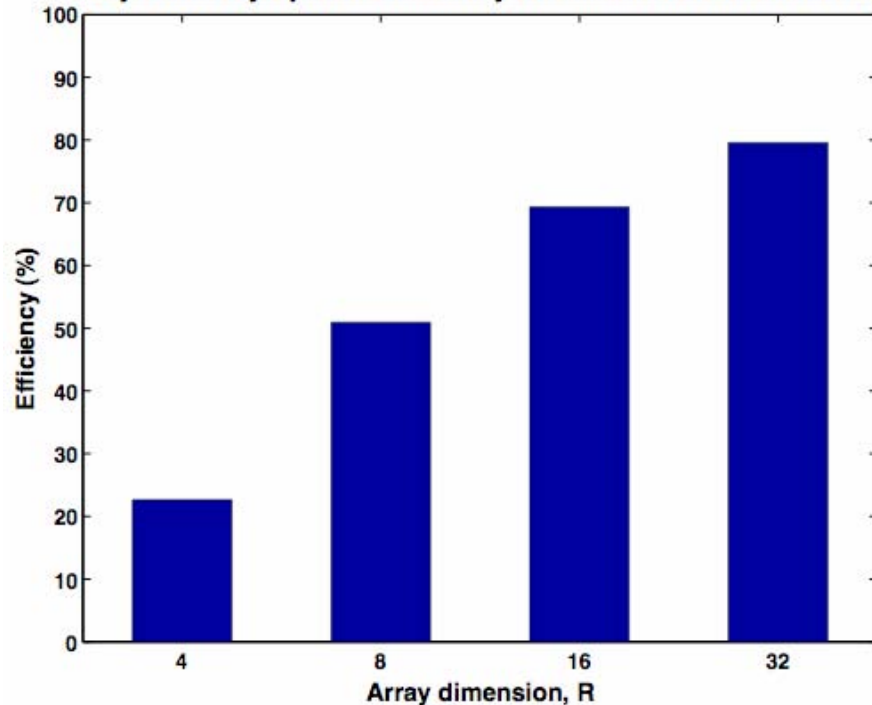
The QR factorization has a *constant ratio* of input data (W) to intermediate products (Q)

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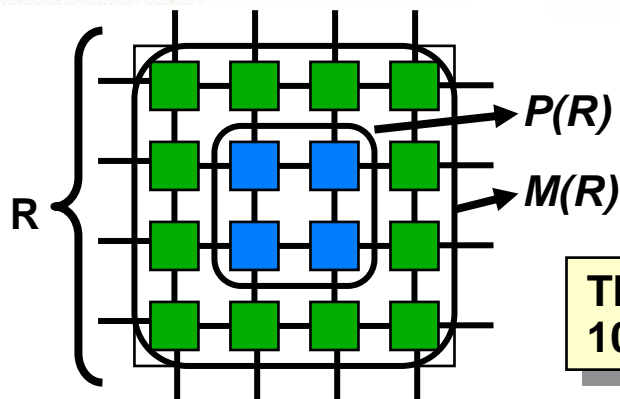
Projected matrix size N_{80} to achieve 80% efficiency on compute tiles $P(R)$:

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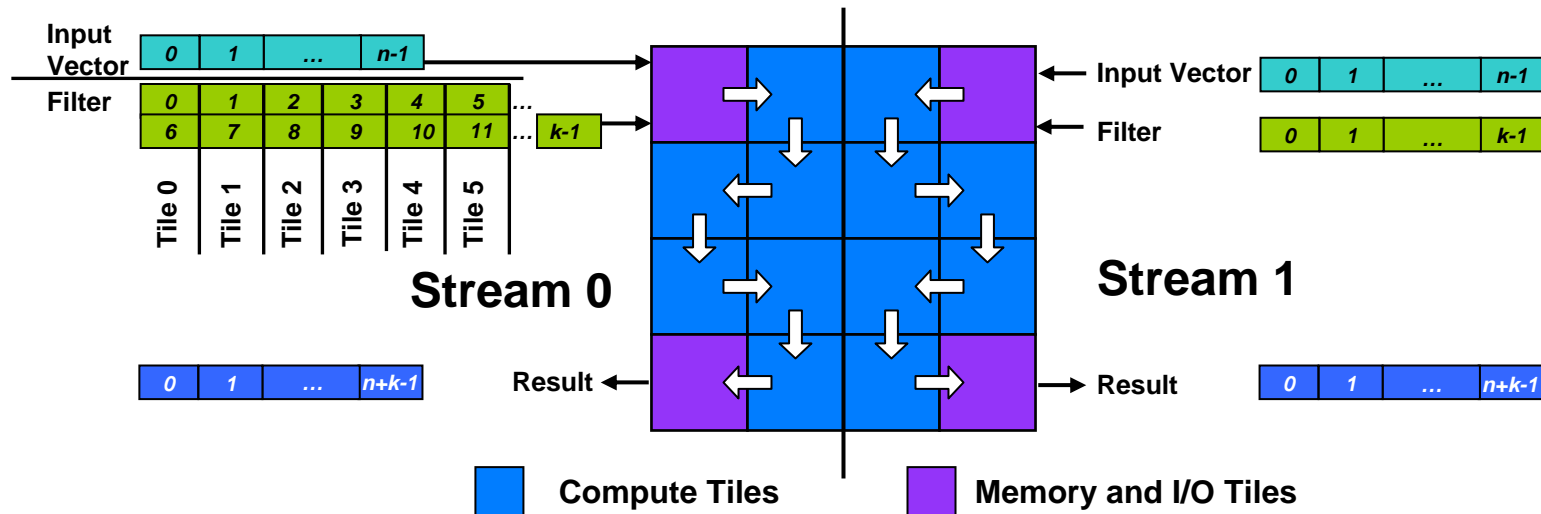
- **Filter coefficients distributed cyclically to tiles**
  - **Each compute tile convolves the input with a subset of the filter**
  - **Assume n (data length) > k (filter length)**
- **Each stream is a different convolution operation**
  - **In multichannel signal processing applications we rarely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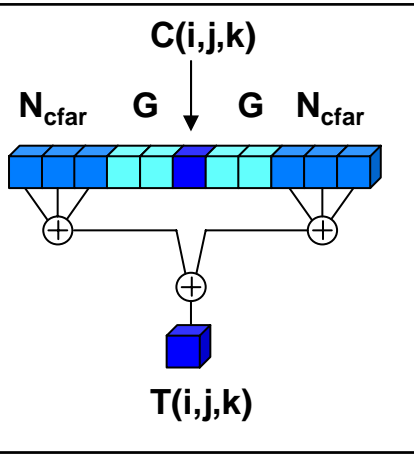- **12 of 16 tiles used for computation**
  - **Maximum 75% efficiency**

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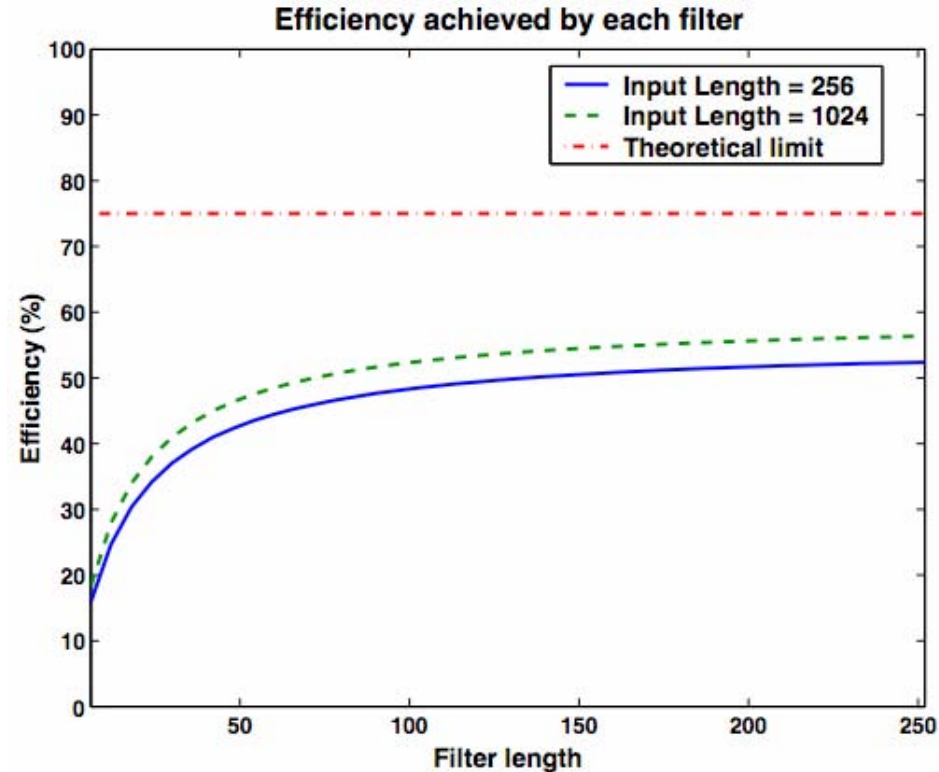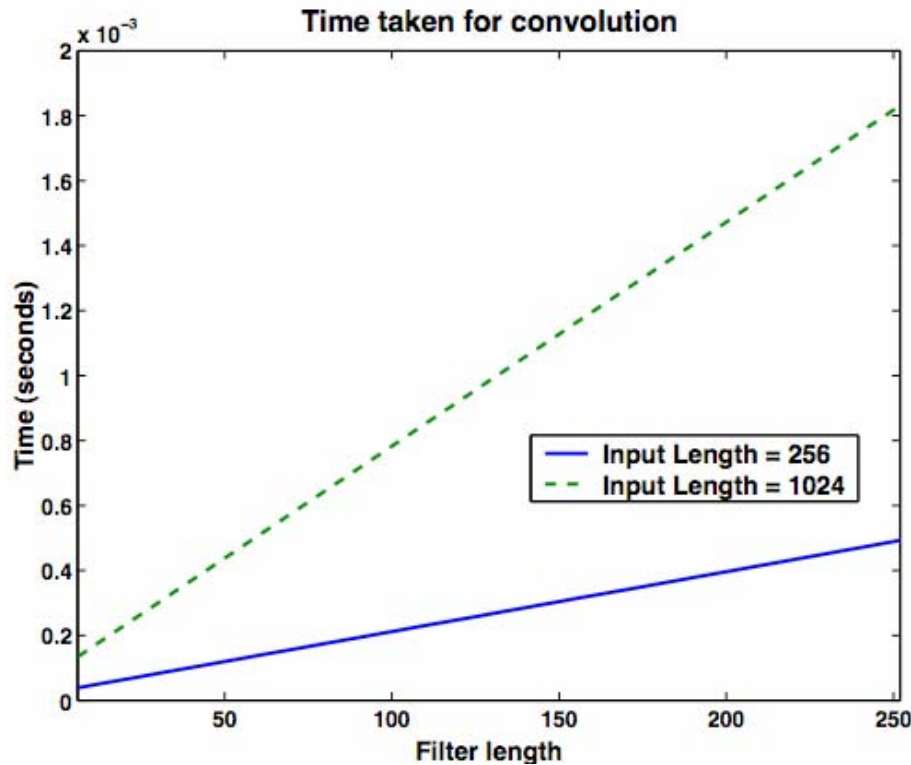
- **Convolution achieves good performance in RAW simulator**
- **Longer filters and input vectors are more efficient**
- **Longer input vectors are also more easily mapped to more processors**

CFAR Mapping

$C(i,j,k)$

N_{cfar} G G N_{cfar}

$T(i,j,k)$

- **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
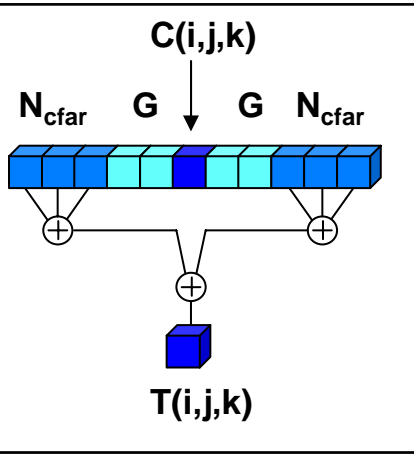
C(i,j,k)

N_{cfar} **G**   **G** N_{cfar}

T(i,j,k)

- **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**

N_{rg} Range Gates

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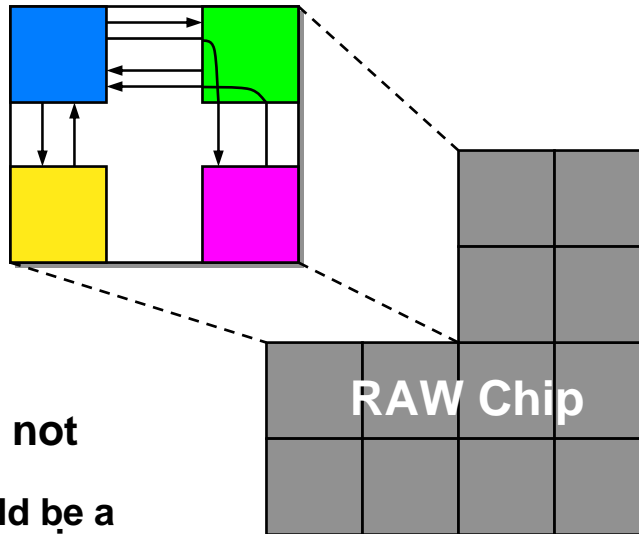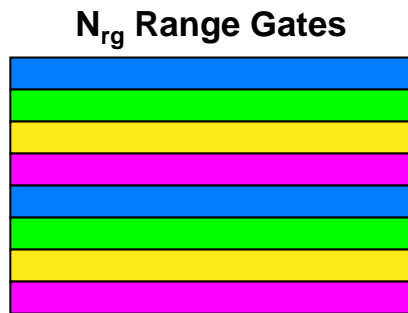
- **Data cube is streamed into RAW using the static network**
- **Corner input ports receive data**
- **Each quadrant processes data from one port**
- **One row of range data ("one stream") is processed by a single tile**
- **Results gathered to corner tile and output**

- **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**
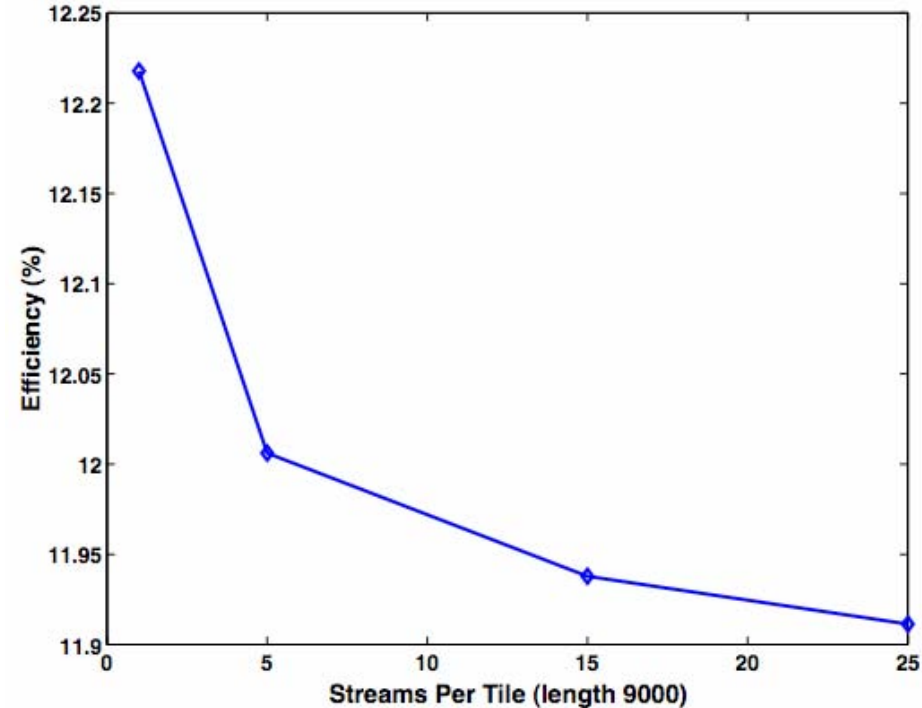
CFAR Performance

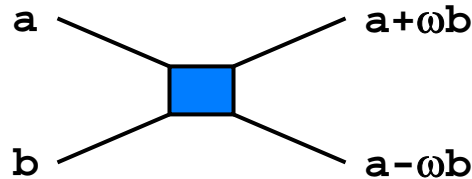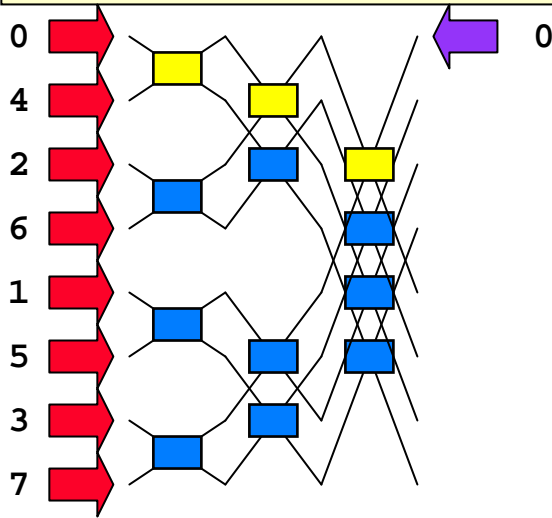
Stream fits in cache
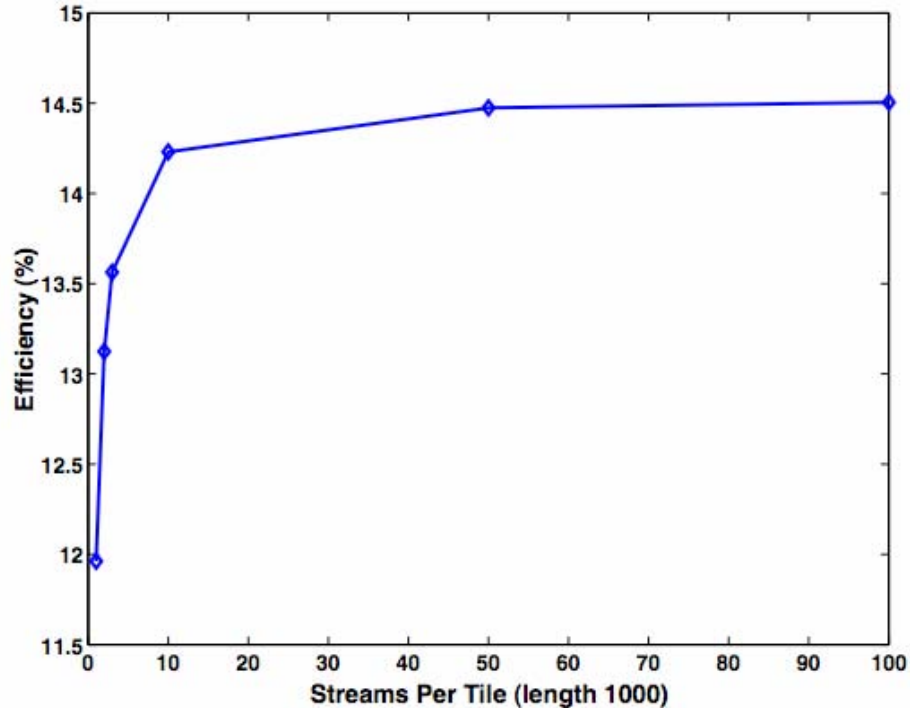
Stream does not fit in cache

- **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 (log2N) stages`
`  compute N/2 "butterflies"`

Radix-2 butterfly diagram: inputs a, b; outputs $a + \omega b$, $a - \omega b$

Radix-2 butterfly:
- **2 complex inputs**
- **precomputed weight ω**
- **10 real operations**

Data flow diagram with inputs 0, 4, 2, 6, 1, 5, 3, 7 and output 0

- **For each output** [purple arrow] **produced**
- **There are W inputs** [red arrow] **required (O(N))**
- **The input i is used** [yellow box] **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

Stage 1:

```
0   4
1   5
2   6
3   7
```

6 2 4 0 → ┌─────────┐
           │ 0 │ 4 │
           └─────────┘
7 3 5 1 → ┌─────────┐
           │ 1 │ 5 │
           └─────────┘

Stage 2:

```
3   7
1   5
2   6
0   4
```

0 4 1 5 ← ┌────┬────┐
           │ 0 │ 4 │
           │ 2 │ 6 │
2 6 3 7 ← └────┴────┘

Stage 3:

```
0   4
2   6
1   5
3   7
```

5 4 1 0 → ┌─────────┐
           │ 0 │ 1 │
           └─────────┘
7 6 3 2 → ┌─────────┐
           │ 2 │ 3 │
           └─────────┘
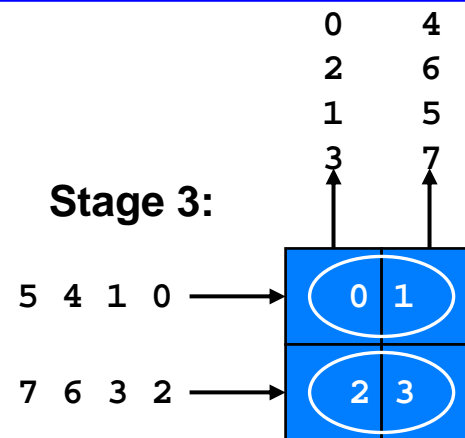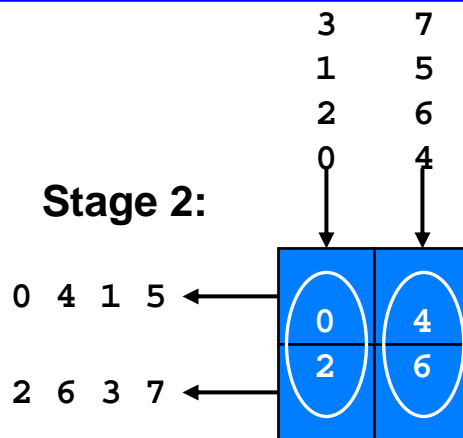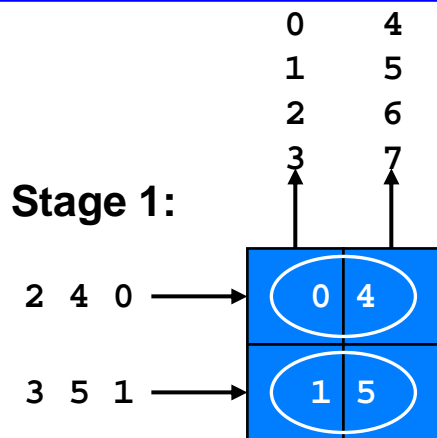
- **For each butterfly:**
    - **4 + (R-1) cycles to clock inputs across the array**
    - **10/R computations per tile**
    - **When R=2, tiles are used efficiently**
        - **Can overlap computation (5 cycles) and communication (5 cycles)**
    - **When R>2, cannot use tiles efficiently**
        - **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

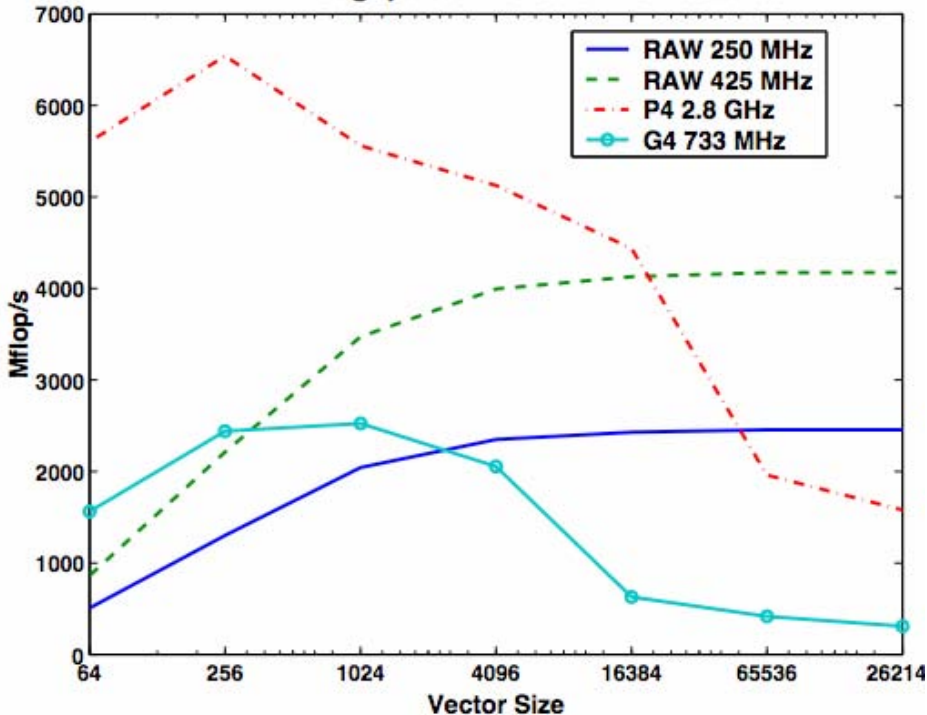**Idea: use a Radix-R FFT algorithm on an R by R array**

- **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
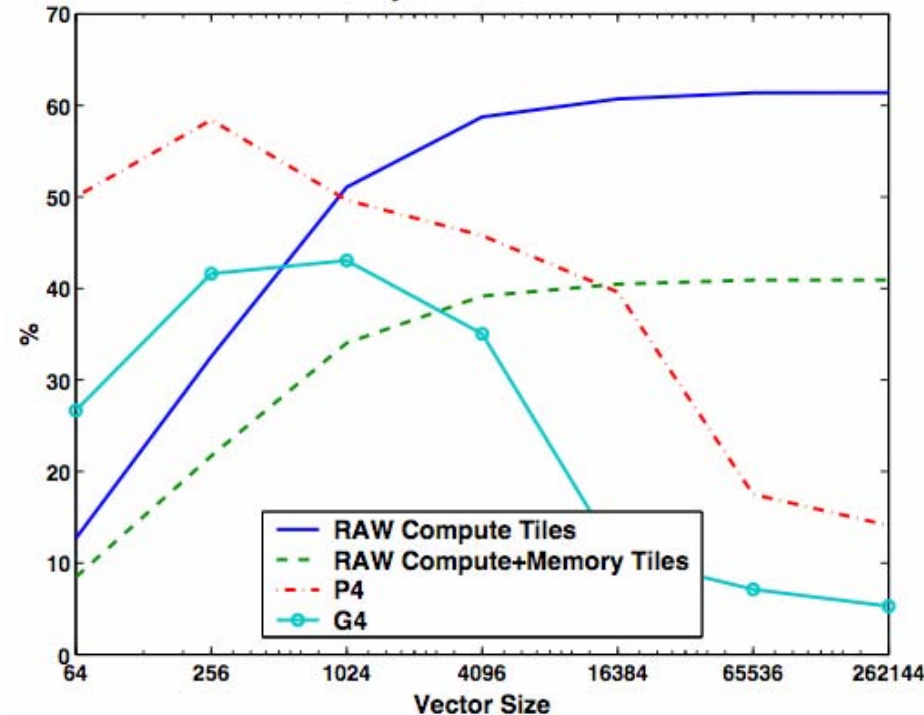 - 8 memory tiles required

Radix-4 FFT Algorithm Performance

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

FFT Throughput on Various Architectures

Legend:
- RAW 250 MHz
- RAW 425 MHz
- P4 2.8 GHz
- G4 733 MHz

Y-axis: Mflop/s (0 to 7000)
X-axis: Vector Size (64, 256, 1024, 4096, 16384, 65536, 26214)

FFT Efficiency on Various Architectures

Legend:
- RAW Compute Tiles
- RAW Compute+Memory Tiles
- P4
- G4

Y-axis: % (0 to 70)
X-axis: Vector Size (64, 256, 1024, 4096, 16384, 65536, 262144)

- **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

W/Q

Linear

Sub-linear

Constant

Data set size, N
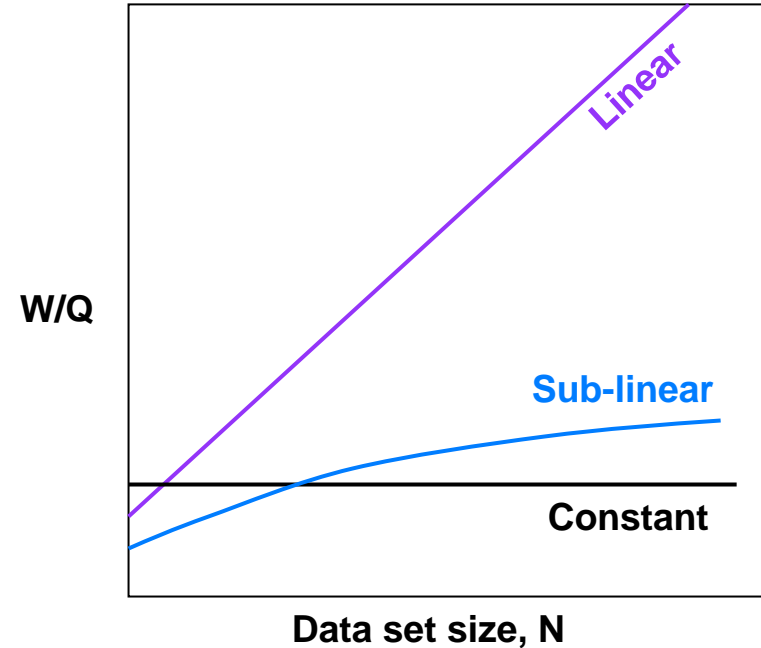
Examining W/Q gives insight into whether a stream algorithm exists for the kernel

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**