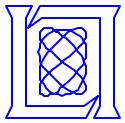


Kernel Benchmarks and Metrics for Polymorphous Computer Architectures

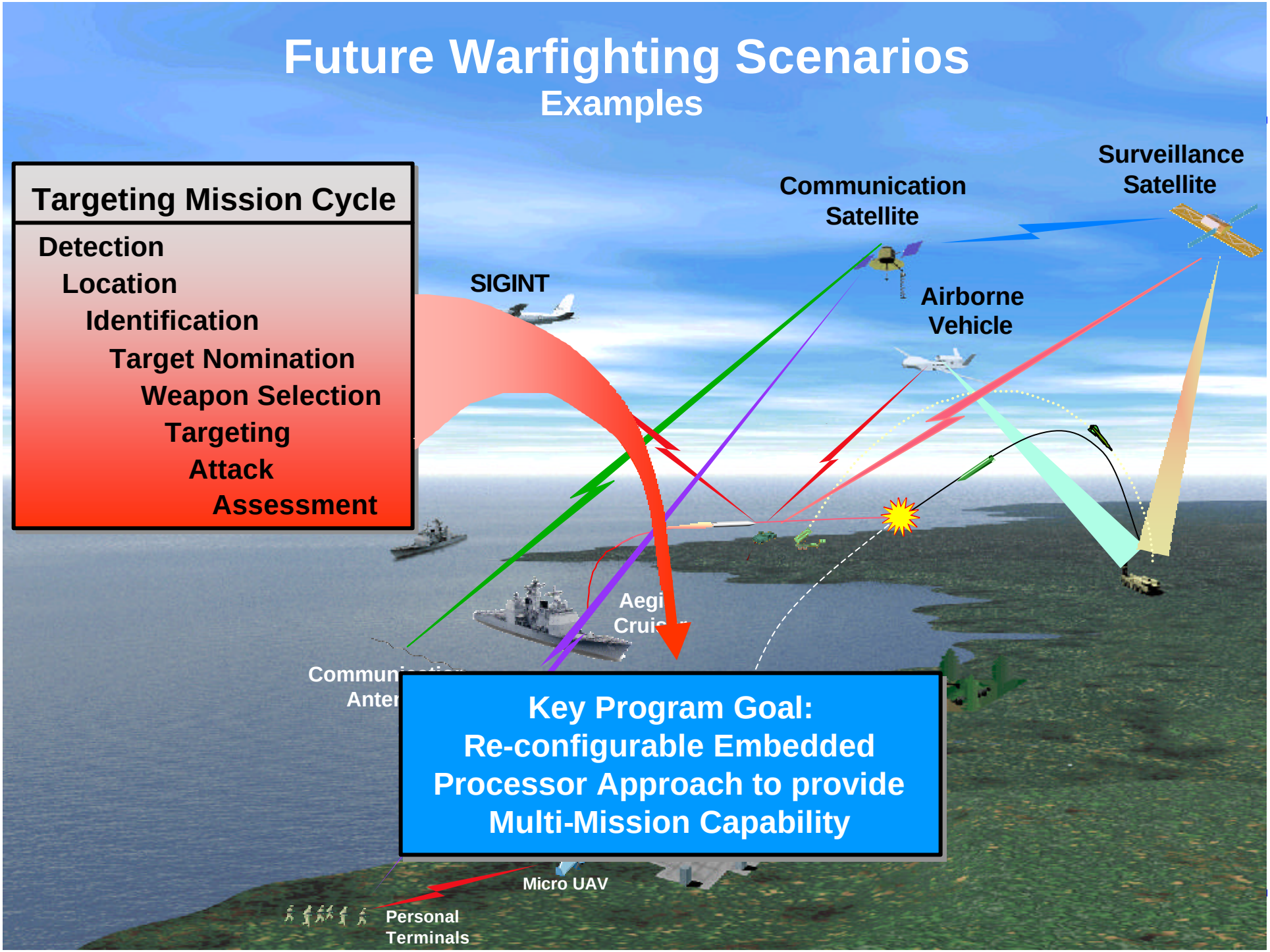
Hank Hoffmann
James Lebak (Presenter)
Janice McMahon
MIT Lincoln Laboratory

Seventh Annual High-Performance Embedded Computing Workshop (HPEC)

24 September 2003

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.

Future Warfighting Scenarios
Examples



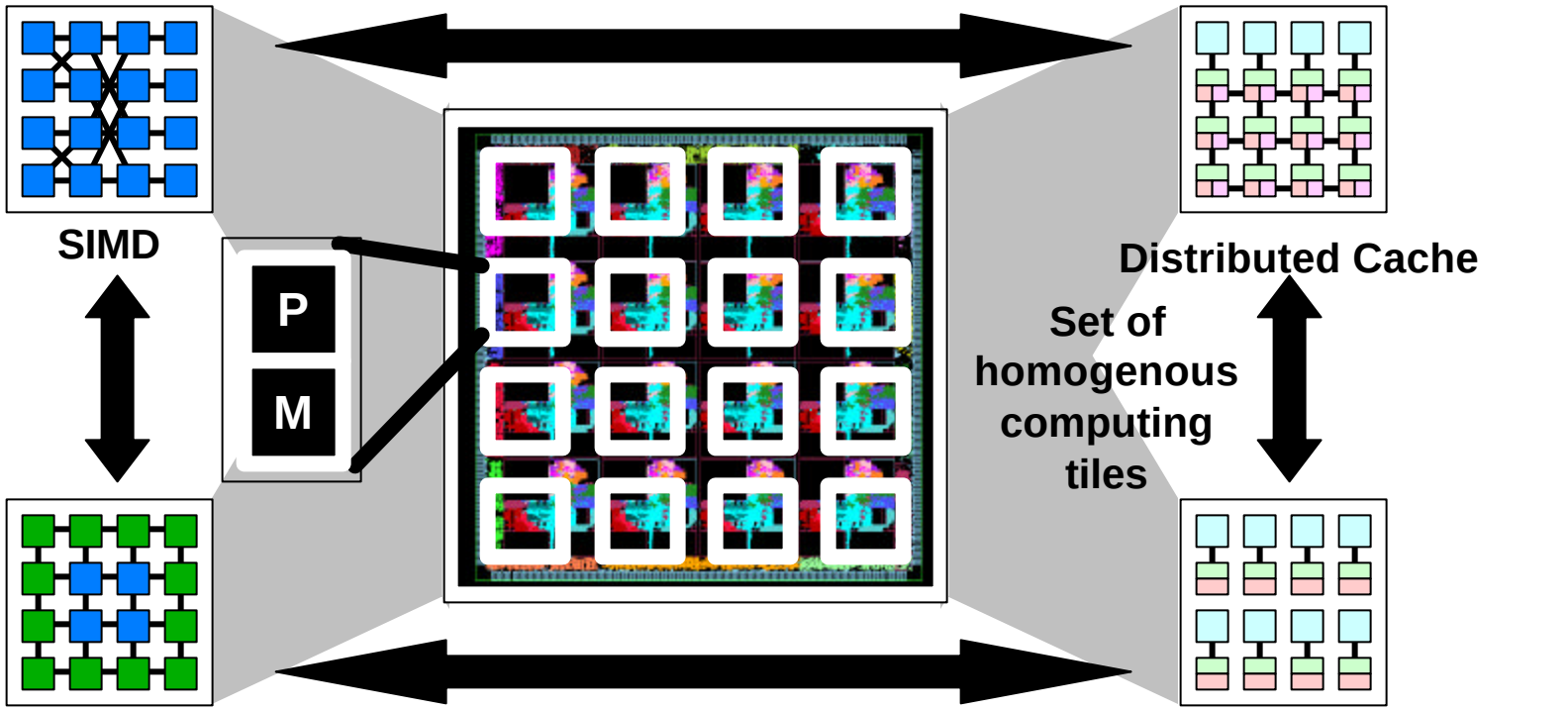
Polymorphous Computing

Stream processing

- Regular, deterministic operations
- Constant flow of input data

Threaded processing

- Complex operations
- Dynamic data movement

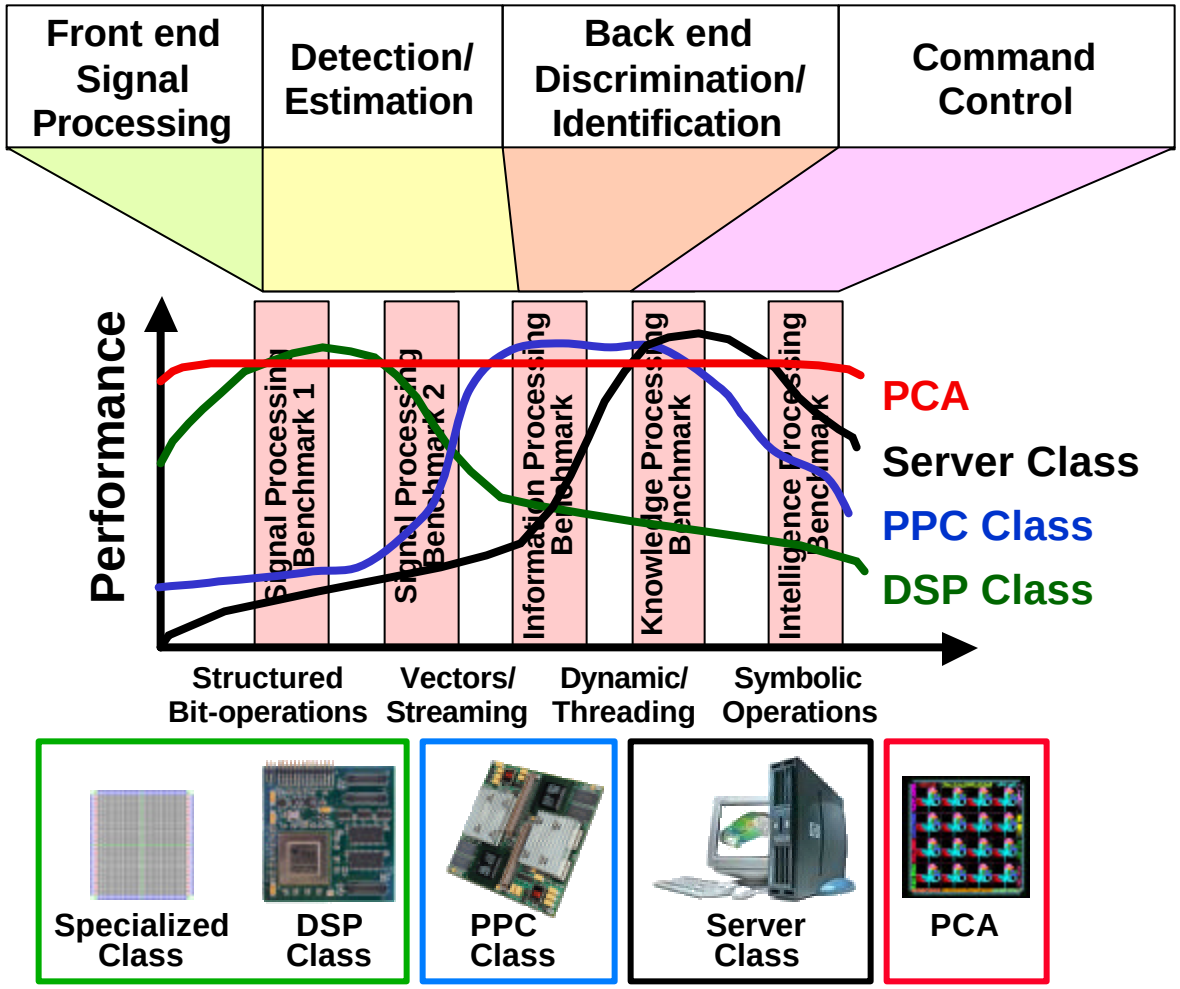


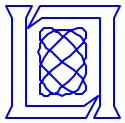
[1]morph \'mor-()f\ *n* : re-structuring of tiles for optimized processing

[2]morph \'mor-()f\ *vt* : to re-structure tiles for optimized processing

Architectural Flexibility

Radar Processing Flow





Outline

- Introduction
- Kernel Benchmarks and Metrics
- Programming PCA Architectures
- Case Study: SVD Kernel
- Conclusions

Kernel Synthesis from Application Survey

Specific Application Areas

Radar | Sonar | Infrared | Hyper-Spectral | SIGINT | Communication | Data Fusion

Broad Processing Categories

"Front-end Processing"

- Data independent, stream-oriented
- Signal processing, image processing, high-speed network communication
- Examples:
  - pulse compression
  - adaptive beamforming
  - target detection

"Back-end Processing"

- Data dependent, thread oriented
- Information processing, knowledge processing
- Examples:
  - workload optimization
  - target classification

Specific Kernels

Signal/Image Processing	Communication	Information/Knowledge Processing
• FIR Filter • SVD • CFAR Detection	• Corner Turn	• Graph Optimization • Pattern Recognition • Real-time Database Operations

MIT-LL Surveyed DoD Applications to Provide:

- Kernel Benchmark Definitions
- Example Requirements and Data Sets

Kernel Performance Evaluation

Kernel Benchmarks

Signal/Image Processing
- FIR Filter
- SVD
- CFAR Detection

Communication
- Corner Turn

Information/Knowledge Processing
- Graph Optimization
- Pattern Recognition
- Real-time Database Operations

Performance Metrics

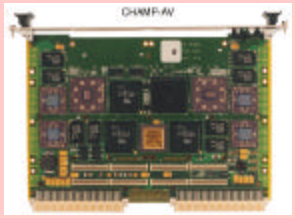
- Floating point and integer ops
- Latency
- Throughput
- Efficiency
- Stability
- Density and cost
  - –Size
  - –Weight
  - –Power

Definitions

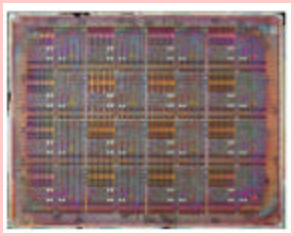
$$\frac{\text{Workload (FLOPS or OPS)}}{\text{Execution time (seconds)}}$$

$$\frac{\text{Throughput}}{\text{Hardware Peak}}$$

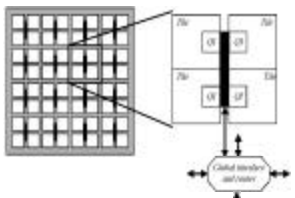
$$\frac{\text{MIN(Throughput)}}{\text{MAX(Throughput)}}$$



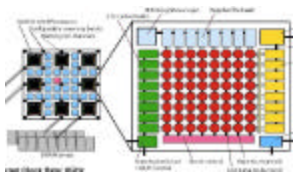
PowerPC(G4)



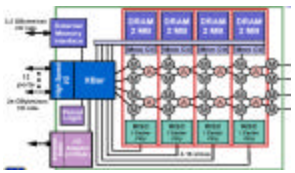
RAW



Smart Memory



TRIPS



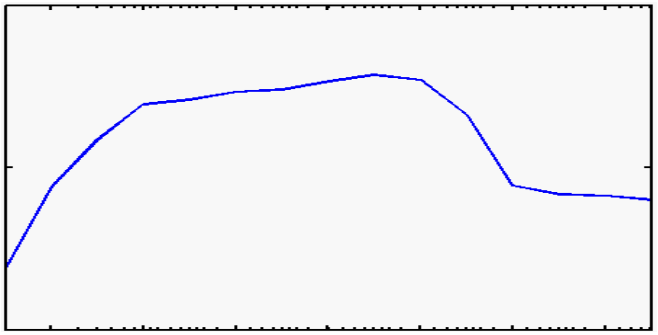
MONARCH

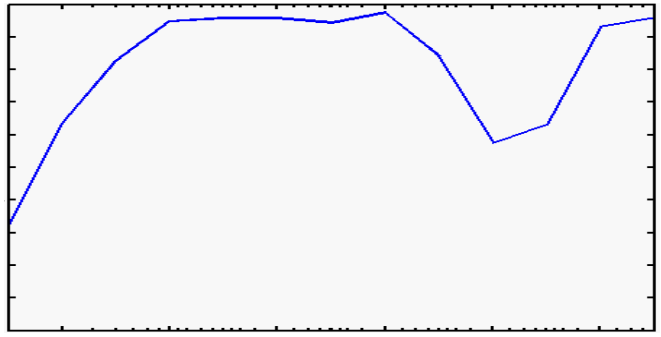
Throughput-Stability Product

A New Kernel Metric

Throughput

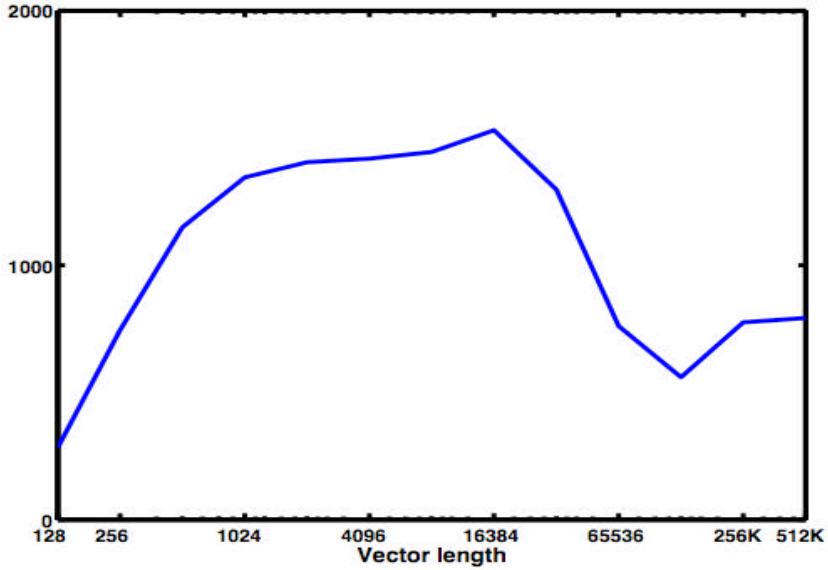
$$\frac{\text{Workload (FLOPS or OPS)}}{\text{Execution time (seconds)}}$$





Interval Stability

$$\frac{\text{MIN}_I(\text{Throughput})}{\text{MAX}_I(\text{Throughput})}$$



Throughput x *Stability*

- rewards consistent high performance
- penalizes lack of performance or lack of consistency

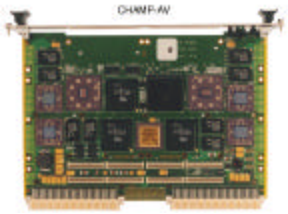
For a given application, PCA processors should achieve higher product of throughput and stability than conventional processors

Outline

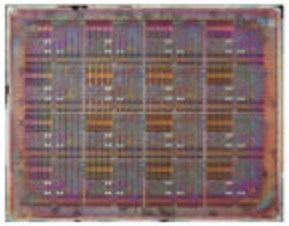
- Introduction
- Kernel Benchmarks and Metrics
- Programming PCA Architectures
- Case Study: SVD Kernel
- Conclusions

High Performance Programming: Conventional vs. PCA Processors

PowerPC(G4)



Raw



Characteristics:

- **Rigid** memory hierarchy
- **Rigid** datapath
- **Specialized** Structures

High Performance Programming:

- Change algorithm to match memory hierarchy
- One degree of freedom
- Can only work with blocking factor

Characteristics:

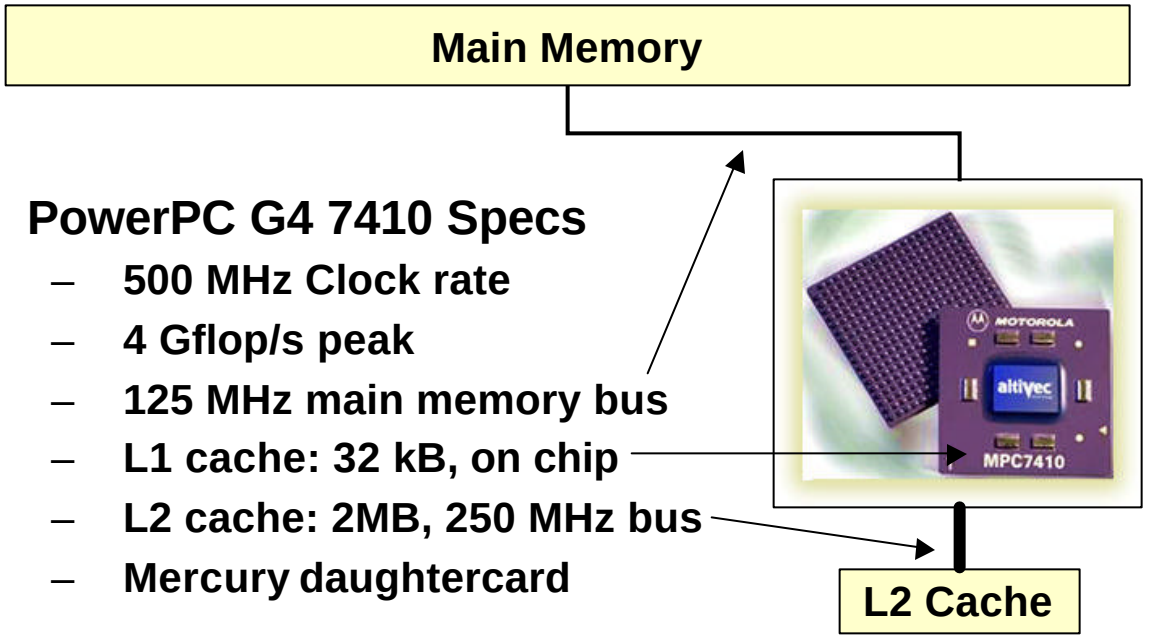
- **Flexible** memory hierarchy
- **Flexible** datapath(s)
- **Generic** Structures

High Performance Programming:

- Co-optimize algorithm and architecture
- Many degrees of freedom
- Optimize time/space tradeoff

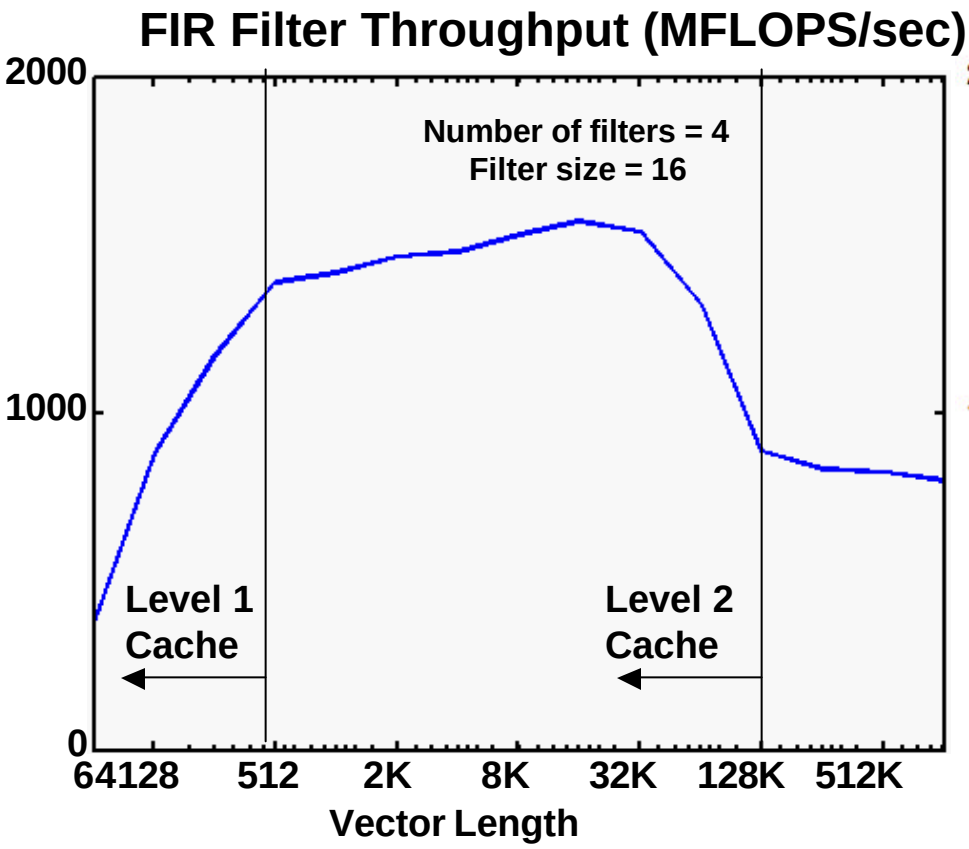
PCA provides more degrees of freedom, and thus greater flexibility (morphability) and greater performance over a range of applications

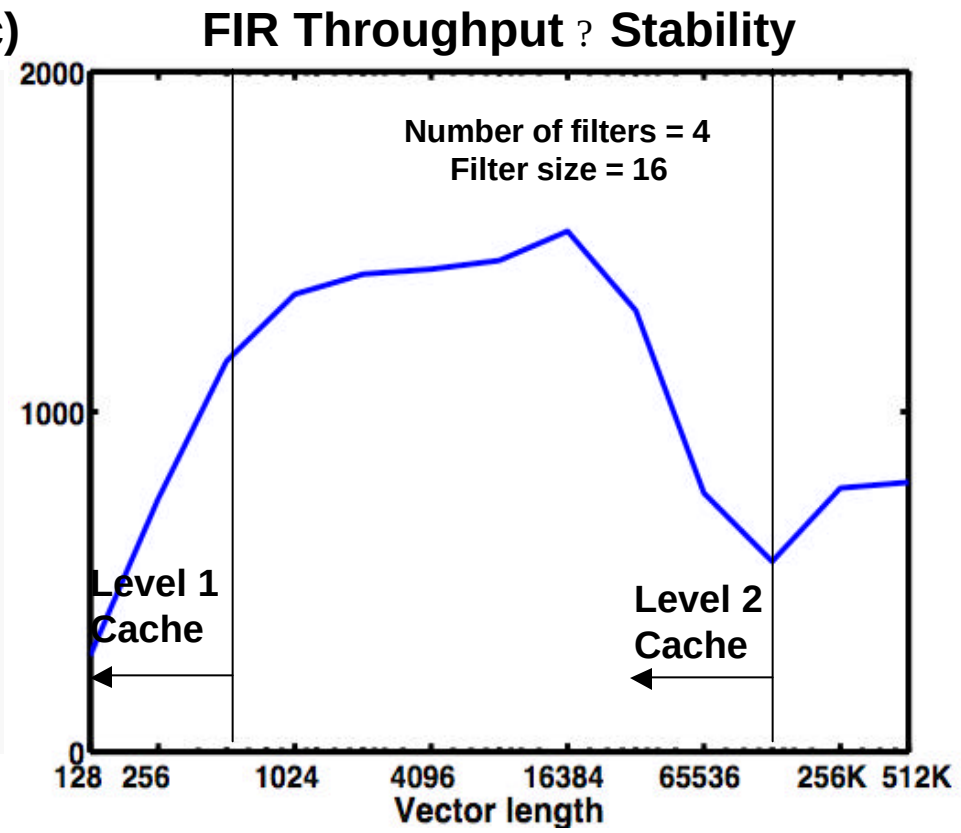
Kernel Benchmarks and the PowerPC G4



- Two predictors of kernel performance:
  - Programmer's maximization of data reuse and locality (blocking factor)
  - Memory hierarchy of G4
- Blocking factor determines max achieved performance
- Memory hierarchy determines shape of performance curve
- Want to maximize blocking factor to limit memory hierarchy bottleneck

FIR Filter (G4)





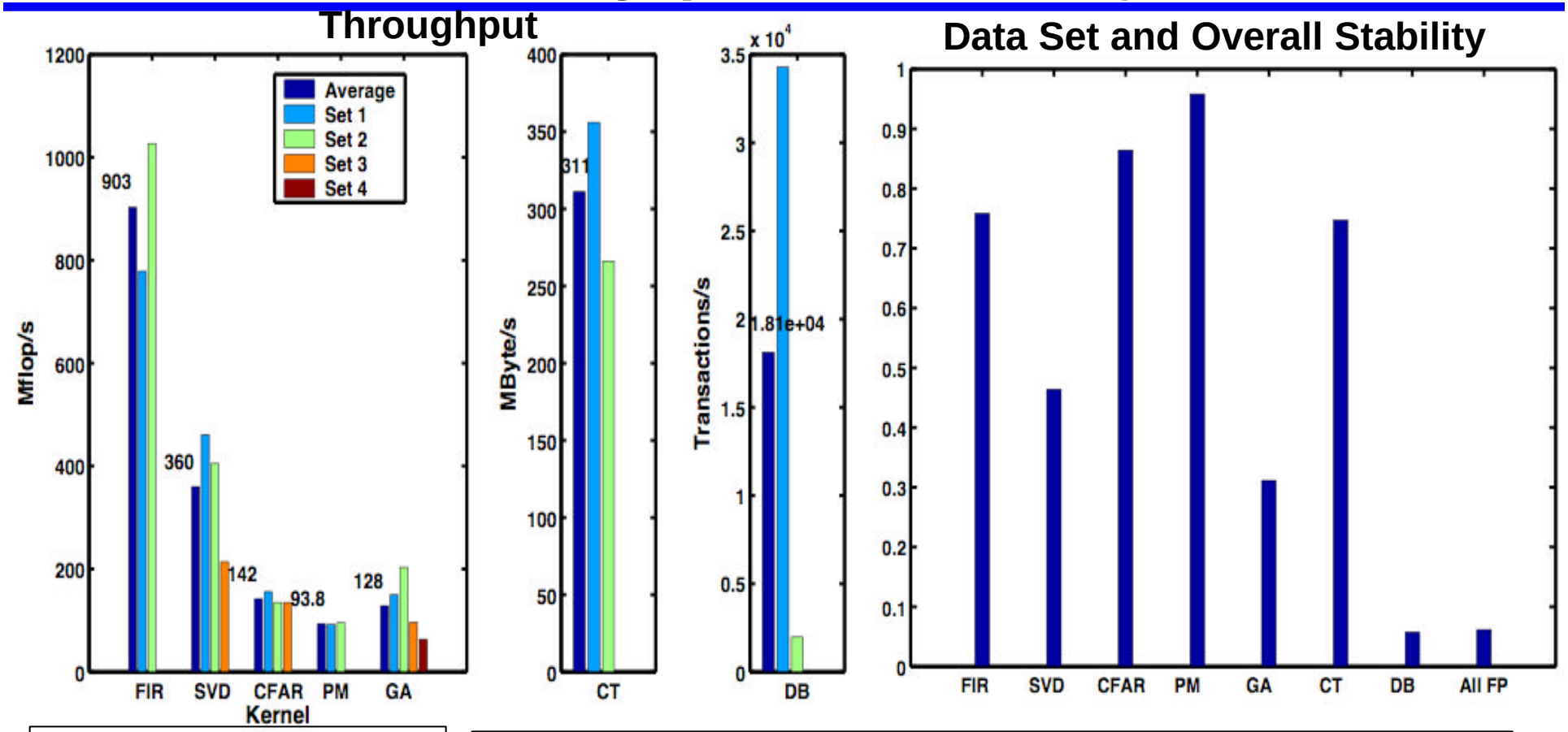
*Implemented with VSIPL Real FIR Filter

PowerPC G4 (Mercury)
- **500 MHz**
- **Peak: 4 GFLOPS/sec**

Mean Efficiency: 29%

Caches are performance bottlenecks
- Performance curve changes when cache is full
- Product metric penalizes G4 for performance drop at cache boundaries

Baseline Performance Measurements: Throughput and Stability

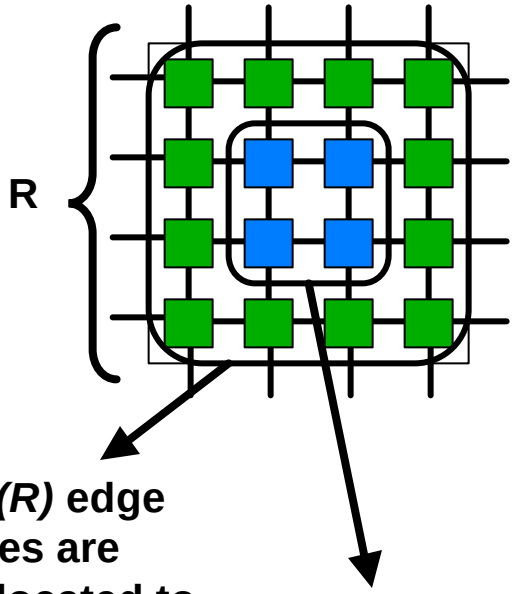


PowerPC G4 (Mercury)
- 500 MHz
- 32 KB L1
- 2 MB L2
- Peak: 4 GFLOPS/sec

Data Set Stability:	**Ratio of minimum to maximum over all data set sizes for a particular kernel**
Overall Stability:	**Ratio of minimum to maximum over all floating-point kernels&all data set sizes**

Stream Algorithms for Tiled Architectures

Systolic Morph



M(R) edge tiles are allocated to memory management

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

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 processors*

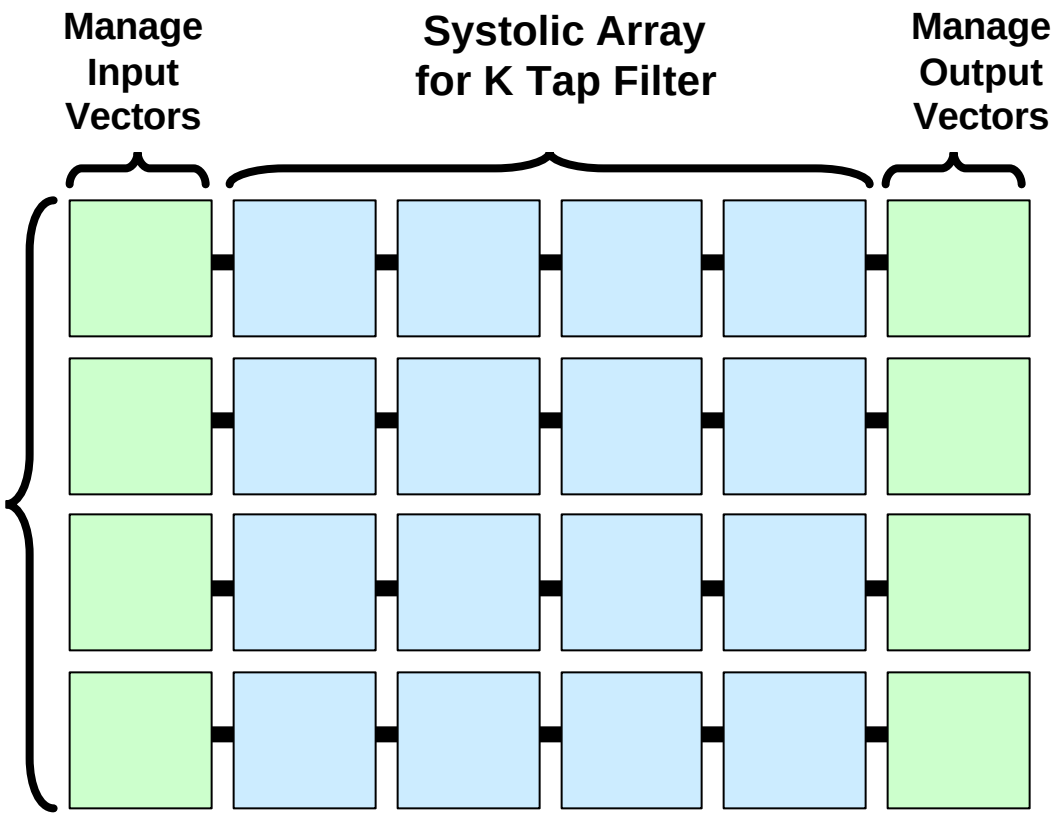
Compute Efficiency Condition: $\lim_{?,R\ ?\ ?} E(?,R) = 1$

where $? = N/R$

Stream algorithms achieve high efficiency by optimizing time space tradeoff – tailoring memory hierarchy and datapaths to specific needs of application

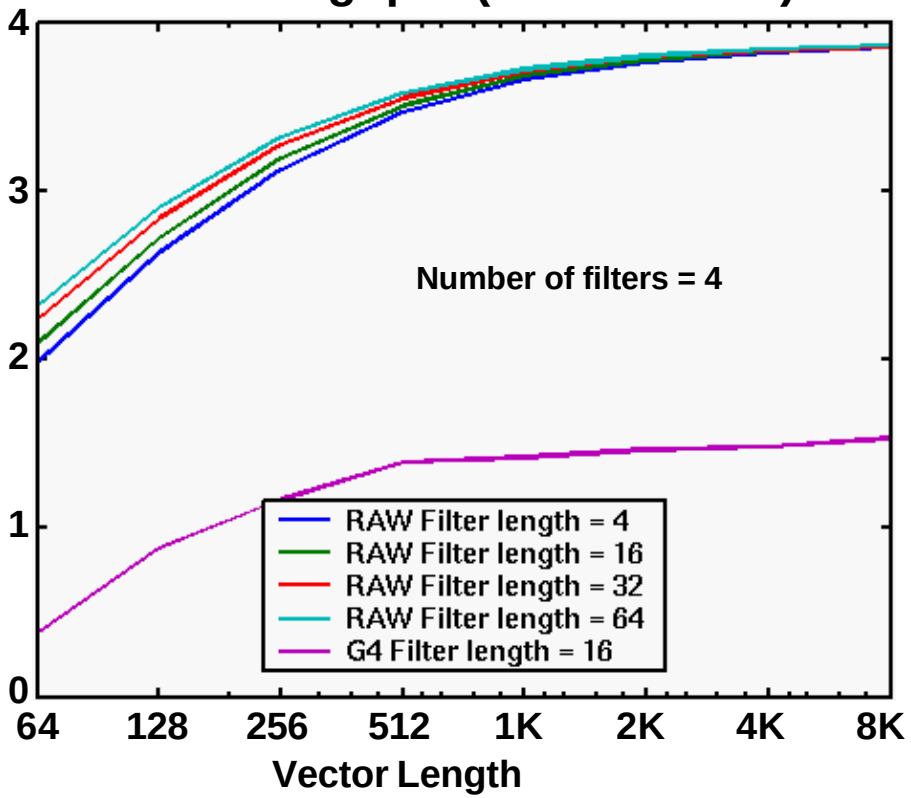
Time Domain Convolution on RAW

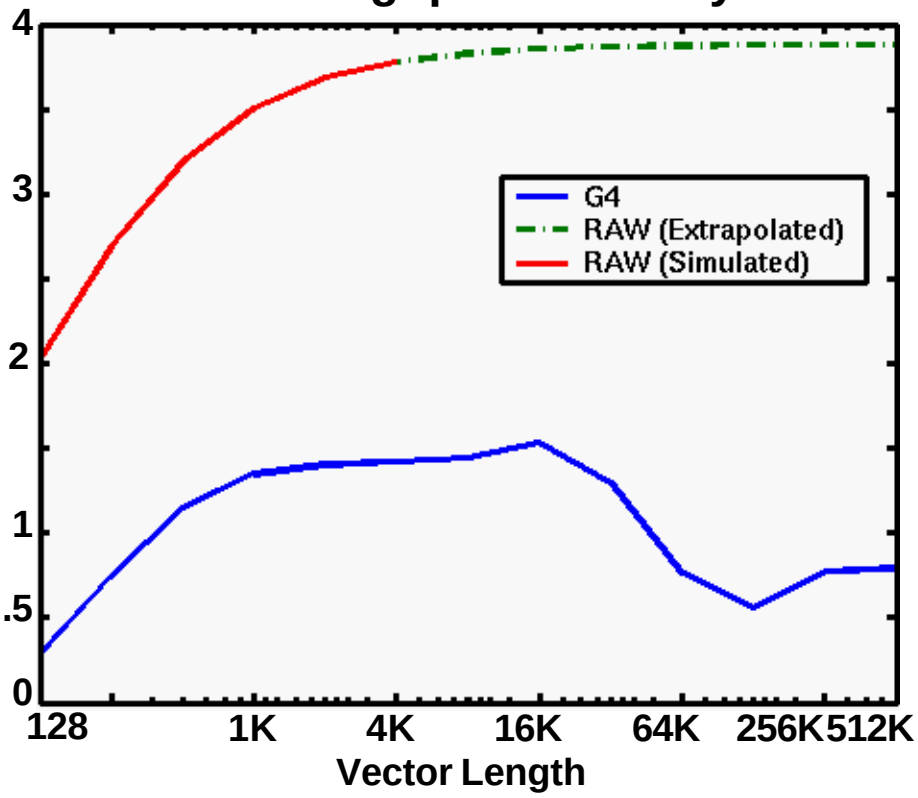
RAW Chip with R rows and R+2 columns:
Number of filters = R
Number of memory tiles:
$M = 2*R$
Number of processing tiles:
$P = R^2$



Stream algorithms achieve high performance by removing memory access bottleneck from computational critical path

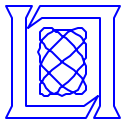
FIR Filter (RAW)





RAW: 250 MHz, 4 GFLOPS/sec **G4: 500 MHz, 4 GFLOPS/sec**

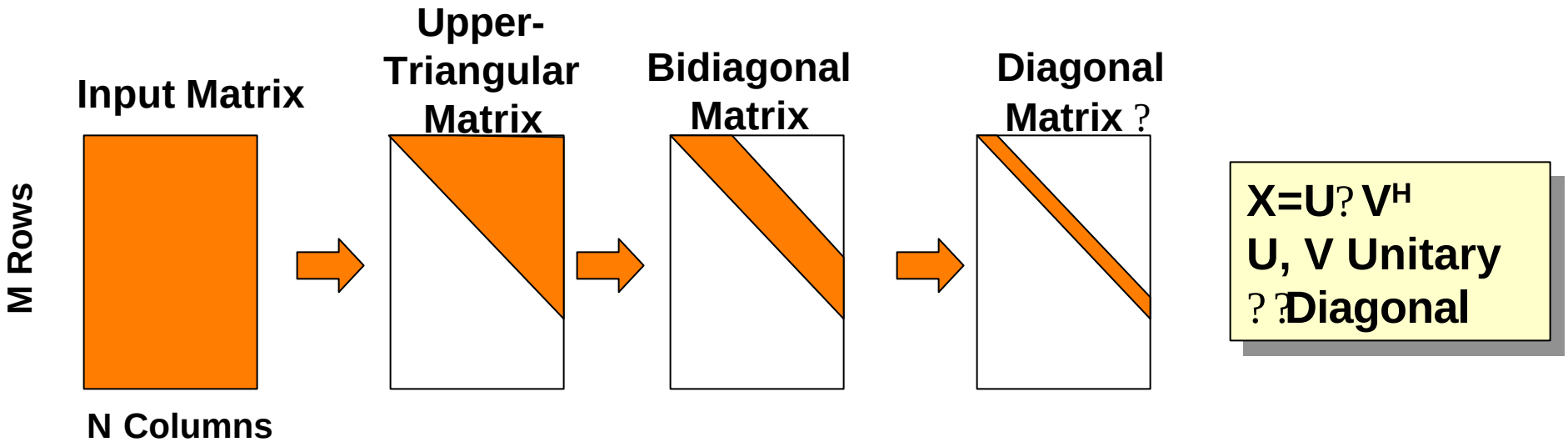
Raw implements the appropriate memory hierarchy for the problem
Raw's Throughput x Stability score stays high



Outline

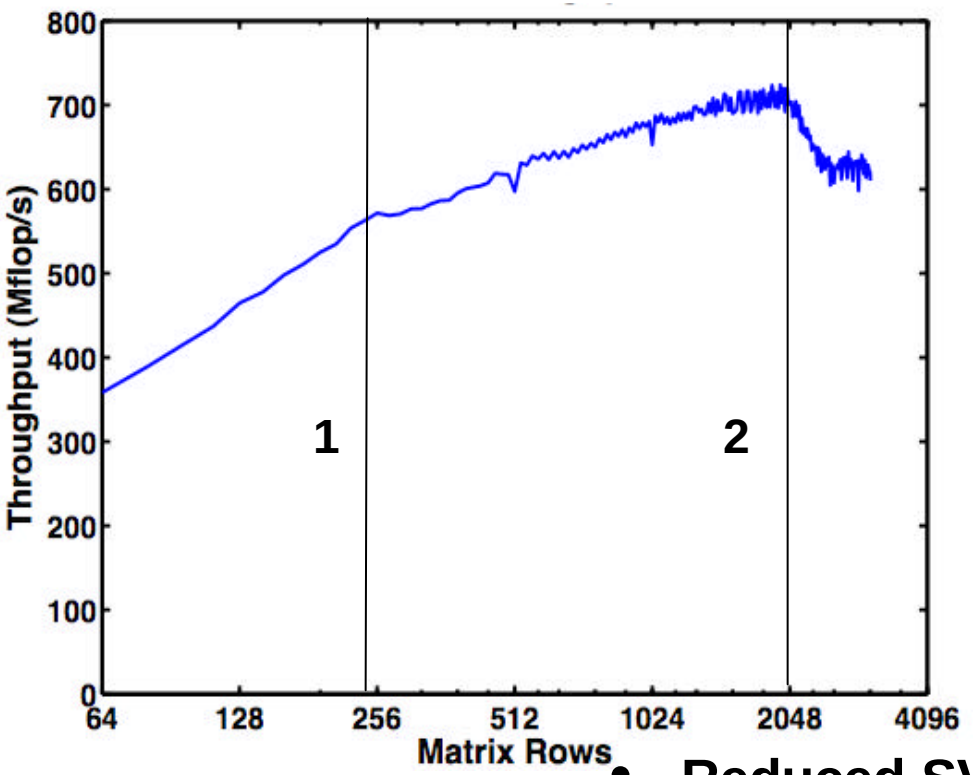
- Introduction
- Kernel Benchmarks and Metrics
- Programming PCA Architectures
- Case Study: SVD Kernel
- Conclusions

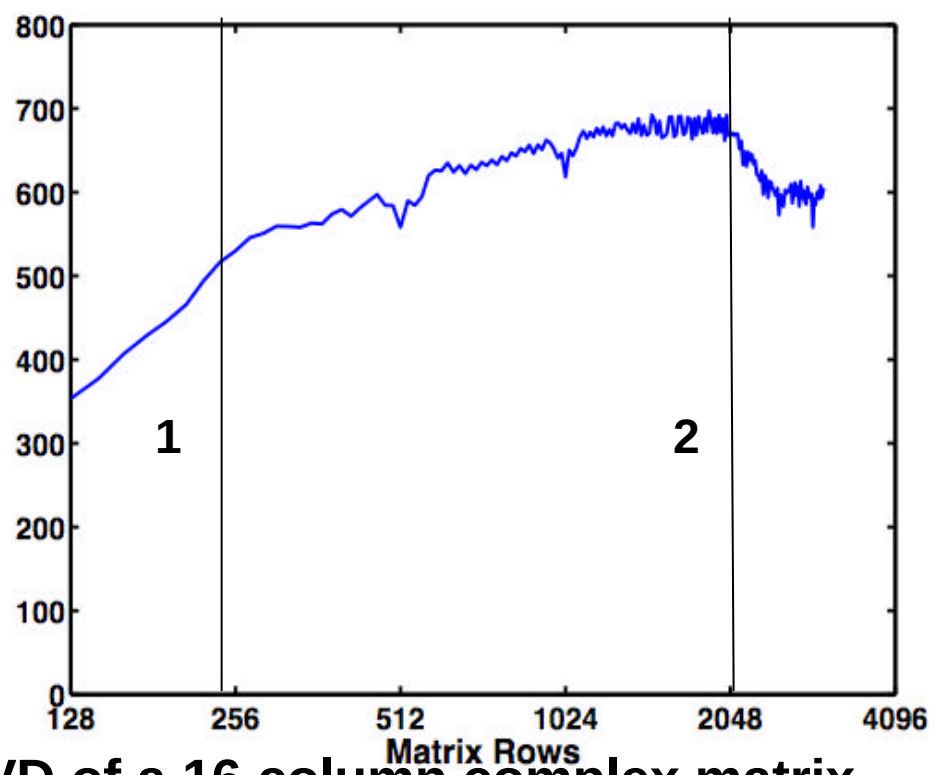
Singular Value Decomposition (SVD)



- SVD is becoming more widely used in signal and image processing
  - Important for spectral analysis
  - Can also be used for adaptive beamforming, especially for ill-conditioned problems
- SVD kernel implementation is a Reduced SVD that begins with a QR factorization if M > N
  - Uses Modified Gram-Schmidt QR factorization
  - Many possible optimizations, especially block factorization

SVD Results (G4)



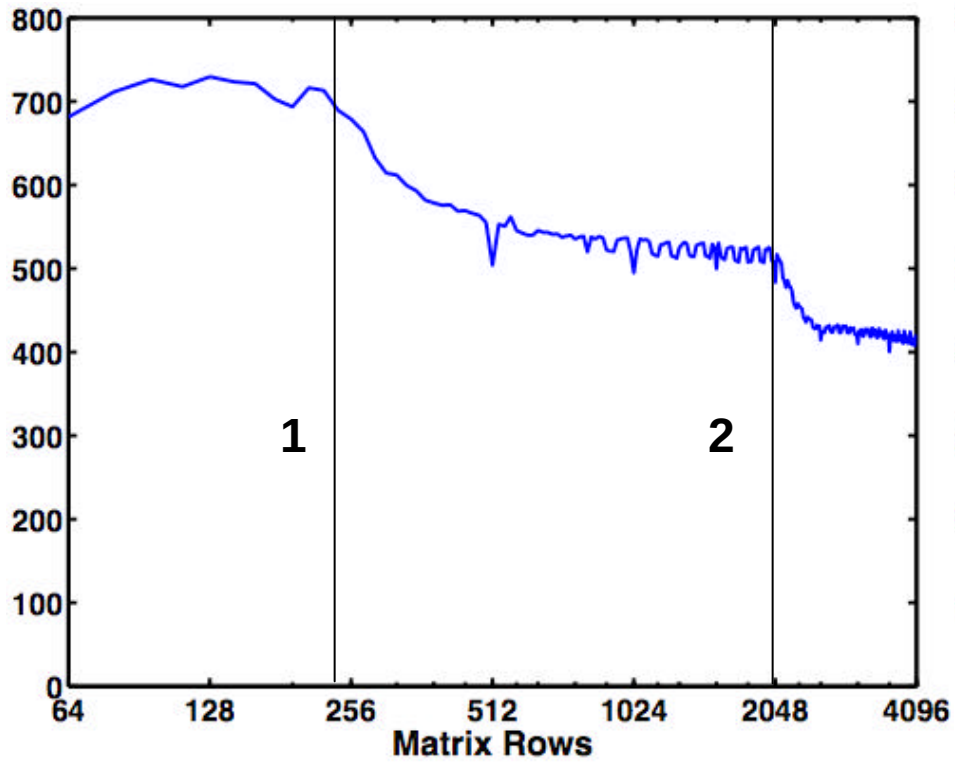


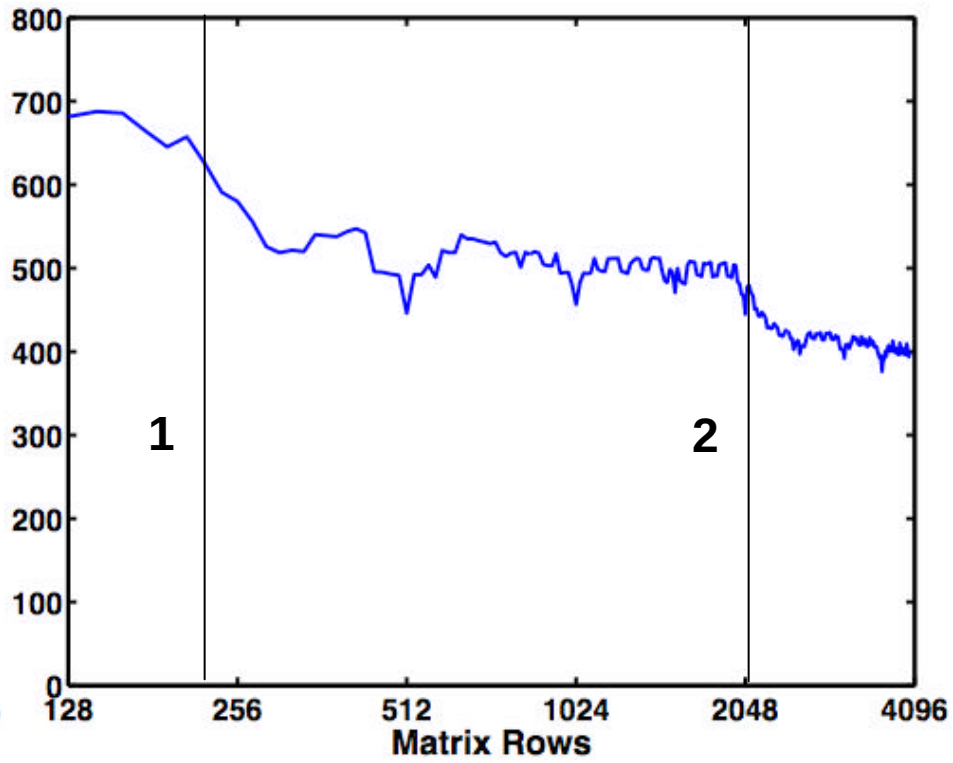
PowerPC G4 (Mercury)
- 500 MHz
- Peak: 4 GFLOPS/sec

Mean Efficiency: 16%

- Reduced SVD of a 16-column complex matrix
- Begins with MGS QR factorization (needs A+R)
- L1 cache drives inner loop performance
  - 1: A+R fills L1 cache
  - 2: One column of A is half of L1 cache

Modified Gram-Schmidt QR Results (G4)



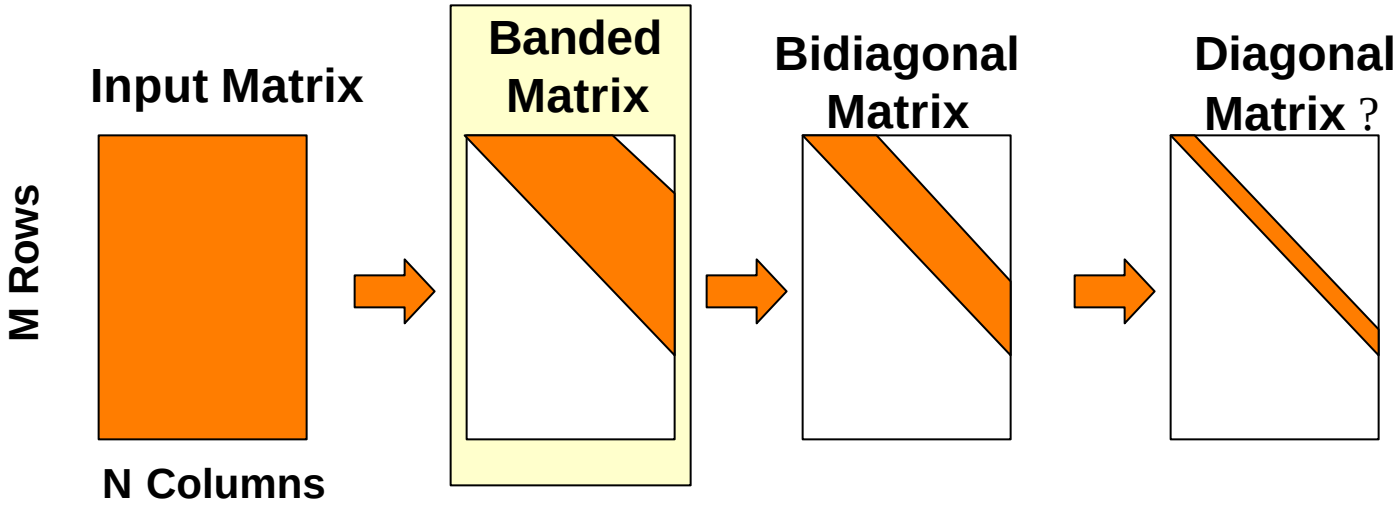


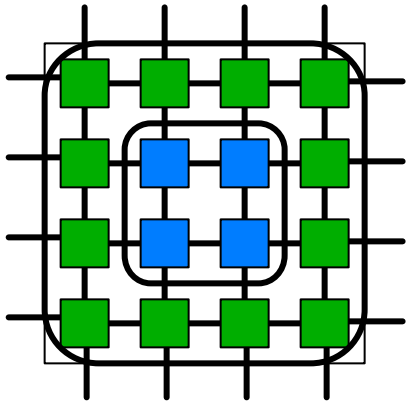
PowerPC G4 (Mercury)
- 500 MHz
- Peak: 4 GFLOPS/sec

Mean Efficiency: 12%

- Modified Gram-Schmidt QR factorization of a 16-column complex matrix
- MGS is about 60% of SVD time
- L1 cache drives inner loop performance
  - 1: A+R fills L1 cache
  - 2: One column of A is half of L1 cache

SVD for RAW Architecture



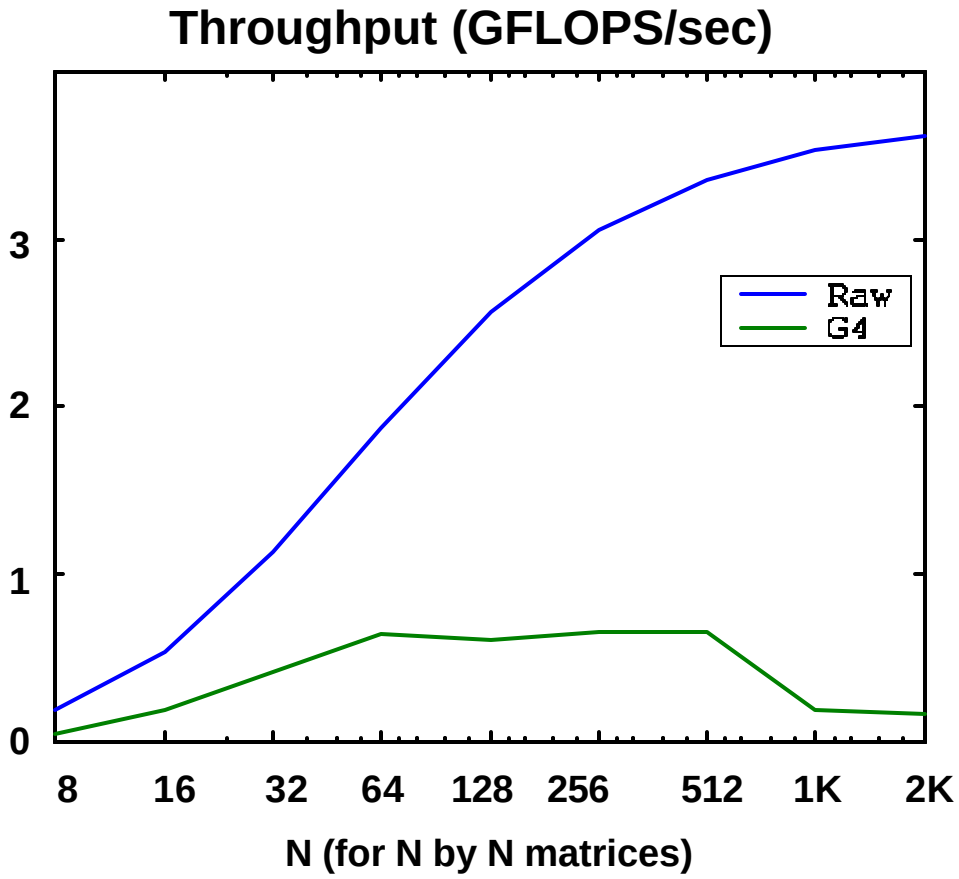


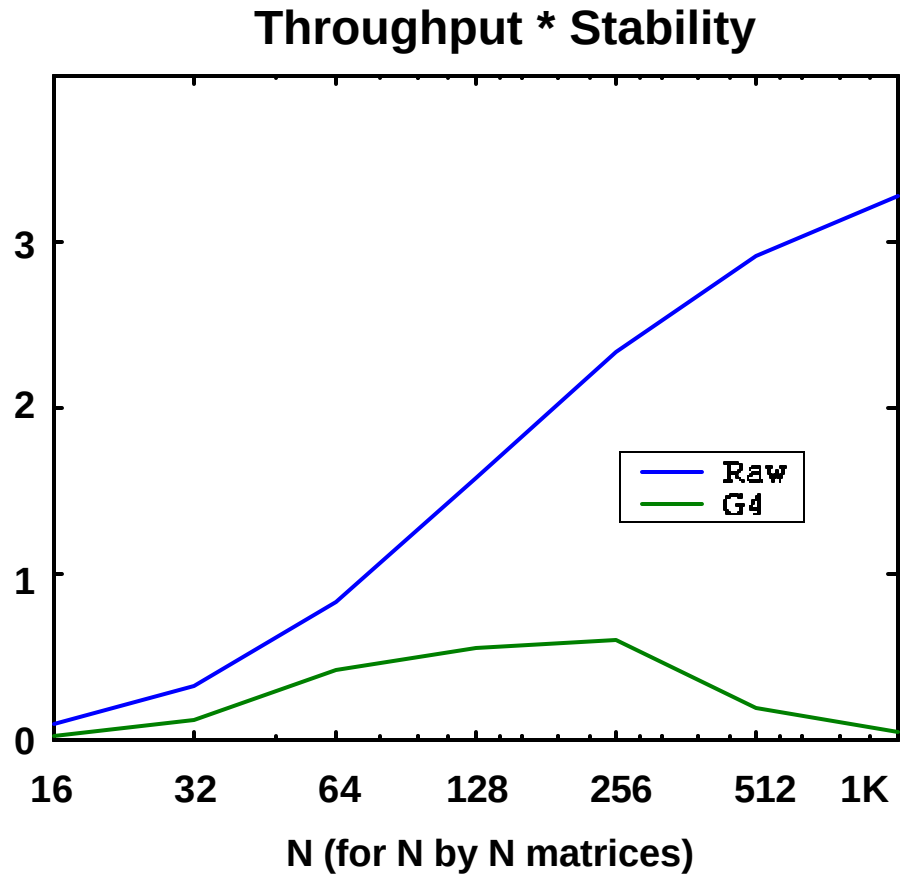
- Goal is to match problem size and architecture
- Use 2D systolic morph
  - maximizes time/space efficiency
  - uses architecture in a scalable way
- Uses efficient QR/LQ approach to get to banded form
  - Fast Givens approach for QR/LQ
  - Decoupled algorithm with good parallelism
- Banded form matches array dimension of systolic morph
  - provides high locality for reduction to bidiagonal form

Raw implementation seeks to efficiently match the many possible algorithms to the many possible architectural configurations

RAW and G4 Results: Fast Givens QR Factorization

The QR is a key sub-kernel of the SVD





The QR performance demonstrates the benefit of the PCA approach on matrix algebra operations

Lincoln Laboratory PCA Testbed

Test Bed Architecture

Clusters on LLAN

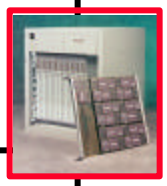
Ethernet LAN

Mercury RACE/VME
Solaris/MCOS

SBC

G4

DSP



Intel PC
- Dual processor
- 66 MHz/64-bit wide PCI bus
- Running Linux

PCI bus

Annapolis Wildstar

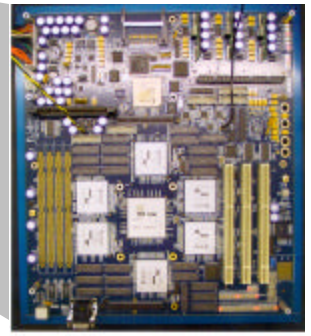
DSP/ FPGA

Unit under test

High Speed I/O

Test Bed Objectives

- Kernel performance evaluation
- Application morphing demonstration
- High-level software prototyping



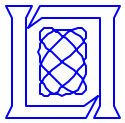
RAW Test Board (October 2003)
- 2 MB DRAM
- High Speed I/O
- USB Interface
- Daughtercard
- High Speed A/D

Outline

- Introduction
- Kernel Benchmarks and Metrics
- Programming PCA Architectures
- Case Study: SVD Kernel
- Conclusions

Conclusions

- MIT Lincoln Laboratory has defined kernel benchmarks for the PCA program
  - Multiple categories of processing
  - Based on DoD application needs
- Establishing a performance baseline on conventional architectures
  - Performance is limited by the blocking factor and by the memory hierarchy
  - Example: CFAR – low ops/byte, 3% efficiency: FIR – high ops/byte, 29% efficiency
- PCA processors allow opportunities for high performance
  - Performance achieved through co-optimization of the algorithm and the architecture
  - Example: unusual SVD algorithm leads to high performance on Raw
  - The greater degree of freedom allows greater optimization across a variety of problem domains



MIT Lincoln Laboratory PCA Team

Hector Chan
Bill Coate
Jim Daly
Ryan Haney
Hank Hoffmann
Preston Jackson
James Lebak
Janice McMahon
Eddie Rutledge
Glenn Schrader
Edmund Wong