Computational Architecture Analysis of KASSPER Algorithms

Matthew French, Karandeep Singh

mfrench@isi.edu, karan@isi.edu

University of Southern California

Information Sciences Institute

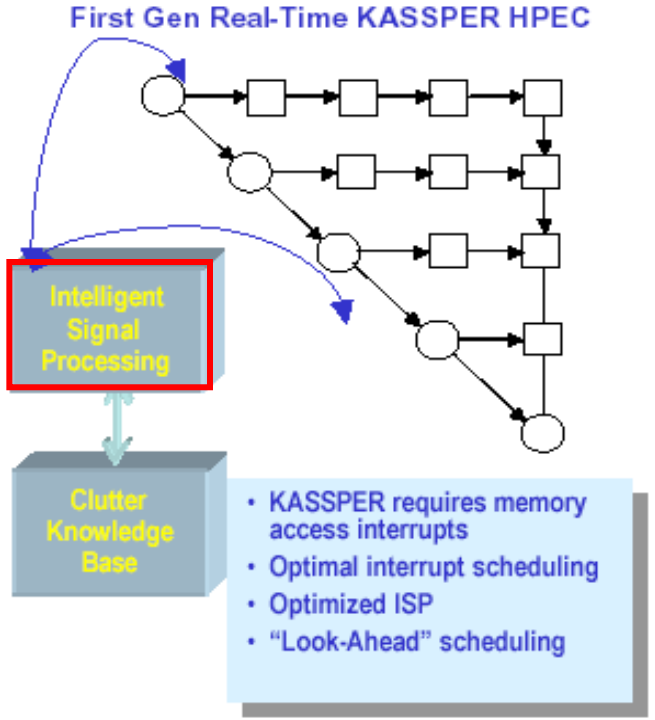
February 16th, 2005

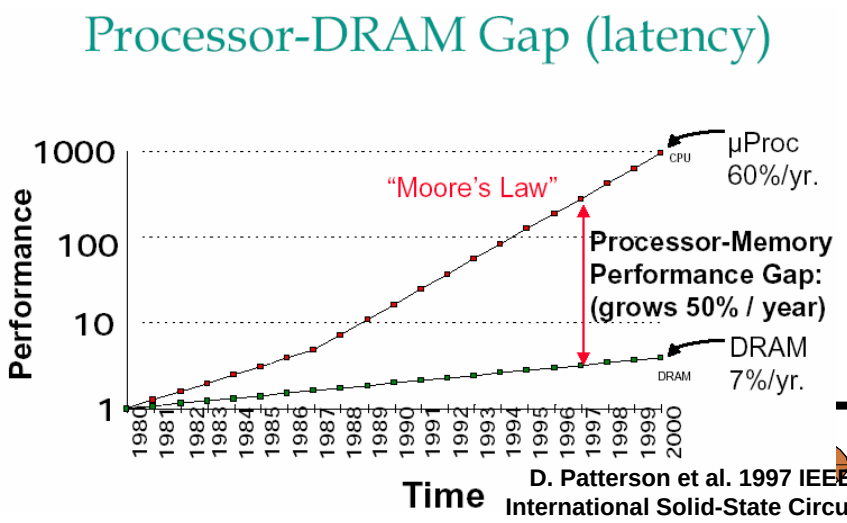
Outline

- Computer Architecture Motivation
- Brief Overview of Prior Research
- ISL Algorithm Update
- Performance Results of Scalability Study
- Future Work
- Summary

Motivation

- **KASSPER algorithms blend Knowledge processing with traditional signal processing (Threaded and Streaming)**
- **Traditional Signal Processing**
  - **Streaming, FLOPs**
- **Knowledge Processing**
  - **Database Access Rates, Large Memory Structures**
- **Expert Reasoning**
  - **Decision-in-the-loop**
  - **Adaptive (data-dependant) algorithms**
- **New Set of Computer Architecture Parameters Stressed**
  - **Memory Latency**
  - **I/O Throughput**
  - **Multi-threaded Application Performance**
  - **Data Locality**
- **Computing architectures will drastically change system level requirements**
  - **Size, Weight, Energy, Power, Time (SWEPT)**
  - **Interconnect**
  - **Type / Size of Memory Banks**
- **Realizable today, 18 months, 10 years?**





Candidate Processing Units

PowerPC

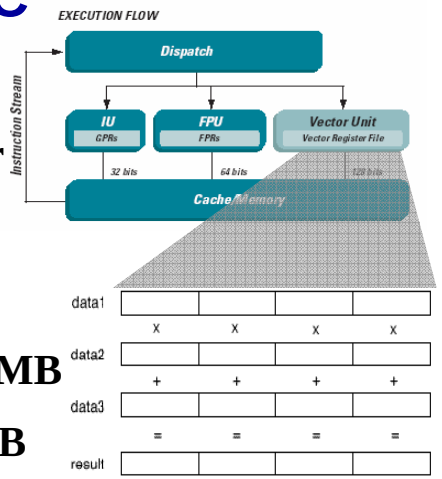
RISC Architecture with Altivec Co-Processor

500 MHz

4 GFLOPS Peak (MAC)

SDRAM: 125 MHz, 256 MB

L2 cache: 250 MHz, 2 MB



V-IRAM

Processor In Memory (PIM) Architecture

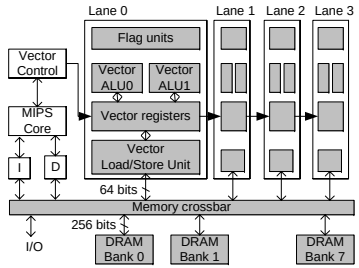
200 MHz; 13MB DRAM

6.4 16-bit GOPS, 1.6 GFLOPS

4 float ALUs; 8 32bit int ALUs; 16 16bit ALUs

12.8 GB/s peak memory access

http://iram.cs.berkeley.edu/



Imagine

Streaming Architecture

300 MHz, VLIW SIMD machine

28 16-bit GOPS, 14 GFLOPS

128 kB Streaming Register File

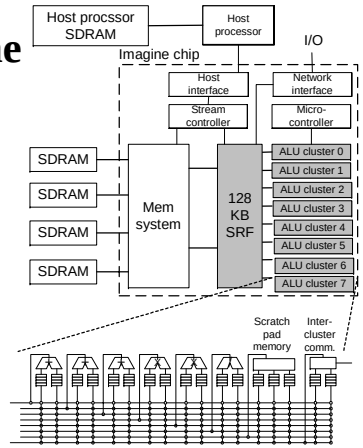
8 ALU Clusters

6 ALUs / cluster

84-95% ALU utilization typical

Streaming Memory Buffers

http://cva.stanford.edu/



Raw

Tiled Architecture

16 tiles of MIPS R4000 @ 425 MHz (chip)

6.8 GOPS or GFLOPS

4 Communication Networks

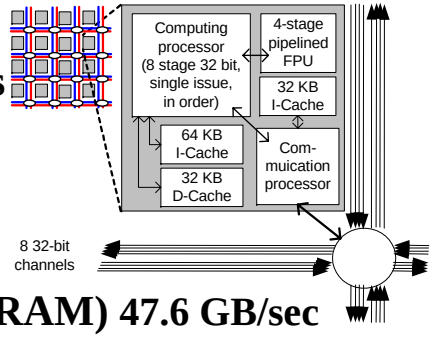
2 Static Networks,

1 cycle throughput

3 cycle latency

54.4 GB / sec

2 Dynamic Networks



14 External Ports (I/O or DRAM) 47.6 GB/sec

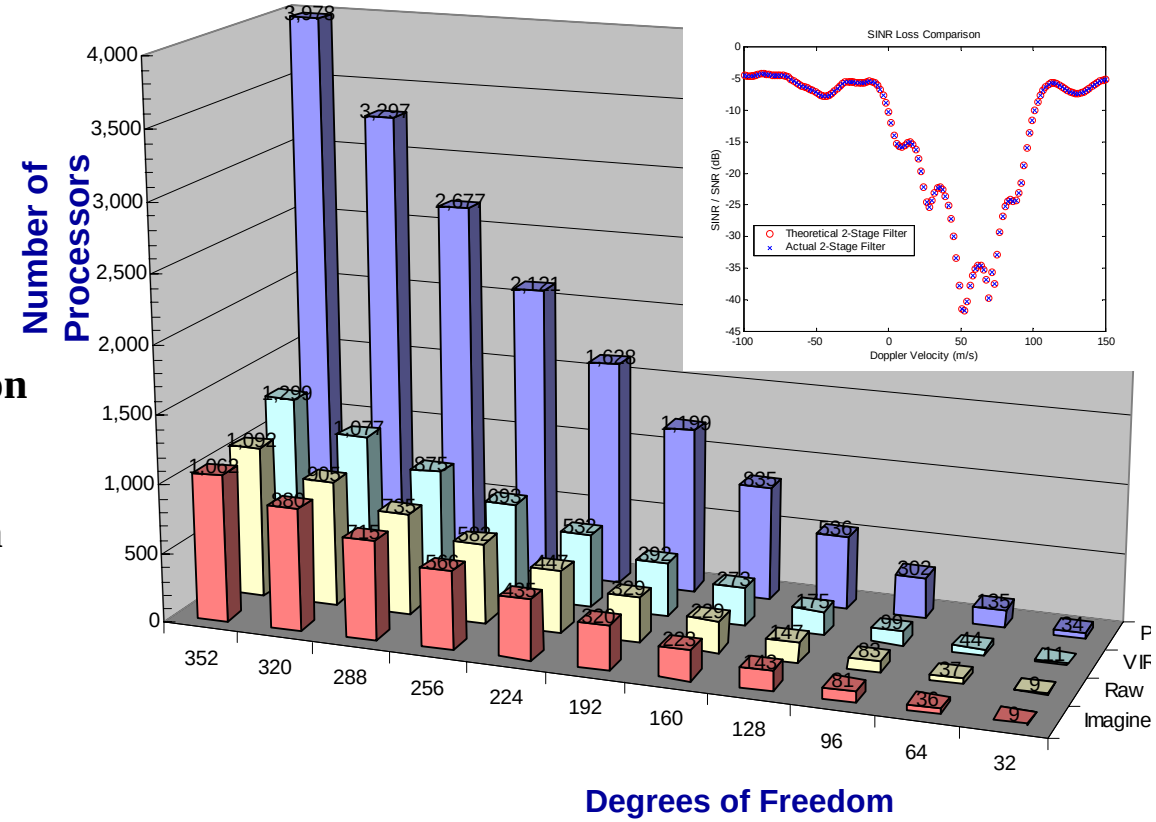
http://www.cag.lcs.mit.edu/raw/

KASSPER
Knowledge-Aided Sensor Signal Processing and Expert Reasoning

DARPA

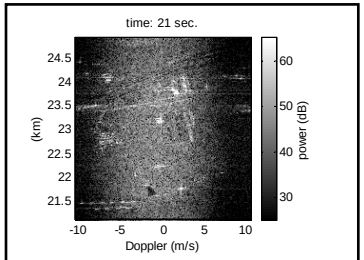
Pre-Filter Prediction System Sizing

- **Bottom-up Approach**
  - Optimal kernel sizing for each processor- No cache misses
  - Scale # of processors to meet system FLOP counts
- **1st Order Analysis**
  - System scalability, parallelization penalties, OS – not taken into account
  - MAC Instruction Not Present in Raw
    - ~2x improvement for all kernels
- **Research Chips**
  - Small Team of PhD Students vs Industry Professionals
  - Standard Cells vs Custom ASIC Logic
- **HPEC Architectures Yielding ~ 4x Improvement over PowerPC**

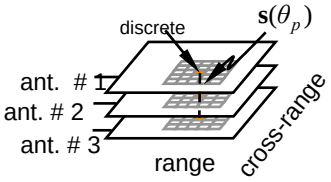


PPC 500 MHz
V-IRAM 200 MHz
Raw 400 MHz
Imagine 300 MHz

Pre-Filter Algorithm – Data Domain (ISL Pre-whitening Algorithm)



Threshold "low resolution" SAR map to detect discrete clutter



$$\mathbf{v}_m(\theta_p, f_p) = (\mathbf{H}_m^H \mathbf{t}(f_p)) \otimes \mathbf{s}(\theta_p)$$

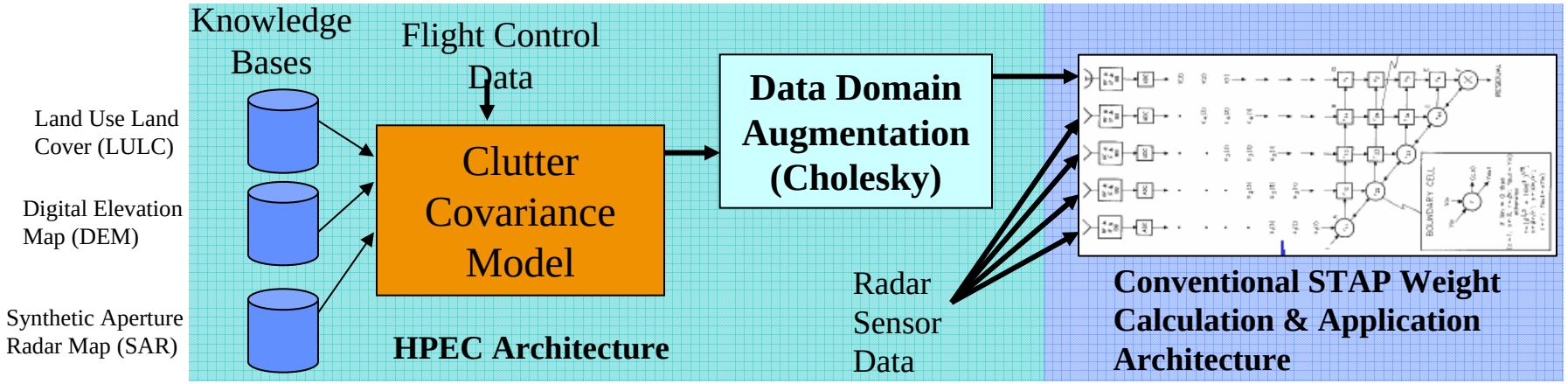
$$\mathbf{t}_{[m]}(f_p) = \exp(j2\pi m f_p T_{pri})$$

Form space-time response for each discrete and transform to post-Doppler space

$$\mathbf{Q}_{m,l} = \sum_{p=1}^{P_c} \mathbf{v}_{m,l}(\theta_p, f_p)\mathbf{v}_{m,l}^H(\theta_p, f_p)$$

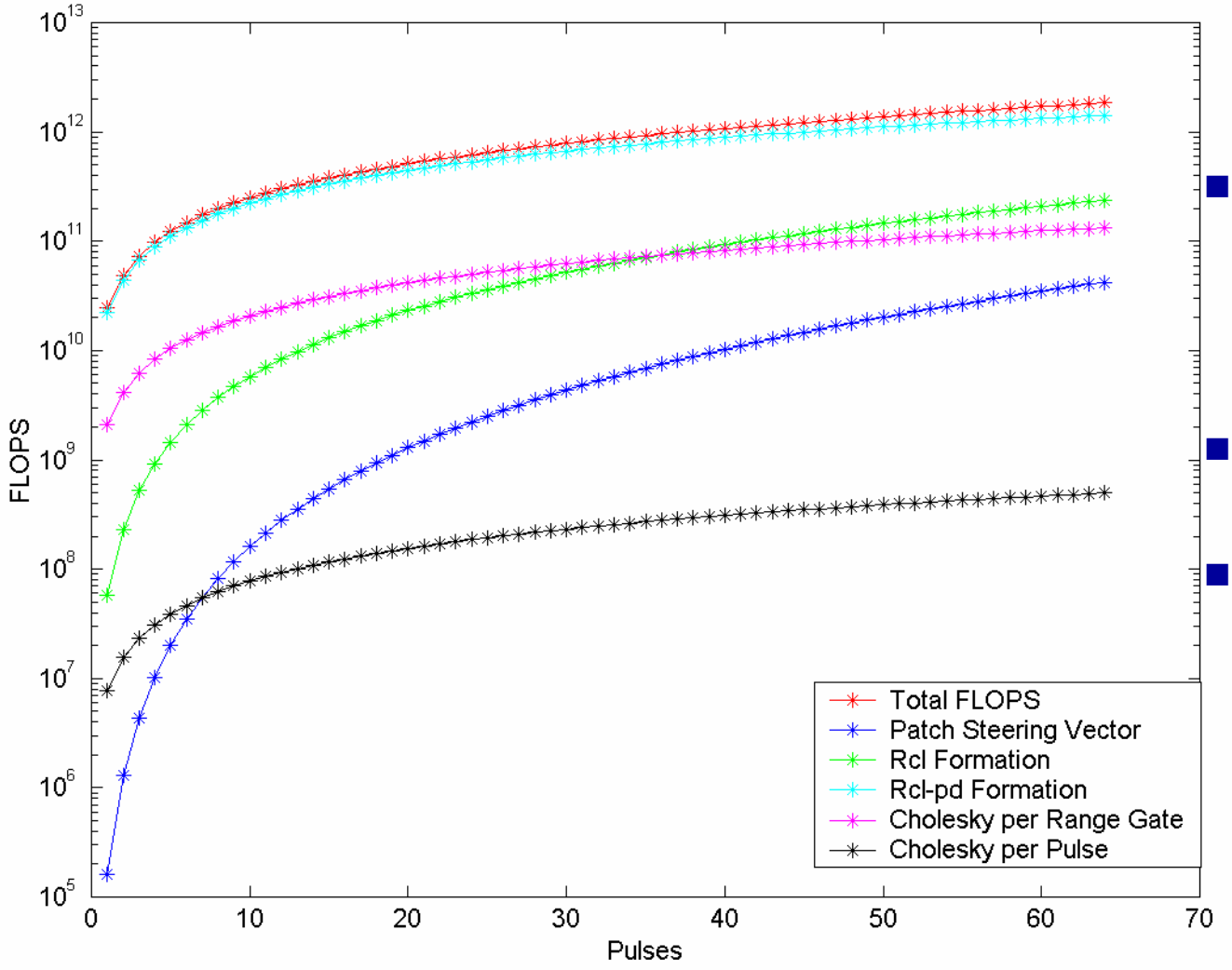
$$\mathbf{R}_{m,l} = \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_{m,k}\mathbf{x}_{m,k}^H + \beta\mathbf{Q}_{m,l} + \mathbf{I}$$

Use responses to form a range-dependent "loading" matrix for each Doppler bin, add to sample cov. and run STAP processor



- **Clutter covariance model calculations (covariance domain colored loading matrix) form** $\mathbf{Q} = \beta_d \mathbf{R}_c + \beta_L \mathbf{I}$
- **Cholesky factor of loading matrix (data domain augmentation matrix) forms** $\mathbf{C} = \mathbf{Q}^{1/2}$
- **Conventional STAP Processing Performs Color Loaded QR Decomposition**

Pre-whitening Computational Analysis



- R_{cl_pd} formation (V^H*Rcl*V) is ~65% of FLOPS
  - Currently brute force
- Performing Cholesky per Doppler bin vs per range gate removes FLOPS but is only ~23% of total
  - $4.9x10^{11}$ -> $1.8x10^{9}$
- Memory sizes within feasibility
- Additional patches have minimal impact
  - Linear vs $O(N^3)$
  - 10-100 patches typical

Using KASSPER Data Set 3 Parameters

Channels = 6; Pulses = 64; Range Gates = 268; Patches = 161; PRF = 1kHz; DoF_{rd}=18;

Pre-whitening Computational Analysis

Operation	Key Kernels	Order of Op	FLOPS (Full Sizing)
Patch Steering Vector	Matrix mult	$O(Patches*Pulses^3)$	$3.2x10^{10}$
Rcl Formation	Vector mult and add	$O(10*Patches*(Ch*Pulses)^2)$	$2.4x10^{11}$
R_{cl_pd} Formation	Matrix mult	$O(8*(Ch*Pulses)^2*DoF_{rd}+(Ch*Pulses*DoF_{rd}^2))$	$1.38x10^{12}$
Cholesky Decomposition	Cholesky Decomposition	$O(8*(DoF_{rd}^3 - DoF_{rd})/6)$	$4.9x10^{11}$
Total			**$2.14x10^{12}$**

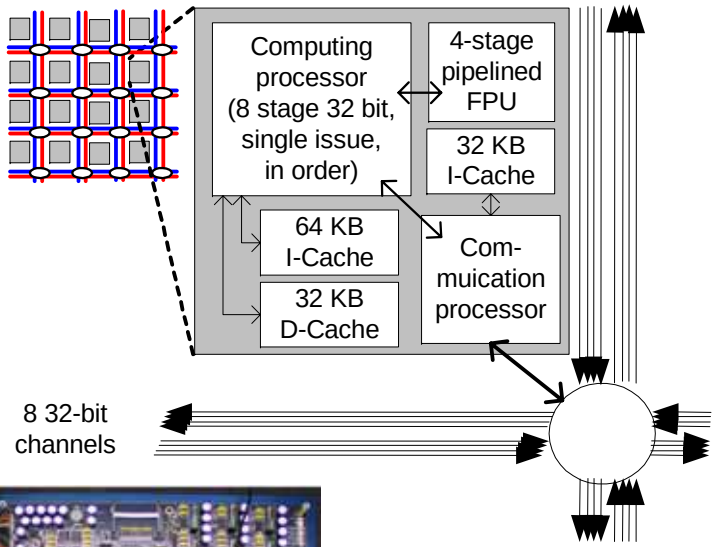
Using KASSPER Data Set 3 Parameters

Channels = 6; Pulses = 64; Range Gates = 268; Patches = 161; PRF = 1kHz; DoF_{rd}=18;

- **Key Kernels**
  - **Matrix*Matrix**
  - **Cholesky**
  - **Matrix addition and subtraction**
  - **Vector*Vector**

Raw (MIT)

- 16 tiles of MIPS R4000 @ 425 MHz
  - Boards limited to 400 MHz
  - 6.4 GOPS or GFLOPS
- 4 Communication Networks
  - 2 Static Networks,
    - 1 cycle throughput
    - 3 cycle latency
    - 54.4 GB / sec
  - 2 Dynamic Networks
- 14 External Ports (I/O or DRAM)
  - 47.6 GB/sec
- C and ASM; gcc based compiler
- 18.2 x 18.2 mm; IBM Foundry
- Fully scalable architecture
- Demonstration on Hardware at DARPATech 2004
- http://www.cag.lcs.mit.edu/raw/

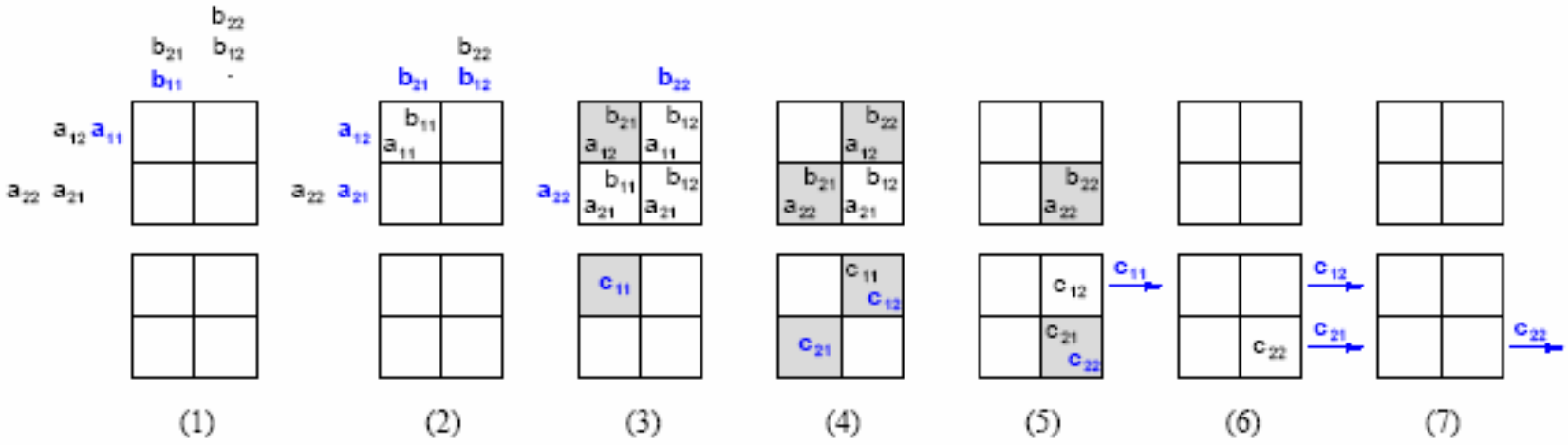


Raw Motherboard

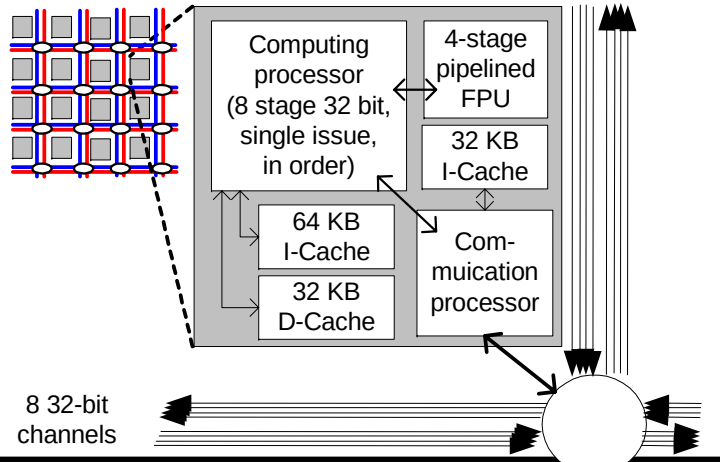
Raw Fabric Quad Board

DARPATech 2004 Demo

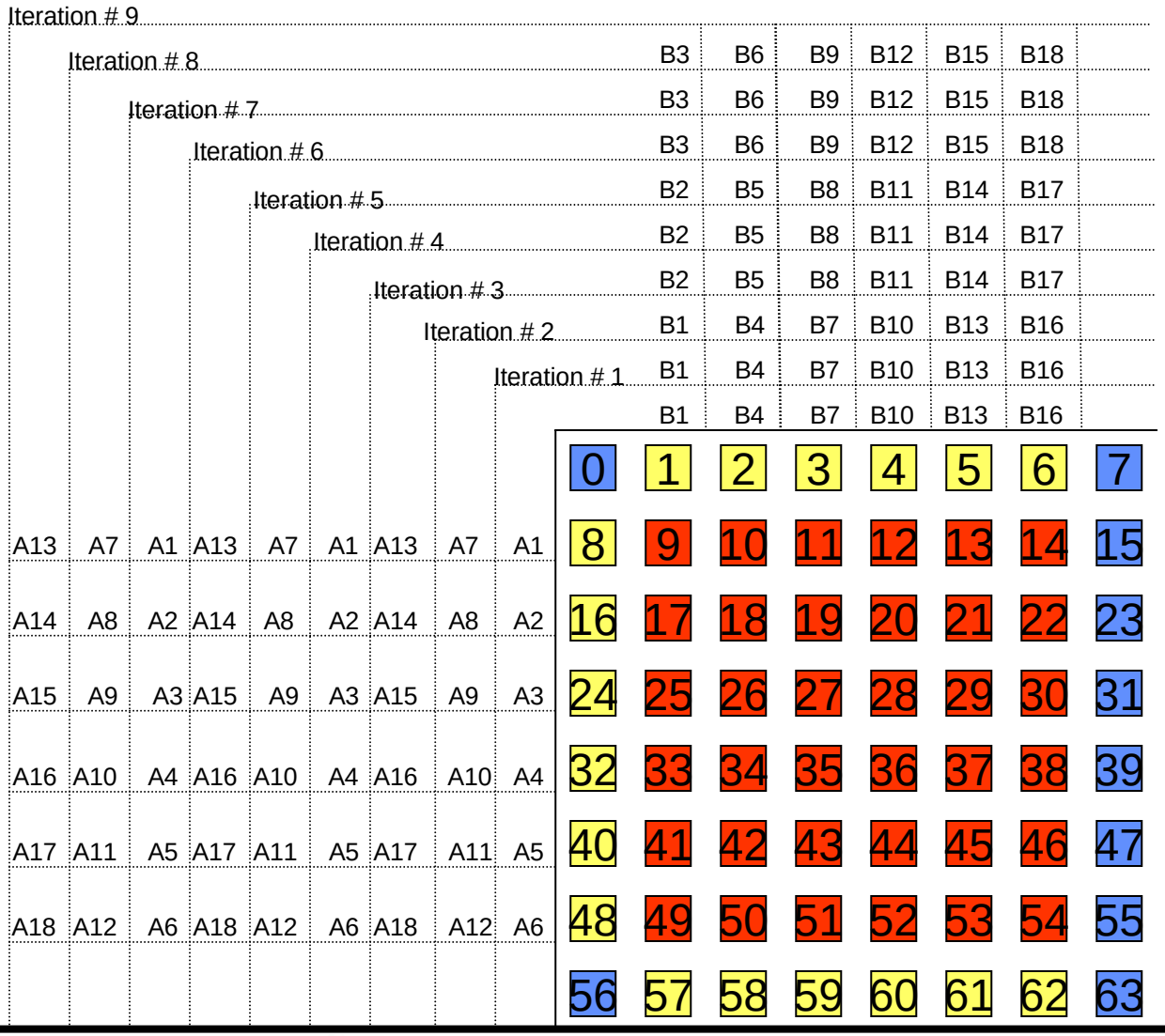
Raw Matrix*Matrix Kernel



- Stream A vectors from West; Stream B vectors from North;
- C streams out South;
- 16 Tiles * 400 MHz = 6.4 GFLOPS Peak



Raw Matrix Multiply



Memory Tile

Compute Tile

Unused Tile

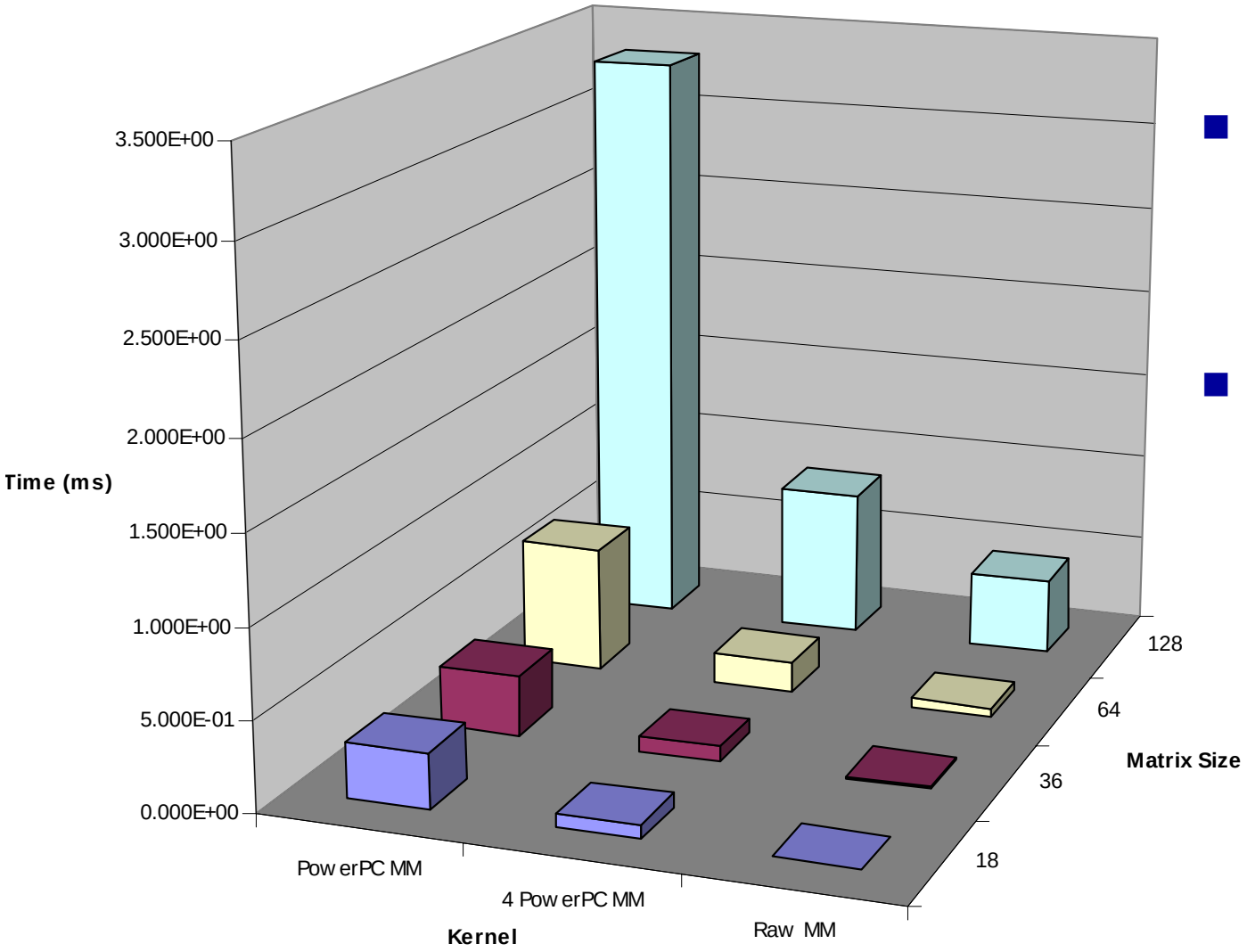
- A_n = row N of A
- B_n= col N of B
- Provides highest memory bandwidth and data locality
- Tiles 57 – 62 get the columns of the result matrix
- Right hand tiles not used

Raw Matrix Multiply Results

Problem Size (Effective Size) N	No. of cycles	FLOPS per Tile (Theoretical) (N*N*N*2)/(R*R)	FLOPS/ Cycle/ Compute Tile	Total FLOPS/ Cycle/
18	980	324	0.3299	11.8764
36	4282	2592	0.6053	21.7908
64	20808	14563.55	0.7675	27.63
128	181516	116508.44	0.7039	25.3404

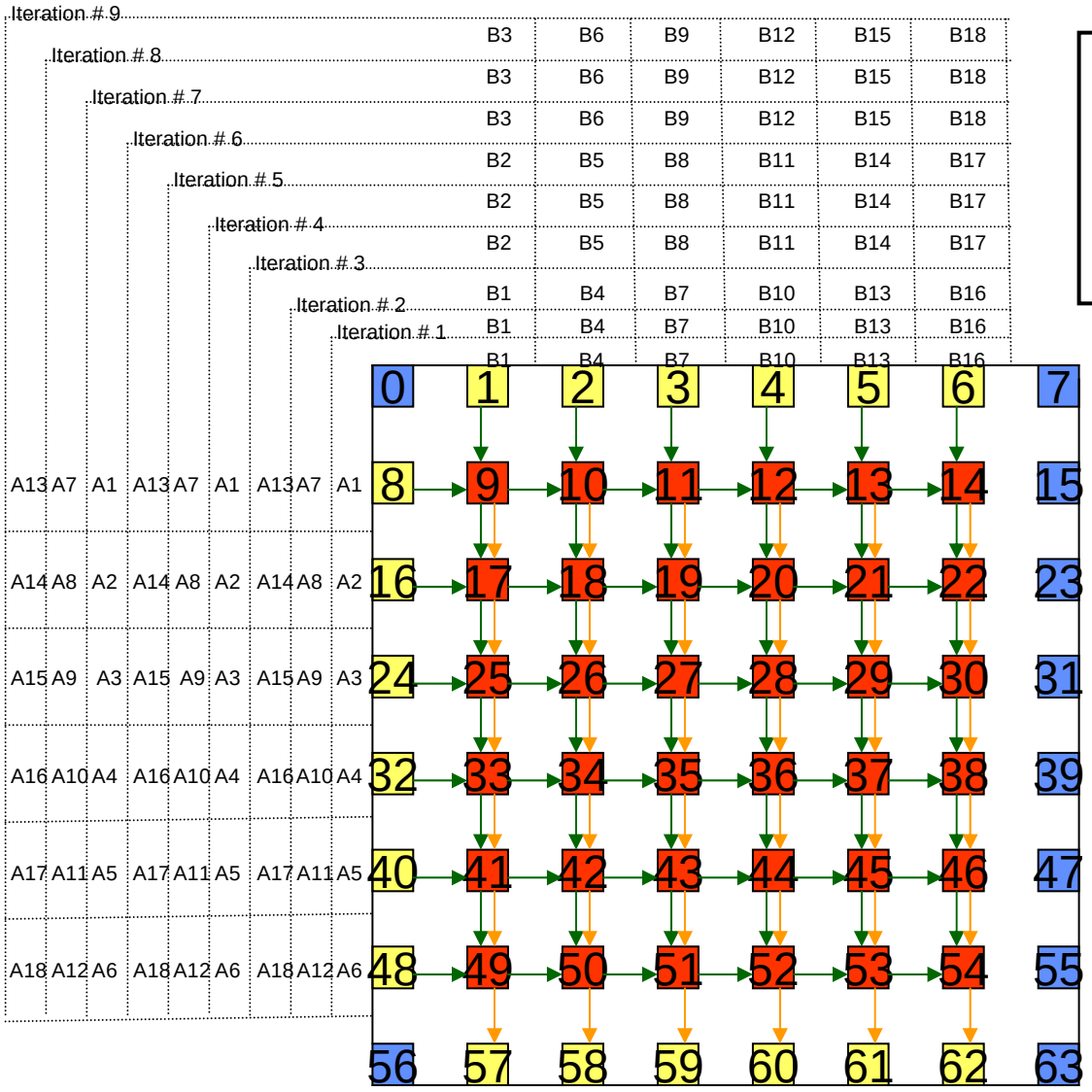
- R = Length of compute tile region =6
- Total FLOPS/Cycle includes memory tile and unused tile overhead
  - Maximum = 64 FLOPS/cycle
  - Achieving 27.6/64 = 42% efficiency
- Phase I study (1 Chip = 16 tiles max, 4 compute)
  - 4/16 = 25% efficiency
- Efficiency Increasing with Tile Scaling

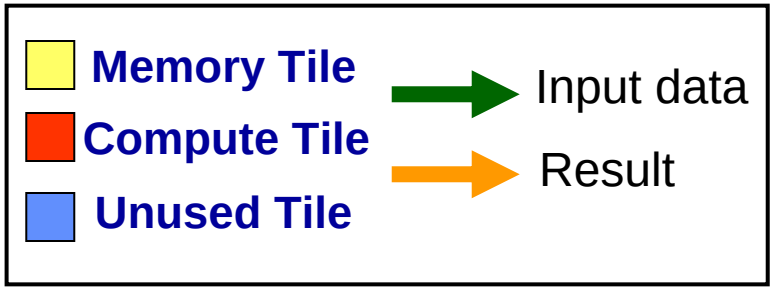
Matrix Multiply: Raw vs PowerPC



- Assuming linear scaling for PowerPC (unrealistic)
- Raw achieving 2-4x over equivalent number of PowerPCs

Vector Cross Product



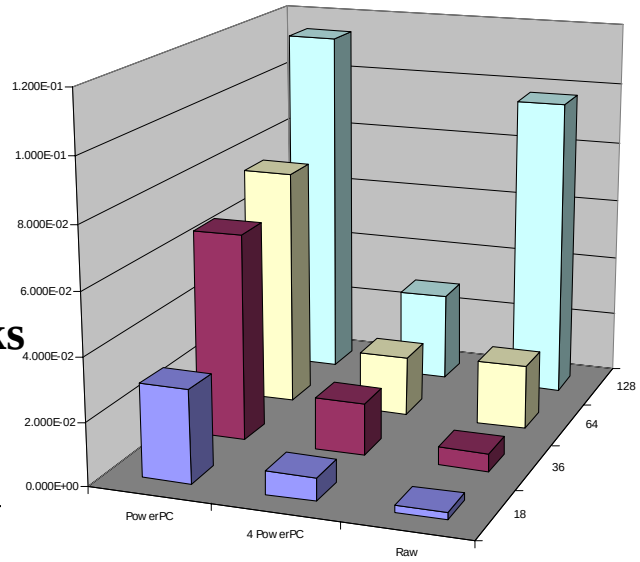


- Similar to Matrix Multiply but..
  - No partial product
  - Each compute tile produces an output on each iteration
- Therefore
  - Each compute tile must pass the current data value, its output, and the output of all tiles North
  - Mesh network limits peak efficiency

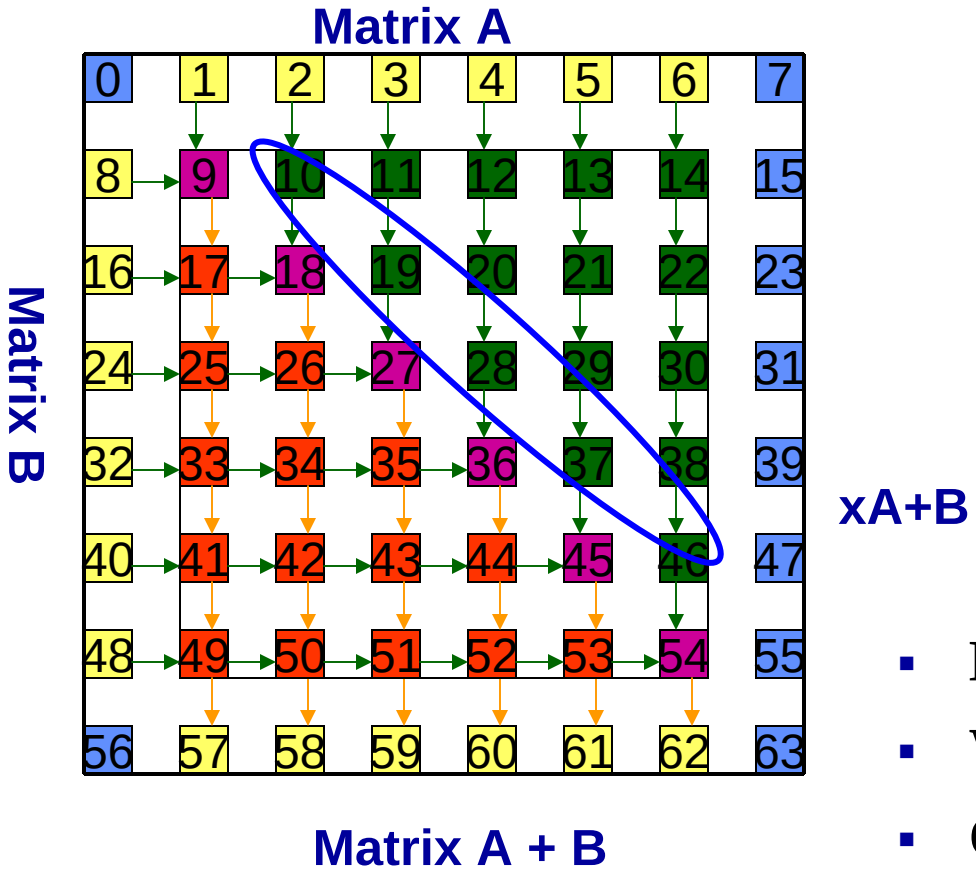
Raw Vector Multiply Results

Problem Size (Effective Size) N	No. of cycles	FLOPS per Tile (Theoretical) (N*N)/(R*R)	FLOPS/ Cycle/ Compute Tile	Total FLOPS/ Cycle
36	2285	36	0.0157	0.5652
64	8322	113.78	0.0173	0.6228
128	38989	455.11	0.0148	0.5328

- R = length of compute tile region =6
- Kernel is communication bound
- Could optimize further
  - Interleave outputs to East and South memory banks
  - Utilize 2nd static network more efficiently
- Not the most significant kernel in application
  - Matrix Multiplication 65% of FLOPS



Matrix Addition



Input data

Result data

Memory processor

Compute processor

Pass input data from north to south

Pass input data from west to east and results from north to south

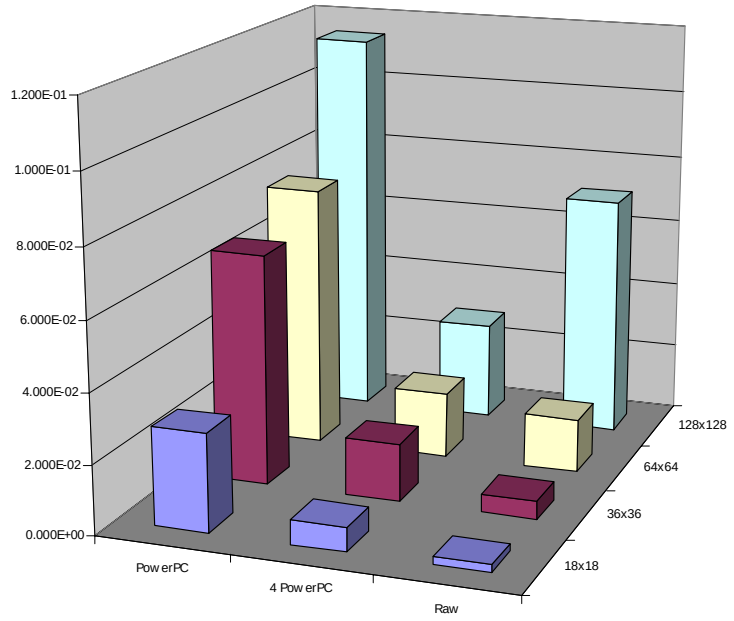
Unused tiles

- **North tiles get rows of A**
- **West tiles get rows of B**
- **Only diagonal tiles do computation**
- **Can add other computations in off diagionals**

Raw Matrix Addition Results

Problem Size (Effective Size) N	No. of cycles	FLOPS per Tile (Theoretical) (N*N)/(R)	FLOPS/ Cycle/ Compute Tile	Total FLOPS/ Cycle
18	801	54	0.0672	0.4032
36	2093	216	0.1031	0.6186
64	6266	682.67	0.1158	0.6948
128	28265	2730.67	0.1027	0.6162

- R = length of compute tile region =6
- Small portion of overall application



Cholesky Blocking Algorithm (PLAPACK)

$$A = \left(\begin{array}{c|c} a_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c} \lambda_{11} & 0 \\ \hline l_{21} & L_{22} \end{array}\right)\left(\begin{array}{c|c} \lambda_{11} & l_{21}^T \\ \hline 0 & L_{22}^T \end{array}\right) = \left(\begin{array}{c|c} \lambda_{11}^2 & \star \\ \hline \lambda_{11} l_{21} & l_{21} l_{21}^T + L_{22} L_{22}^T \end{array}\right)$$

From this we derive the equations

$$\begin{aligned} \lambda_{11} &= \sqrt{\alpha_{11}} \\ l_{21} &= a_{21} / \lambda_{11} \\ A_{22} - l_{21} l_{21}^T &= L_{22} L_{22}^T \end{aligned}$$

1. Partition $A = \left(\begin{array}{c|c} \alpha_{11} & \star \\ \hline a_{21} & A_{22} \end{array}\right)$
2. $\alpha_{11} \leftarrow \lambda_{11} = \sqrt{\alpha_{11}}$
3. $a_{21} \leftarrow l_{21} = a_{21} / \lambda_{11}$
4. $A_{22} \leftarrow A_{22} - l_{21} l_{21}^T$
5. continue recursively with A_{22}

Data dependency

vector*vector, matrix add

- **Optimize for vector*vector**
- **Use existing vector*vector and matrix add kernels**
- **No results yet**
  - **Large kernel invoking several smaller kernels**
  - **Raw fabric not functional yet**
  - **Simulator 10,000x slower**

Raw Quad Processor Board Design Complexities

- **Design Constraints**
  - 11” x 12” 22-layer board
  - 7 power-supply voltages
  - 21700 drilled holes
  - 2124 buried resistors under 4 Raw chips
  - 5723 connections (most high speed)
  - 608 signals per side coming off board
- **Fabrication problems for vendor**
  - Vendor promised delivery based on spec and could not deliver because they couldn’t handle number and density of buried resistors
  - New vendor (used by ISI on other jobs) identified to replicate modules
- **Verification**
  - Expense of each board and value of Raw chips required extensive JTAG testing before bringing boards up with firmware
  - Number of off-chip connections required extensive inter-board testing with I/O board

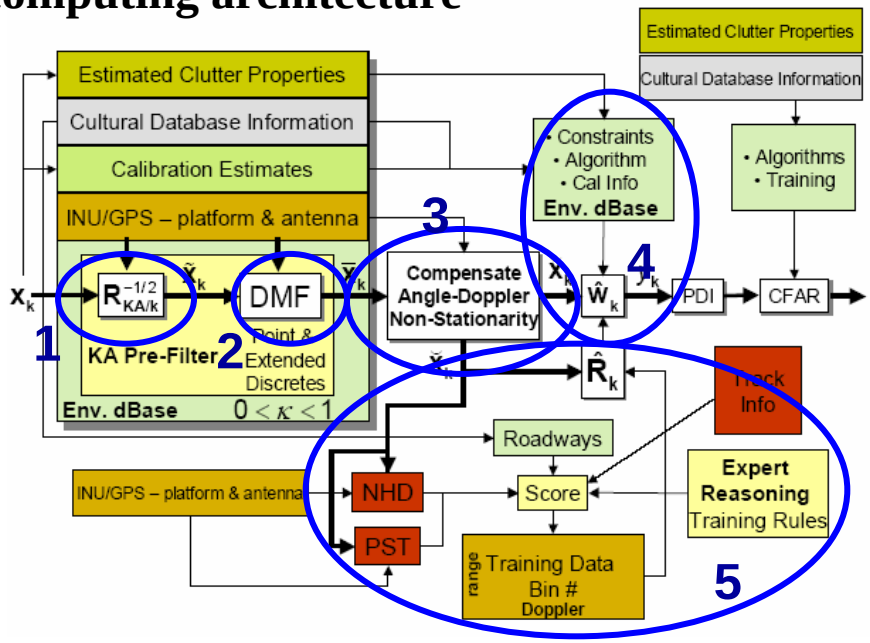
Outline

- Computer Architecture Motivation
- Brief Overview of Prior Research
- ISL Algorithm Update
- Performance Results of Scalability Study
- Future Work
- Summary

KASSPER Cost-Benefit Analysis

- **Determine the Cost-Benefit relationship of KASSPER algorithms**
  - **Profile, analyze and benchmark the computer architecture requirements of several KASSPER algorithms**
  - **Identify points of diminishing returns**
  - **Recommend most cost effective computing architecture-**

- **Metrics**
  - **SINR – Benefit**
  - **Processor sizing – Number of most effective processors**
  - **Memory sizing – Bytes of mass storage, L2 cache**
  - **Cost - $**
  - **Power – Estimate of power**



GTRI Knowledge Aided Architecture

Candidate Architectures

PowerPC G4 or G5

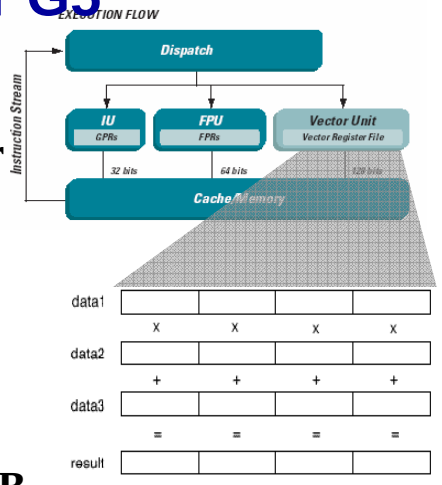
RISC Architecture with Altivec Co-Processor

500 MHz

4 GFLOPS Peak (MAC)

SDRAM: 125 MHz, 256 MB

L2 cache: 250 MHz, 2 MB



Digital Signal Processor

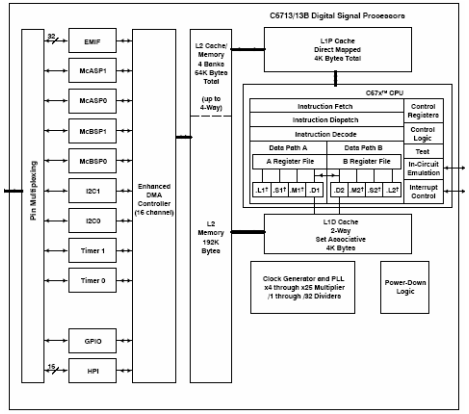
TI TMS320C67xx

VLIW Architecture

1.8 GFLOPS

400 MB/s peak memory access

1.57 Watts



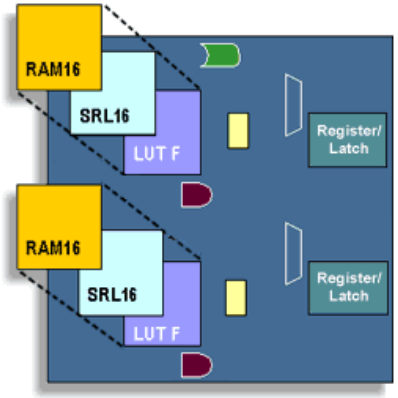
Field Programmable Gate Array

Xilinx Virtex-II Pro or Virtex4

133 GIPS

7.8 GB/s memory access

~30 Watts



HPEC Processors

PCA next generation, ACIP etc

16 tiles of MIPS R4000 @ 425 MHz (chip)

6.8 GOPS or GFLOPS

4 Communication Networks

2 Static Networks,

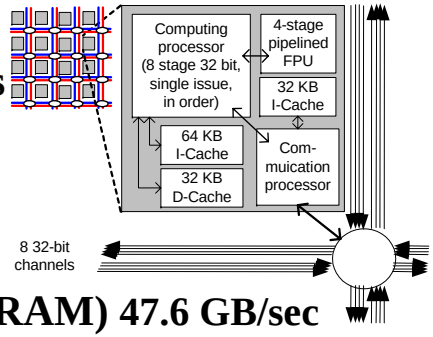
1 cycle throughput

3 cycle latency

54.4 GB / sec

2 Dynamic Networks

14 External Ports (I/O or DRAM) 47.6 GB/sec

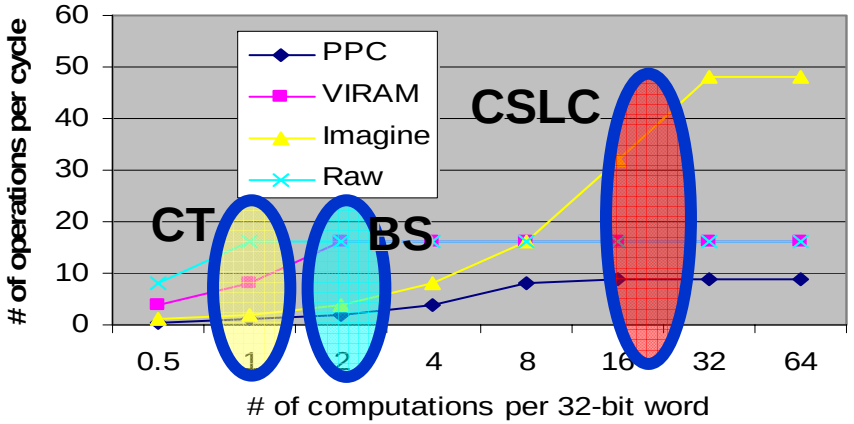


KASSPER
Knowledge-Aided Sensor Signal Processing and Expert Reasoning

DARPA

Approach

- Obtain Matlab from algorithm developer / SPEAR testbed
- Work with algorithm developer to understand assumptions, trade-offs, limitations
- Profile algorithm
  - Matlab profile
  - FLOPS trade-offs
  - Memory / I/O access patterns
- Architecture fit analysis
  - # of FLOPS per data element
  - Cache size limitations
  - Data dependencies



Architecture fit analysis

- Develop key kernels on most promising architectures
- Profile Results
- Feedback / consult with algorithm developer
- Project results to full-system cost
  - Document

Summary

- **HPEC Scalability Study**
  - **Raw Matrix multiply results extremely encouraging**
    - **Kernel efficiency increasing with scaling**
    - **~4x better than equivalent number of PowerPCs**
    - **Tackles ~65% of Pre-Whitening FLOPS**
  - **Vector multiply, matrix addition**
    - **Less efficient**
    - **Raw still on par with PowerPC**
  - **Cholesky**
    - **Waiting for hardware to debug kernel**
- **Cost-Benefit Analysis**
  - **Estimating cost of implementation of promising algorithms**