DARPA STAP-BOY:

Fast Hybrid QR-Cholesky Factorization and Tuning Techniques for STAP Algorithm Implementation on GPU Architectures

Dr. Dennis Healy
DARPA MTO

Dr. Dennis Braunreiter
Mr. Jeremy Furtek
Dr. Nolan Davis
SAIC

Dr. Xiaobai Sun
Duke University

High Performance and Embedded Computing (HPEC) Workshop

18 - 20 September 2007

SAIC
From Science to Solutions

STAP-BOY: Concept

Problem:

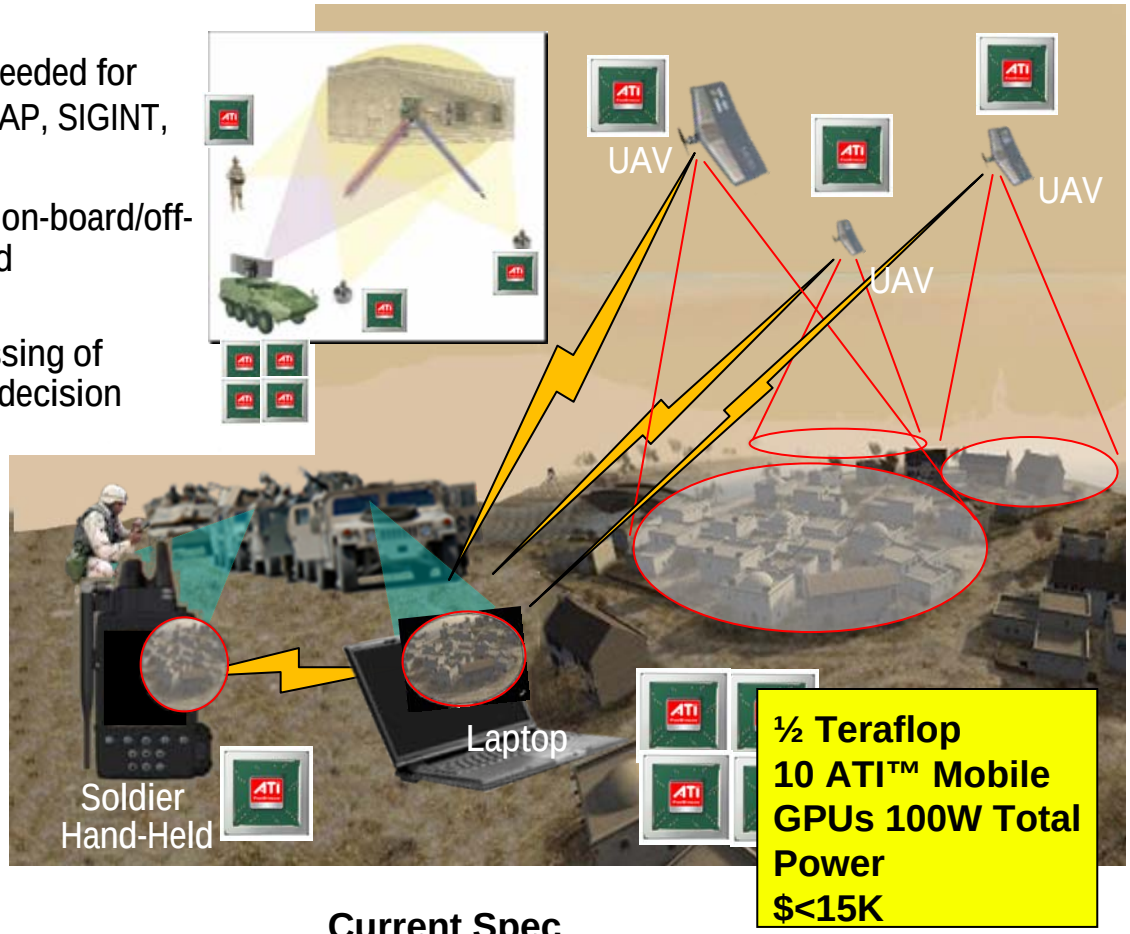
- Complex sensor modalities and algorithms needed for smaller platforms (SAR, 3D-motion video, STAP, SIGINT, ...)
- Low-cost platform constraints limit real-time on-board/off-board and distributed sensing algorithms and performance
- Timely distribution, visualization, and processing of mission-critical data not available to tactical decision makers

STAP-BOY Goal:

- Develop low-cost, scalable, teraflop, embedded multi-modal sensor processing capability based on COTS graphics chips

STAP-BOY Approach:

- Map complex algorithms to COTS graphics chips with open source graphics languages
- Prototype scalable, parallel, embedded computing architecture for handhelds to teraflop single card
- Demonstrate on available, tactically representative sensor systems

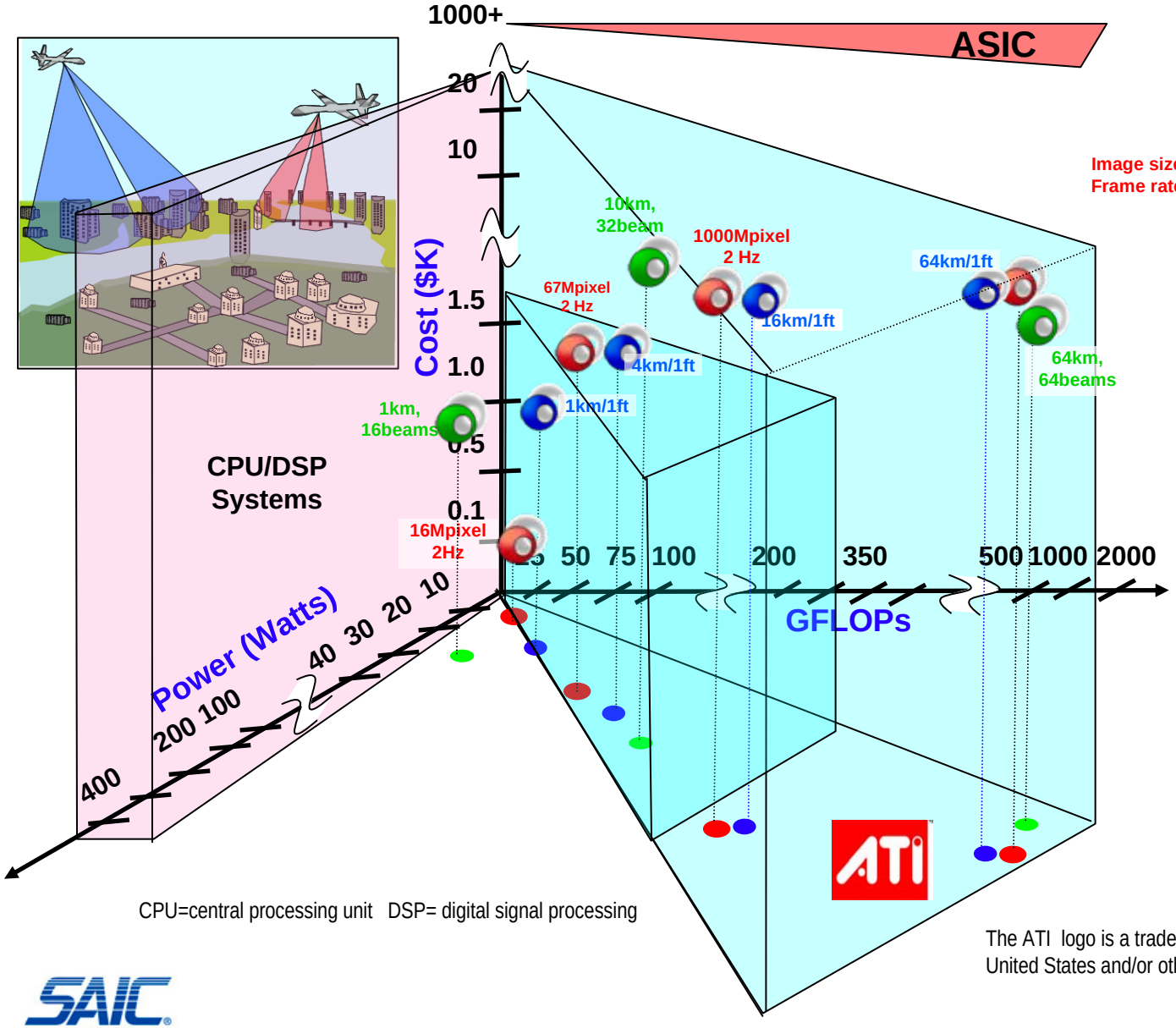


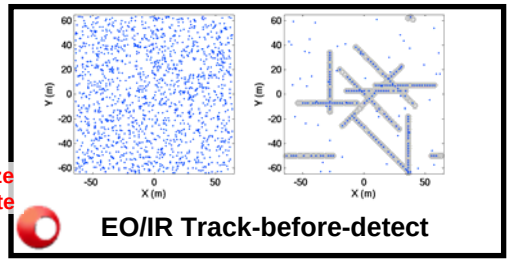
Current Spec

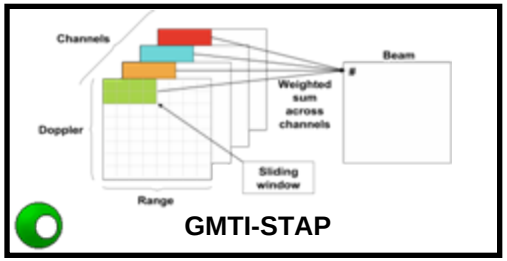
Constant Hawk Advanced EO/IR Processor
100Mpixel camera, 10 GPUs (10kmx10km, 1m)

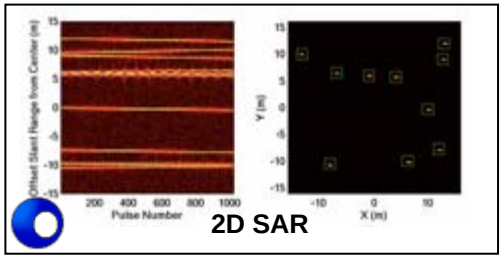
SAIC From Science to Solutions

Applications Pull







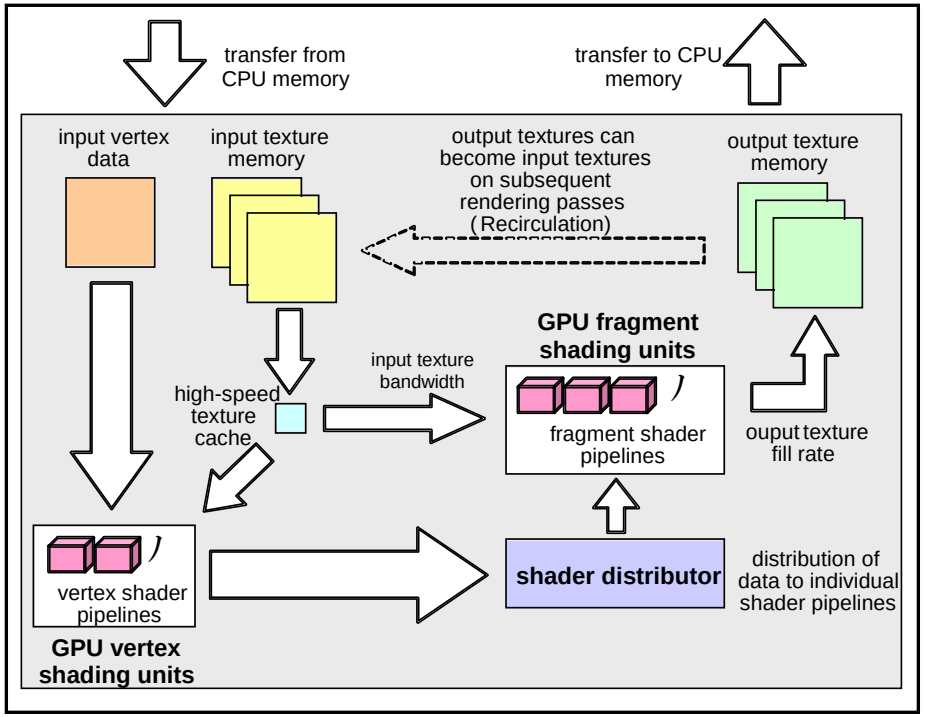


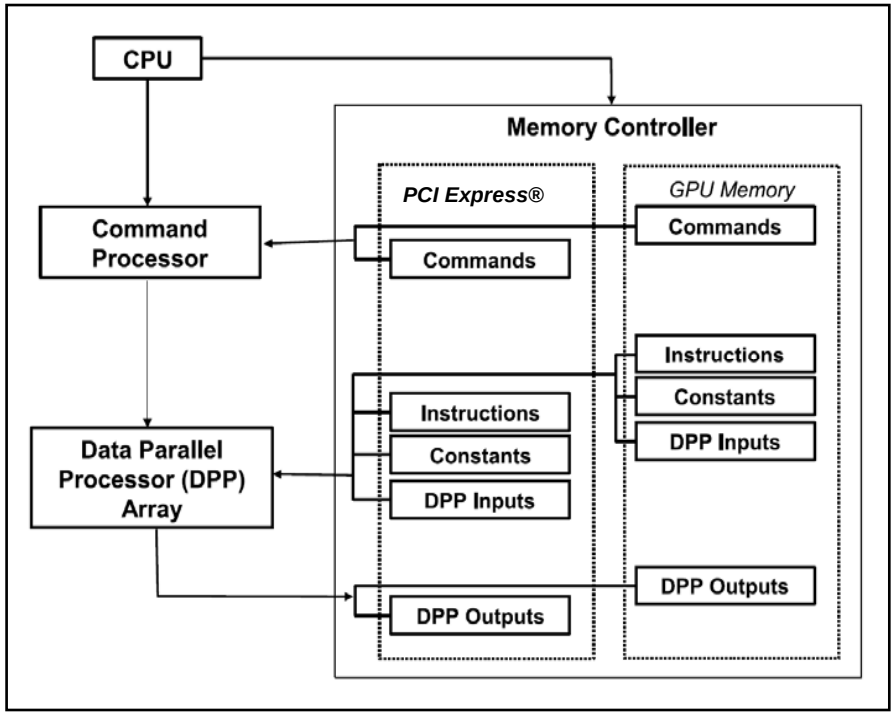
CPU=central processing unit DSP= digital signal processing

CPUs vs. GPUs

Intel® quad-core QX6700		NVIDIA® 8800 GTX
582 million	Transistors	681 million
2.66 GHz	Clock Speed	1.35 Ghz
4	# of Cores	128
Serial	Programming Model	Highly parallel
Minimize latency	Design Goal	Maximize throughput
Complex cores: • Branch prediction • Out-of-order execution	Design Approach	Simple cores: • Smaller caches • In-order execution
43 GFLOPS	Theoretical Max. Computation Rate	346 GFLOPS

OpenGL® Graphics Pipeline Vs. Data Parallel Virtual Machine





- **Requires geometry set-up to perform computation**
  - **Vertex shaders needed to get data into pixel shaders**
  - **More complex graphics programming model**
- **Shader memory access controlled by OpenGL**
  - **Hidden copies and cache control limit pixel shader FLOP performance**

- "Virtual machine" abstraction for GPUs
  - Eliminates complicated graphics programming concepts
  - Exposes hardware as a data-parallel processor array
  - Simplified programming model
- Direct programming and memory management

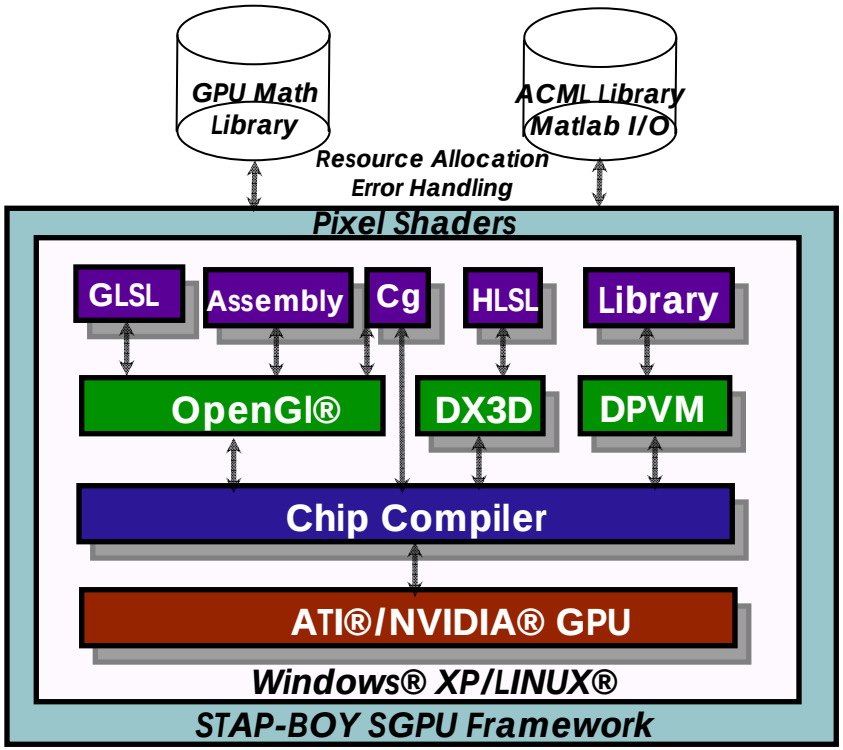
Source: "A Performance-Oriented Data Parallel Virtual Machine for GPUs," Segal, M., and Peercy, M. ACM SIGGRAPH Sketch, 2006.

SAIC From Science to Solutions

Outline

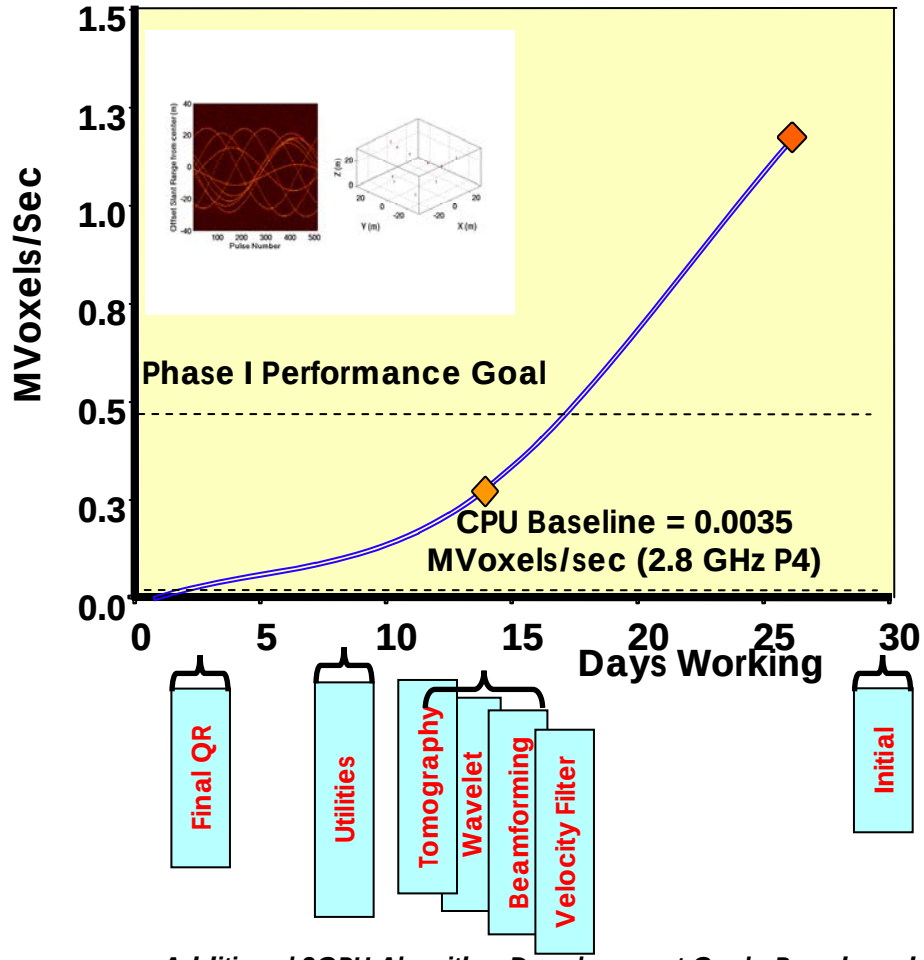
- Algorithms that take advantage of the highly parallel nature of the GPU programming model can run significantly faster than on CPUs
  - Radar STAP
    - Weight Solver:
      - Covariance method is more parallelizable than QR
      - Sliding window algorithm results in additional speed-up
    - STAP beamforming: matrix-matrix multiply is fast on GPU
  - Spin Images
    - Spin-image matching component: parallel over model and scene points, reduction over image pixels
    - Geometric consistency component: parallel over pairs of point correspondences
  - SAR/Tomography
- Continuing advances in GPU hardware and stream software will enable single chip solutions for a large class of STAP airborne applications and similarly sized problems

Productivity



STAP-BOY Integrated Development Environment

- 100% COTS and/or open source
- 42,000 lines of code
- Cross platform suite of libraries
- Automation of common tasks
- Utilities developed by college interns



Additional SGPU Algorithm Development Cycle Benchmarks

Weight Solver Methods

QR Method

$$A = \begin{bmatrix} y_1 & y_2 & \cdots & y_{N_T-1} & y_{N_T} \\ y_2 & y_3 & \cdots & y_{N_T} & y_{N_T+1} \\ y_3 & y_4 & \cdots & y_{N_T+1} & y_{N_T+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{M_1-2} & y_{M_1-1} & \cdots & y_{M_1+N_T-4} & y_{M_1+N_T-3} \\ y_{M_1-1} & y_{M_1} & \cdots & y_{M_1+N_T-3} & y_{M_1+N_T-2} \\ y_{M_1} & y_{M_1+1} & \cdots & y_{M_1+N_T-2} & y_{M_1+N_T-1} \end{bmatrix}$$

Data Matrix A

QA=R
$R^TRx=y$
Solve for x

Covariance Method

$$\Lambda \propto \begin{bmatrix} \sum_{i=1}^{M_1} y_i^* y_i & \sum_{i=1}^{M_1} y_i^* y_{i+1} & \cdots & \sum_{i=1}^{M_1} y_i^* y_{i+N_T-2} & \sum_{i=1}^{M_1} y_i^* y_{i+N_T-1} \\ \sum_{i=1}^{M_1} y_{i+1}^* y_i & \sum_{i=1}^{M_1} y_{i+1}^* y_{i+1} & \cdots & \sum_{i=1}^{M_1} y_{i+1}^* y_{i+N_T-2} & \sum_{i=1}^{M_1} y_{i+1}^* y_{i+N_T-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{M_1} y_{i+N_T-2}^* y_i & \sum_{i=1}^{M_1} y_{i+N_T-2}^* y_{i+1} & \cdots & \sum_{i=1}^{M_1} y_{i+N_T-2}^* y_{i+N_T-2} & \sum_{i=1}^{M_1} y_{i+N_T-2}^* y_{i+N_T-1} \\ \sum_{i=1}^{M_1} y_{i+N_T-1}^* y_i & \sum_{i=1}^{M_1} y_{i+N_T-1}^* y_{i+1} & \cdots & \sum_{i=1}^{M_1} y_{i+N_T-1}^* y_{i+N_T-2} & \sum_{i=1}^{M_1} y_{i+N_T-1}^* y_{i+N_T-1} \end{bmatrix}$$

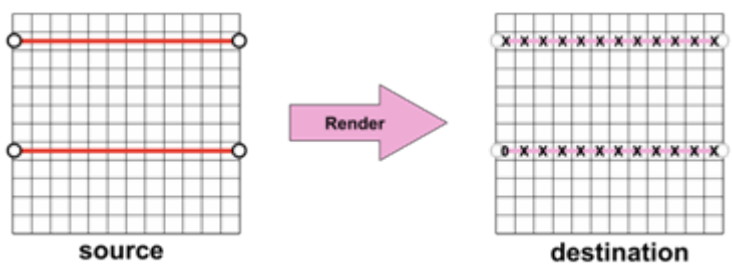
Covariance Matrix $\not\subset$

$\not\subset = A^TA$
$L^TLx=y$
Solve for x

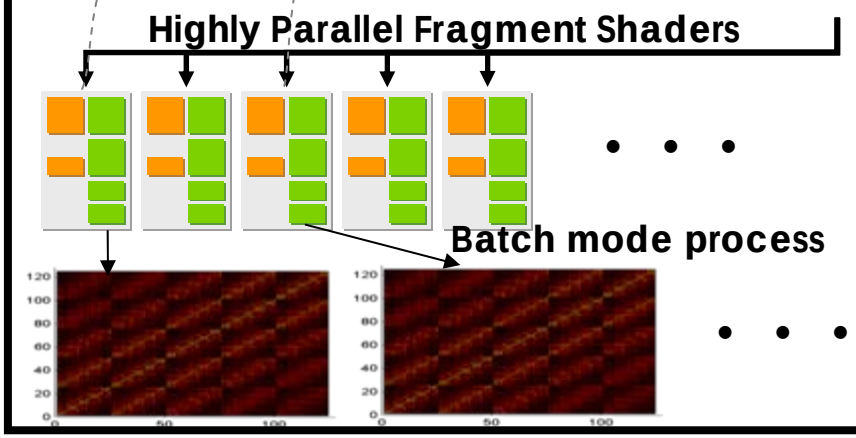
$R^T == L$

GPU Implementation

$$A([i,k],:) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A([i,k],:) \qquad c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}} \qquad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}$$

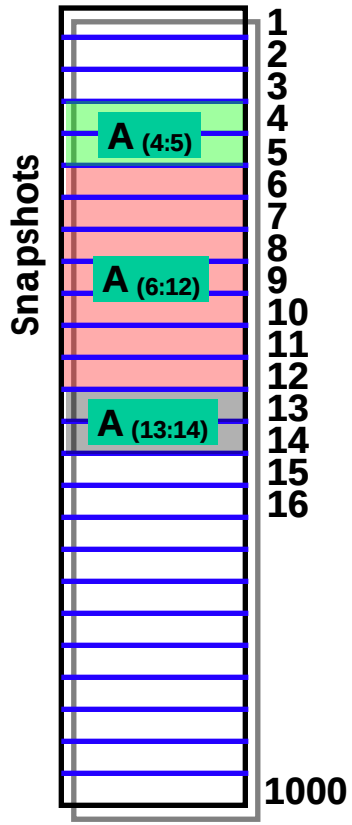


GPU Implementation



Covariance matrix method yields identical mathematical solution to QR and exploits 2-D matrix operations in a highly parallel fashion

Shared-row Covariance Method: Algorithm Steps



- **Estimate covariance matrix of the shared rows (6:12)**

$$C_s = \sum_{l=6}^{12} \frac{1}{6} A_l A_l^T$$

where A_l is a snapshot vector

- **If covariance matrix is block Toeplitz**

$$C_s = \sum_{l=6}^{12} \frac{1}{6} A_{l(1:n)} A_{l(1:n/k)}^T$$

- **Compute Cholesky factorization of shared-row covariance matrix**

$$C_s = L_s L_s^T$$

where L_s is lower triangular

- **Update Cholesky Factors using shared row method (derived on next slide)**

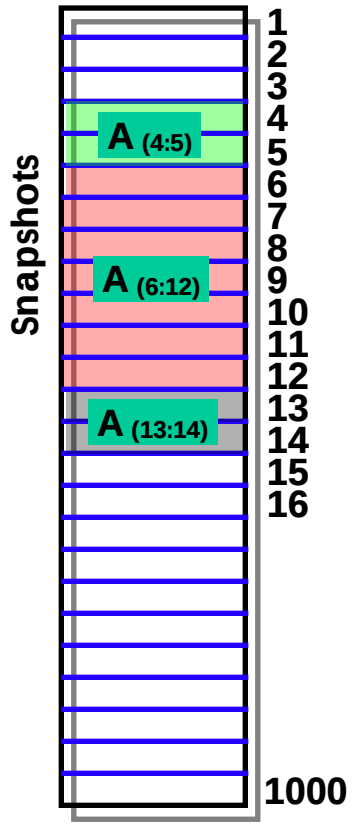
$$\begin{bmatrix} L_{(4:12)} \\ 0 \end{bmatrix} = H \begin{bmatrix} L_s \\ A_{(4)} \\ A_{(5)} \end{bmatrix} \qquad \begin{bmatrix} L_{(5:13)} \\ 0 \end{bmatrix} = H \begin{bmatrix} L_s \\ A_{(5)} \\ A_{(13)} \end{bmatrix} \qquad \begin{bmatrix} L_{(6:14)} \\ 0 \end{bmatrix} = H \begin{bmatrix} L_s \\ A_{(13)} \\ A_{(14)} \end{bmatrix}$$

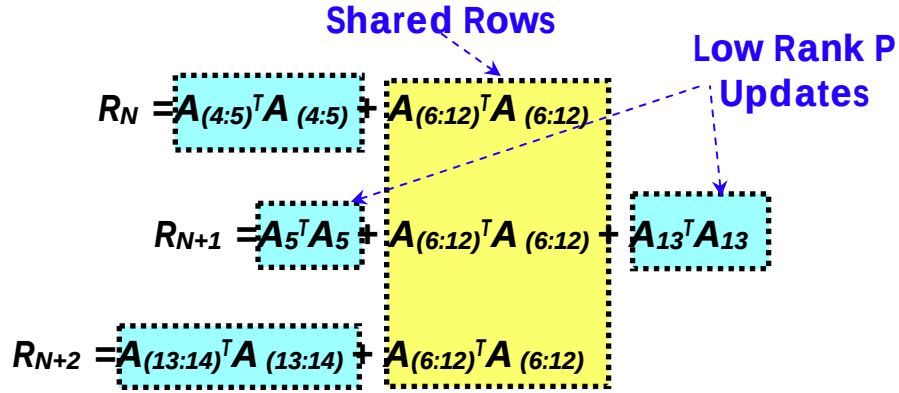
H can be a sequence of Givens or Householder rotations
Now we have computed the following Cholesky factors:

$$C_{(4:12)} = L_{(4:12)} L_{(4:12)}^T \qquad C_{(5:13)} = L_{(5:13)} L_{(5:13)}^T \qquad C_{(6:14)} = L_{(6:14)} L_{(6:14)}^T$$

Modification from Golub and Van Loan, 1996

Shared-Row Covariance Method: Low-Rank Updates





- **Method for Low Rank Update of Cholesky Factor***

$$L_{N+2}^T L_{N+2} = A_{(6:12)}^T A_{(6:12)} + A_{(13:14)}^T A_{(13:14)}$$
$$= L_{(6:12)}^T L_{(6:12)} + A_{(13:14)}^T A_{(13:14)}$$
$$= [L^T_{(6:12)} \quad A^T_{(13:14)}] \begin{bmatrix} I_N & 0 \\ 0 & I_p \end{bmatrix} \begin{bmatrix} L_{(6:12)} \\ A_{(13:14)} \end{bmatrix}$$

- **Goal is to Find an H such that**

$$H^T H = I_{(n+p)}$$

$$H \begin{bmatrix} L_{(6:12)} \\ A_{(13:14)} \end{bmatrix} = \begin{bmatrix} L_{N+2} \\ 0 \end{bmatrix}$$

- **H can be a sequence of Givens or Householder rotations**

Modification from Golub and Van Loan, 1996

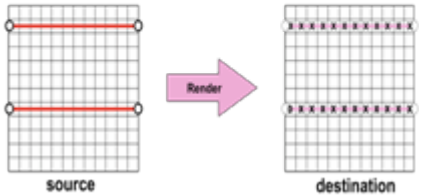
Total Speedup for the STAP Algorithm

STAP Weights Solution

Performance Parameter	Phase One Goals (+12months)	CPU Performance	STAP-BOY GPU Performance
Matrix Size	384K x 128K	384K X 128K	384K X 128K
# Updates	1000	1000	1000
# of Nodes	1	1	1
Computation Time	30 ms	4900 ms	300 ms
Throughput	50 GFLOPS	3	6.2*/64**

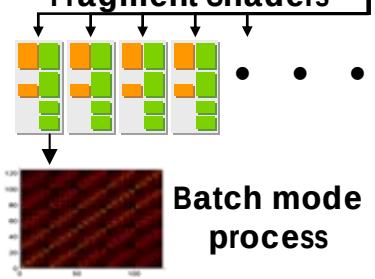
***QR Solver**

$$A([i,k],:) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A([i,k],:) \qquad c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}} \qquad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}$$



****Covariance Solver**

Highly Parallel Fragment Shaders



Batch mode process

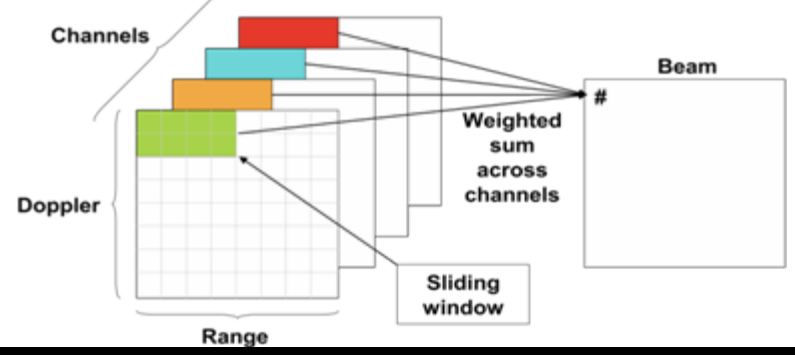
* Throughput for QR Decomposition

** Throughput for matrix-matrix multiply

STAP Beamforming

Performance Parameter	Definition	CPU Performance	STAP-BOY GPU Performance
Filter Size	DopplerxRangexChannel	256x1000x16	256x1000x16
Computation Time	ms	760 ms	32 ms
Throughput	GFLOPs	0.36	8.1

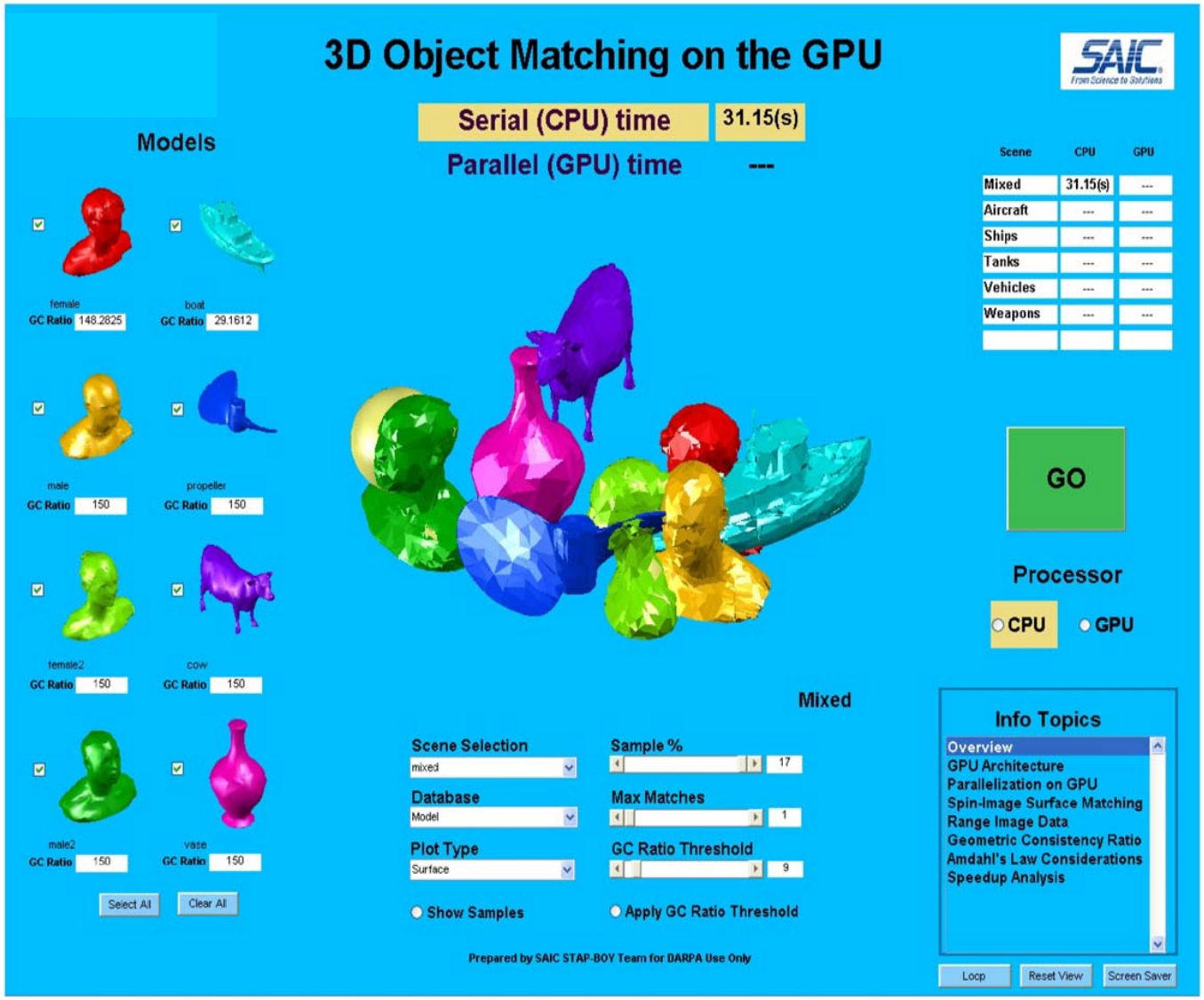
- 128x1 vector formed by 4x2 window across 16 channels
- 128x1 weight vector stored in memory
- Output is dot-product of weight vector with data vector
- Data window moves for each pixel in range doppler map



In Both Cases, Demonstrated One to Two Order Magnitude Speedup Over 64-Bit State-of-the-Art CPUs

SAIC From Science to Solutions

Interpreting Range with Spin-Image Mapping



Spin-Image Surface Mapping

- **Spin-image Matching**
  - **For each sample scene point, compare to all model points**
  - **Match using image correlation**
- **Geometric consistency**
  - **Find pairs of point correspondences with best spin-coordinate match**
- **Transformations**
  - **Best pair of point correspondences determines a transformation that maps the model into the scene**

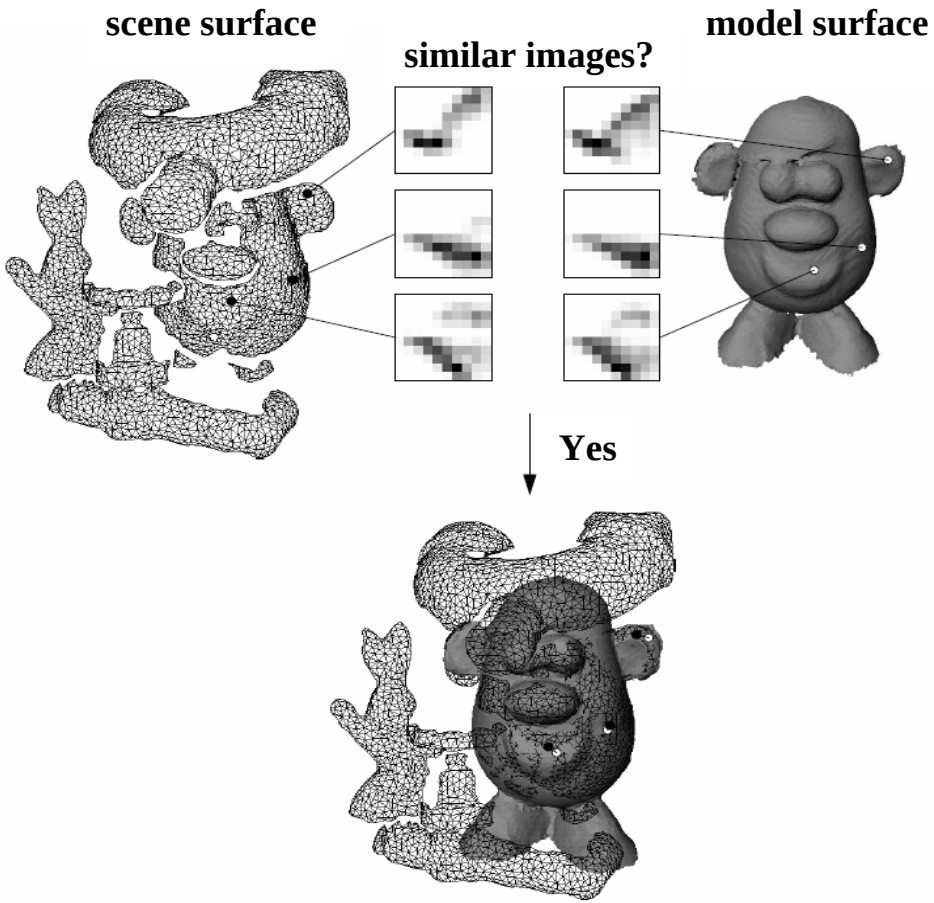


Figure 1-2: Surface matching concept*

6

*A. Johnson, *Spin-Images: A Representation for 3-D Surface Matching,* doctoral dissertation, The Robotics Institute, Carnegie Mellon Univ., 1997.

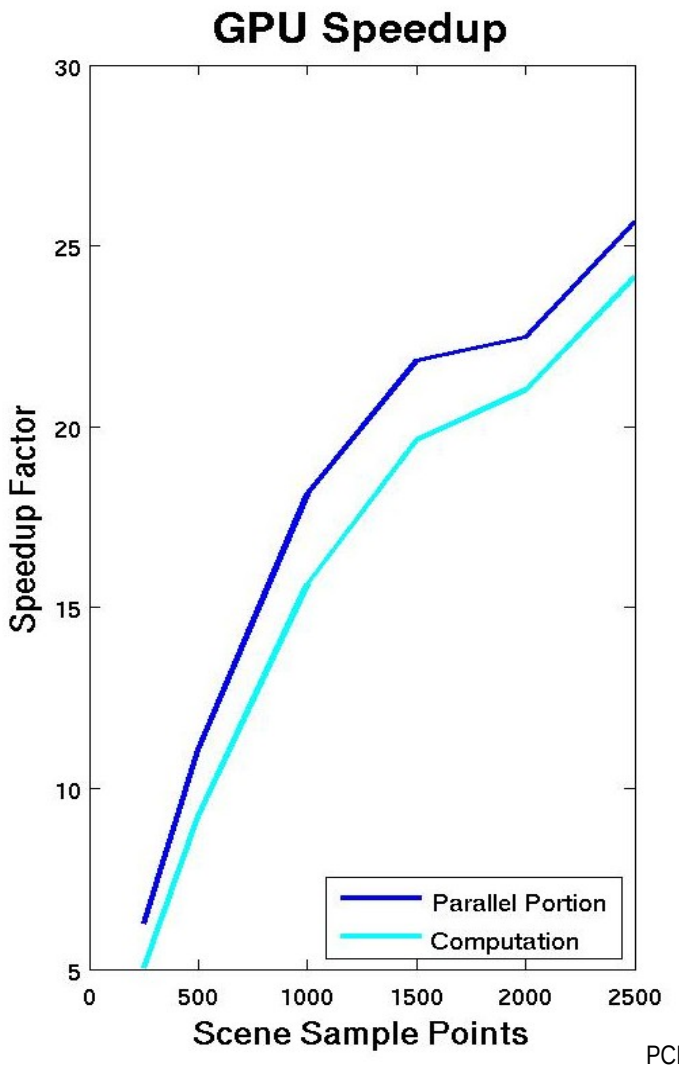
Parallel Processing Opportunities

- **Spin-image matching component**
  - **Image-correlation-based statistic**
    - **Parallel over model and scene points**
    - **Reduction over image pixels**
    - **O(W*H*P*M*S) for WxH spin-image at P model points on each of M models with S sample scene points**
- **Geometric consistency component**
  - **Coordinate match statistic**
    - **Parallel over pairs of point correspondences**
    - **O(M*N^2) for N point correspondences for each of M models**

Achieving Speedup

- **Offload explicitly parallel portions to the GPU**
  - ➢ **Spin-image correlation**
  - ➢ **Spin-image coordinate matching**
  - – **Bulk of processing time (Time Reduction regime)**
  - – **Only 2 times -3 times speedup**
- **Address less obvious parallelizations**
  - ➢ **Geometric consistency thresholding**
  - – **Where not fully parallelizable in current API, then do minimal amount on CPU and utilize GPU/CPU shared memory to reduce data transport.**
  - – **Eliminated most of remaining serial time (Transition regime)**
  - – **8 times – 11 times speedup**
- **Consolidate code on GPU to minimize data upload/download**
  - – **Small reductions in overall time gave large increases in speedup (Data Throughput regime)**
  - – **20 times - 24 times speedup**

GPU Speedup & Timing

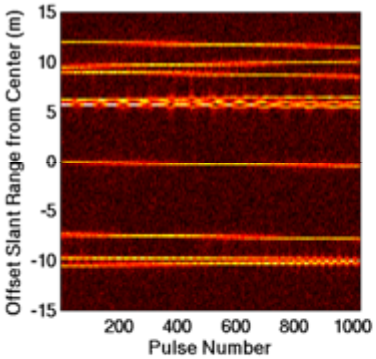


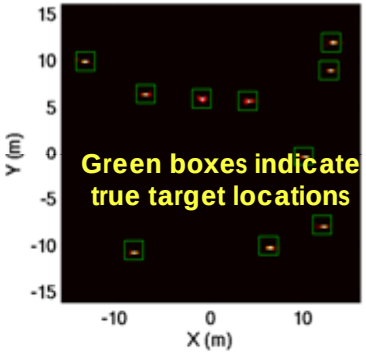
- **Graphics card: ATI™ X1900 XTX**
  - **48 pixel shaders @ 640 MHz**
  - **GPU Memory 512 MB**
  - **GPU Memory bandwidth 1550 MHz**
- **CPU: Xeon® 2800 MHz**
- **Comms: PCI Express®**
  - **250 MB/s each direction, per lane**
  - **16 lanes: 4 GB/s**

Additional results

2D SAR/Tomographic Reconstruction

Performance Parameter	Definition	CPU Performance	STAP-BOY GPU Performance
Matrix Size	Range (ft) x Crossrange (ft)	2048 x 2048	2048 x 2048
Computation Time	sec	1171.3 sec	7.35 sec
Speedup	GPU/CPU	0.006	159.4
Throughput	GFLOPs	0.132	21

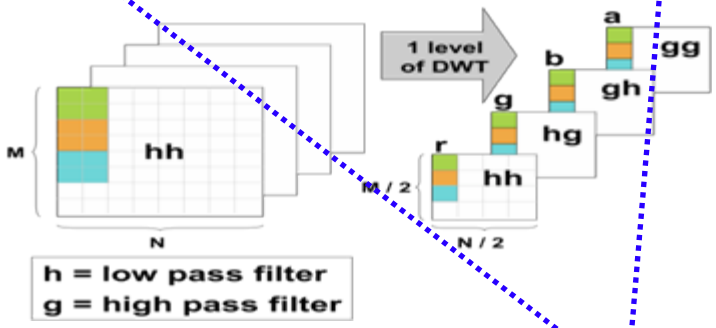




2D Wavelet Transform (Daubechies-6)

Performance Parameter	Definition	CPU Performance	STAP-BOY GPU Performance
Matrix Size	Number of Pixels	1024 x 1024	1024 x 1024
Computation Time	sec	0.953	0.015
Speedup	GPU/CPU	0.016	60
Throughput	GFLOPS	0.36	12

- Motivation: fast numerical linear algebra, sparse matrix representation, QR decomposition
- Non-standard form: HH, HL, LH, LL stored in 4 color textures
- Recirculation of LL to process next level of resolution tree



STAP-BOY Signal Processing Implementations Demonstrated Almost Two Order Magnitude Speedup over State-of-the-Art CPU with Three-Week Development Cycles

SAIC From Science to Solutions

Summary

- Algorithms that take advantage of the highly parallel nature of the GPU programming model can run significantly faster than on CPUs
  - Radar STAP
    - Weight Solver:
      - Covariance method is more parallelizable than QR
      - Sliding window algorithm results in additional speed-up
    - STAP beamforming: matrix-matrix multiply is fast on GPU
  - Spin Images
    - Spin-image matching component: parallel over model and scene points, reduction over image pixels
    - Geometric consistency component: parallel over pairs of point correspondences
  - SAR/Tomography
- Continuing advances in GPU hardware and stream software will enable single chip solutions for a large class of STAP airborne applications and similarly sized problems