# Real-Time Adaptive Signal Processing Development for US Navy Torpedoes

Matthew A. Alexander
maa@ll.mit.edu

HPEC 2006

20 Sept 2006

This work was sponsored by the Department of the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author, and are not necessarily endorsed by the United States Government. Reference to any specific commercial product, trade name, trademark or manufacturer does not constitute or imply endorsement.

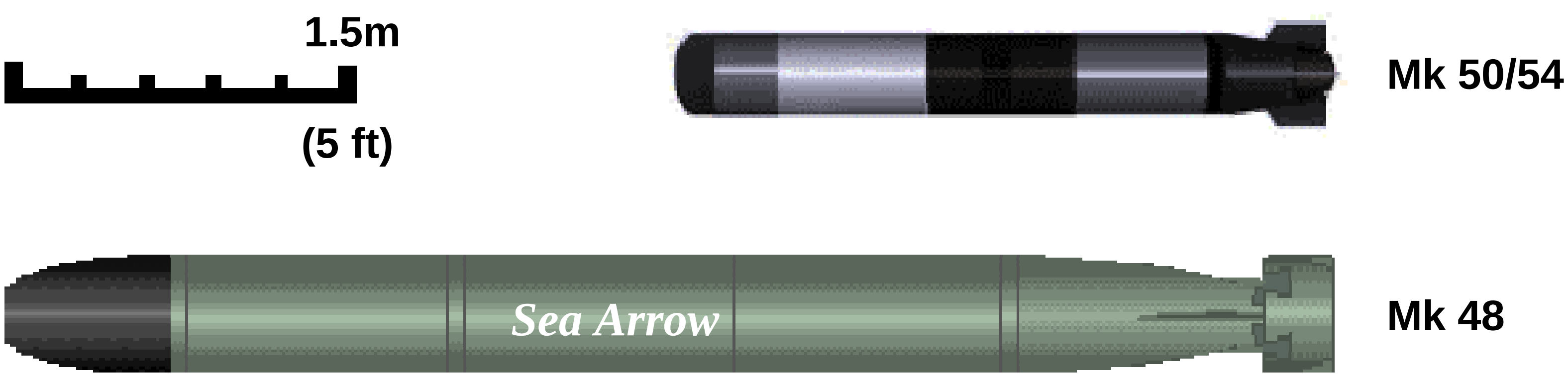
The author wishes to acknowledge the contributions of MIT/LL staff Francine Rayson, Dr. James C. Anderson, and Dr. Nigel Lee, as well as NUWC/Npt staff Dr. Adam Mirkin

# Outline

- Introduction
- Implementation
- Results
- Summary

# Background on Mk 48 / Mk 54 US Navy Torpedoes

- **Mk 48 heavyweight torpedo carried by all U.S. attack (SSN) & ballistic missile (SSBN) subs since 1972 (ADvanced CAPability since 1988)**
  - **Used against deep-diving nuclear subs & littoral diesel subs (SSK)**
  - **50 km range @ 74 km/hr or 38 km @ 102 km/hr (JUWS)**
- **Mk 54 LHT (lightweight hybrid torpedo) has *<u>computer hardware & software commonality with Mk 48</u>***
  - **Mk 54 anti-submarine torpedo launched from surface ship or air-dropped**
  - **Mk 50 (1992) being replaced by Mk 54 (2004)**
  - **20 km range @ 83.4 km/hr (JNWS)**



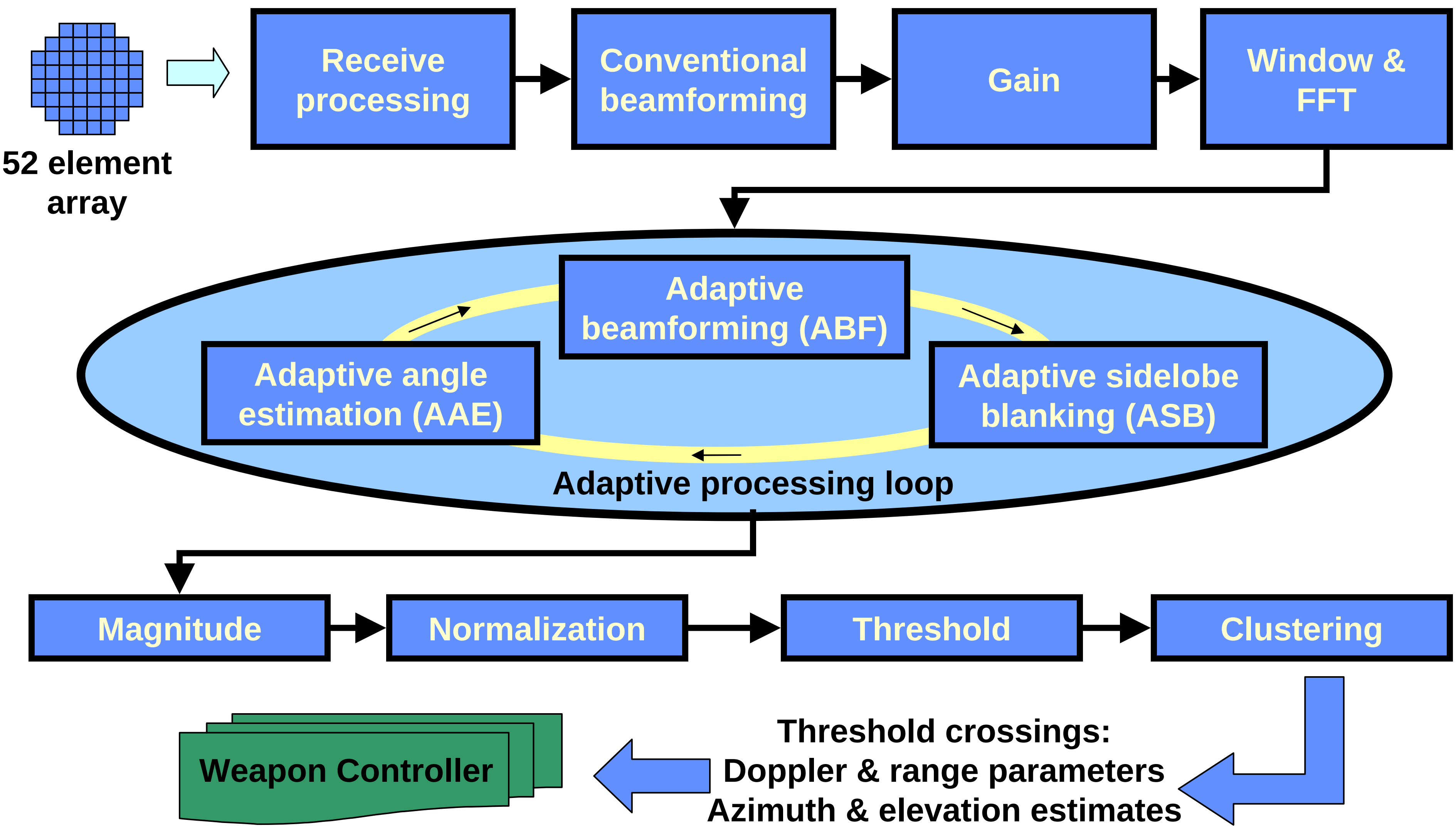
# MIT-LL Involvement in Torpedo Development Process

- **Current Mk 48 and Mk 54 signal processing use "conventional" (non-adaptive) processing**
  - **Conventional beamforming, conventional angle estimation**
  - **Performs well in "clean" scenarios (targets in ambient noise)**

- **MIT-LL is working with NUWC/Npt to implement adaptive signal processing stream to improve torpedo performance in countered scenarios**
  - **Adaptive beamforming rejects energy from nearby countermeasures to uncover targets**
  - **Adaptive angle estimation rejects countermeasure energy to provide unbiased target angle estimates**
  - **Adaptive processing provides improved performance but also requires more computations**

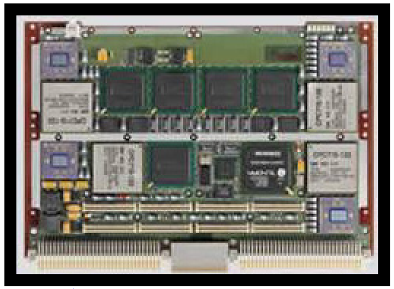
**MIT-LL is developing real-time implementation of torpedo adaptive signal processing stream**

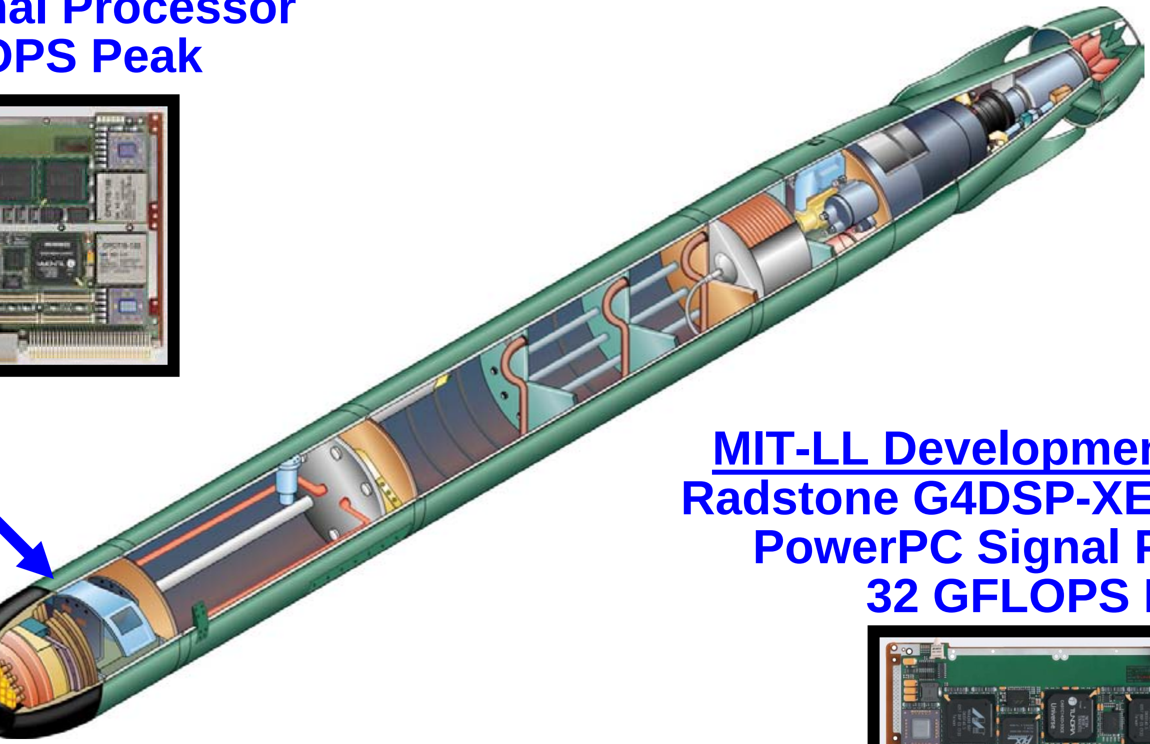
# Torpedo Adaptive Signal Processor (TASP) Stream



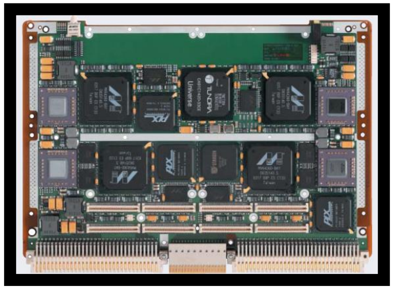
# Signal Processor Prototyping Hardware

**Torpedo Hardware**
**Radstone G4DSP Quad 400 MHz PowerPC Signal Processor**
**12.8 GFLOPS Peak**





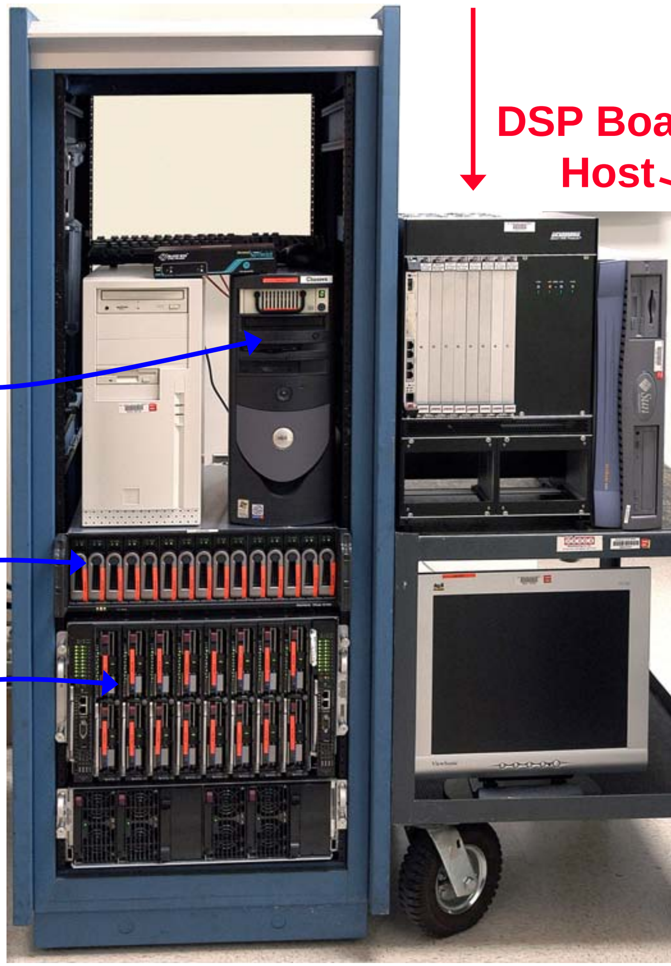
**MIT-LL Development Hardware**
**Radstone G4DSP-XE Quad 1 GHz PowerPC Signal Processor**
**32 GFLOPS Peak**





GFLOPS = Billions of floating point operations per second

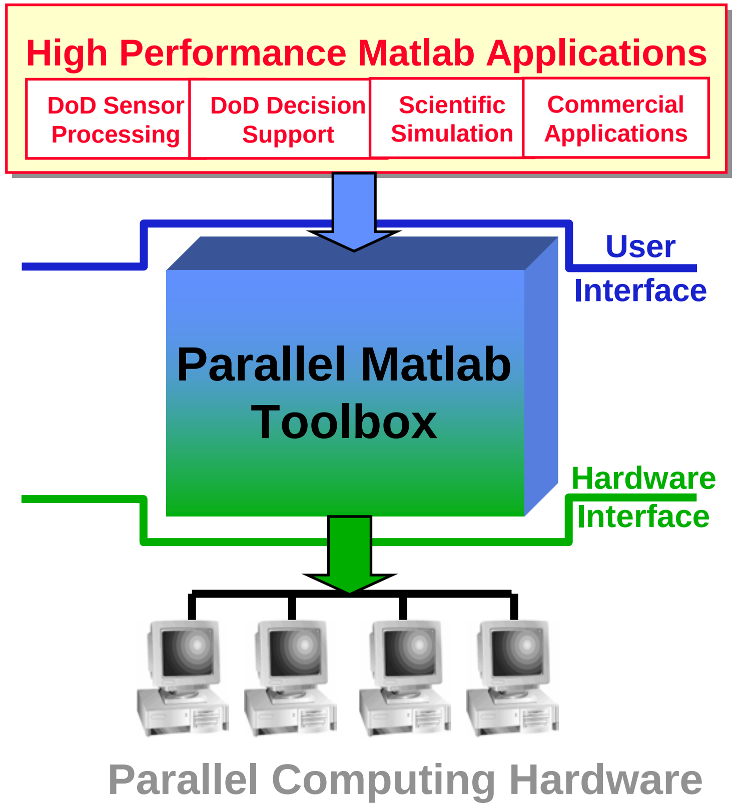
# MIT Torpedo Advanced Processor Build (TAPB) Testbed




**Lab constructed to support algorithm development and computational assessment**

- HP ProLiant cluster (initial development)
  - Supports parallel Matlab, C++, and parallel C++ processing
  - Provides data server
- Radstone G4DSP-XE Quad PPC (emulates target platform)
  - Final C++ and parallel C++ implementations
  - Provides real-time analysis platform

# Parallel Matlab (pMatlab)




**Messaging**

```
B(:,:)=fft(A)
```

```
my_rank=MPI_Comm_rank(comm);
if (my_rank==0)|(my_rank==1)|(my_rank==2)|(my_rank==3)
  A_local=rand(M,N/4);end
if (my_rank==4)|(my_rank==5)|(my_rank==6)|(my_rank==7)
  B_local=zeros(M/4,N);end
A_local=fft(A_local);
tag=0;if (my_rank==0)...MPI_Send(4,tag,comm,A_local(1:M/4,:);
elseif (my_rank==4)...B_local(:,1:N/4) = MPI_Recv(0,tag,comm);end
tag = tag+1;if (my_rank==0)...MPI_Send(5,tag,comm,A_local(M/4+1:2M/4,:);
elseif (my_rank==5)...B_local(:,1:N/4) = MPI_Recv(0,tag,comm);end
tag=tag+1;if (my_rank==0)...MPI_Send(6,tag,comm,A_local(2M/4+1:3M/4,:);
elseif (my_rank==6)...B_local(:,1:N/4) = MPI_Recv(0,tag,comm);end
tag=tag+1;if (my_rank==0)...MPI_Send(7,tag,comm,A_local(3M/4+1:M,:);
elseif (my_rank==7)...B_local(:,1:N/4) = MPI_Recv(0,tag,comm);end
tag=tag+1;if (my_rank==1)...MPI_Send(4,tag,comm,A_local(1:M/4,:);
elseif (my_rank==4)...B_local(:,N/4+1:2N/4) = MPI_Recv(1,tag,comm);end
tag=tag+1;if (my_rank==1)...MPI_Send(5,tag,comm,A_local(M/4+1:2M/4,:);
elseif (my_rank==5)...B_local(:,N/4+1:2N/4) = MPI_Recv(1,tag,comm);end
tag=tag+1;if (my_rank==1)...MPI_Send(6,tag,comm,A_local(2M/4+1:3M/4,:);
elseif (my_rank==6)...B_local(:,N/4+1:2N/4) = MPI_Recv(1,tag,comm);end
tag=tag+1;if (my_rank==1)...MPI_Send(7,tag,comm,A_local(3M/4+1:M,:);
elseif (my_rank==7)...B_local(:,N/4+1:2N/4) = MPI_Recv(1,tag,comm);end
tag=tag+1;if (my_rank==2)...MPI_Send(4,tag,comm,A_local(1:M/4,:);
elseif (my_rank==4)...B_local(:,2N/4+1:3N/4) = MPI_Recv(2,tag,comm);end
tag=tag+1;if (my_rank==2)...MPI_Send(5,tag,comm,A_local(M/4+1:2M/4,:);
elseif (my_rank==5)...B_local(:,2N/4+1:3N/4) = MPI_Recv(2,tag,comm);end
tag=tag+1;if (my_rank==2)...MPI_Send(6,tag,comm,A_local(2M/4+1:3M/4,:);
elseif (my_rank==6)...B_local(:,2N/4+1:3N/4) = MPI_Recv(2,tag,comm);end
tag=tag+1;if (my_rank==2)...MPI_Send(7,tag,comm,A_local(3M/4+1:M,:);
elseif (my_rank==7)...B_local(:,2N/4+1:3N/4) = MPI_Recv(2,tag,comm);end
tag=tag+1;if (my_rank==3)...MPI_Send(4,tag,comm,A_local(1:M/4,:);
elseif (my_rank==4)...B_local(:,3N/4+1:N) = MPI_Recv(3,tag,comm);end
tag=tag+1;if (my_rank==3)...MPI_Send(5,tag,comm,A_local(M/4+1:2M/4,:);
elseif (my_rank==5)...B_local(:,3N/4+1:N) = MPI_Recv(3,tag,comm);end
tag=tag+1;if (my_rank==3)...MPI_Send(6,tag,comm,A_local(2M/4+1:3M/4,:);
elseif (my_rank==6)...B_local(:,3N/4+1:N) = MPI_Recv(3,tag,comm);end
tag=tag+1;if (my_rank==3)...MPI_Send(7,tag,comm,A_local(3M/4+1:M,:);
elseif (my_rank==7)...B_local(:,3N/4+1:N) = MPI_Recv(3,tag,comm);end
```

**Distributed Objects**

```
B(:,:)=fft(A)
```

```
mapA = map([1 4],{},[0:3]);
mapB = map([4 1],{},[4:7]);
A = rand(M,N,mapA);
B = zeros(M,N,mapB);
B(:,:) = fft(A);
```

**Automated Maps**

```
B(:,:)=fft(A)
```

```
A = rand(M,N,p);
B = zeros(M,N,p);
B(:,:) = fft(A);
```

## Goals

**Matlab speedup through transparent parallelism**
**Near-real-time rapid prototyping**

# Parallel Vector Library (PVL)

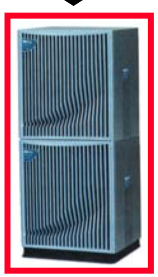


Portable, scalable middleware for high performance array signal processing applications

Application Components (Reusable)

Scalable Components

Portable Kernels




VSIPL = Vector signal image processing library
SSE = Streaming SIMD (Single instruction, multiple data) extensions

# Parallel Vector Library (PVL)



Portable, scalable middleware for high performance array signal processing applications

Application Components (Reusable)

Scalable Components

Portable Kernels

Workstation Simulation

Real-Time Cluster

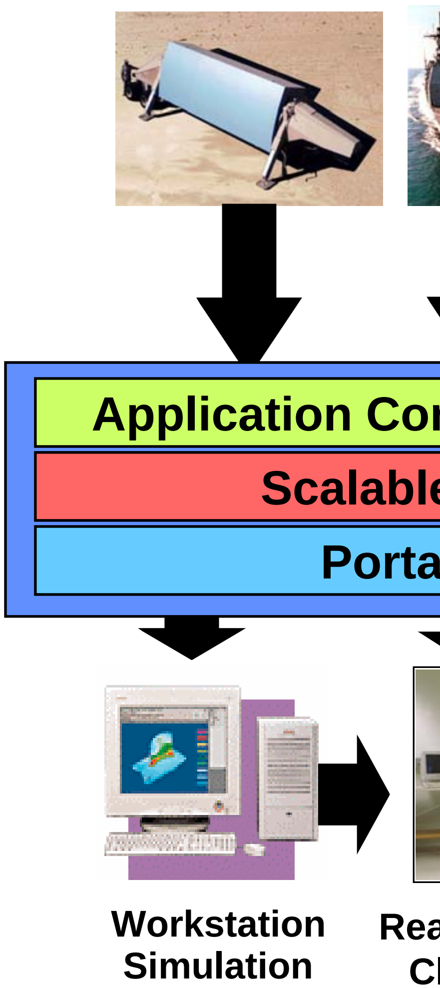
Real-Time Embedded Platform

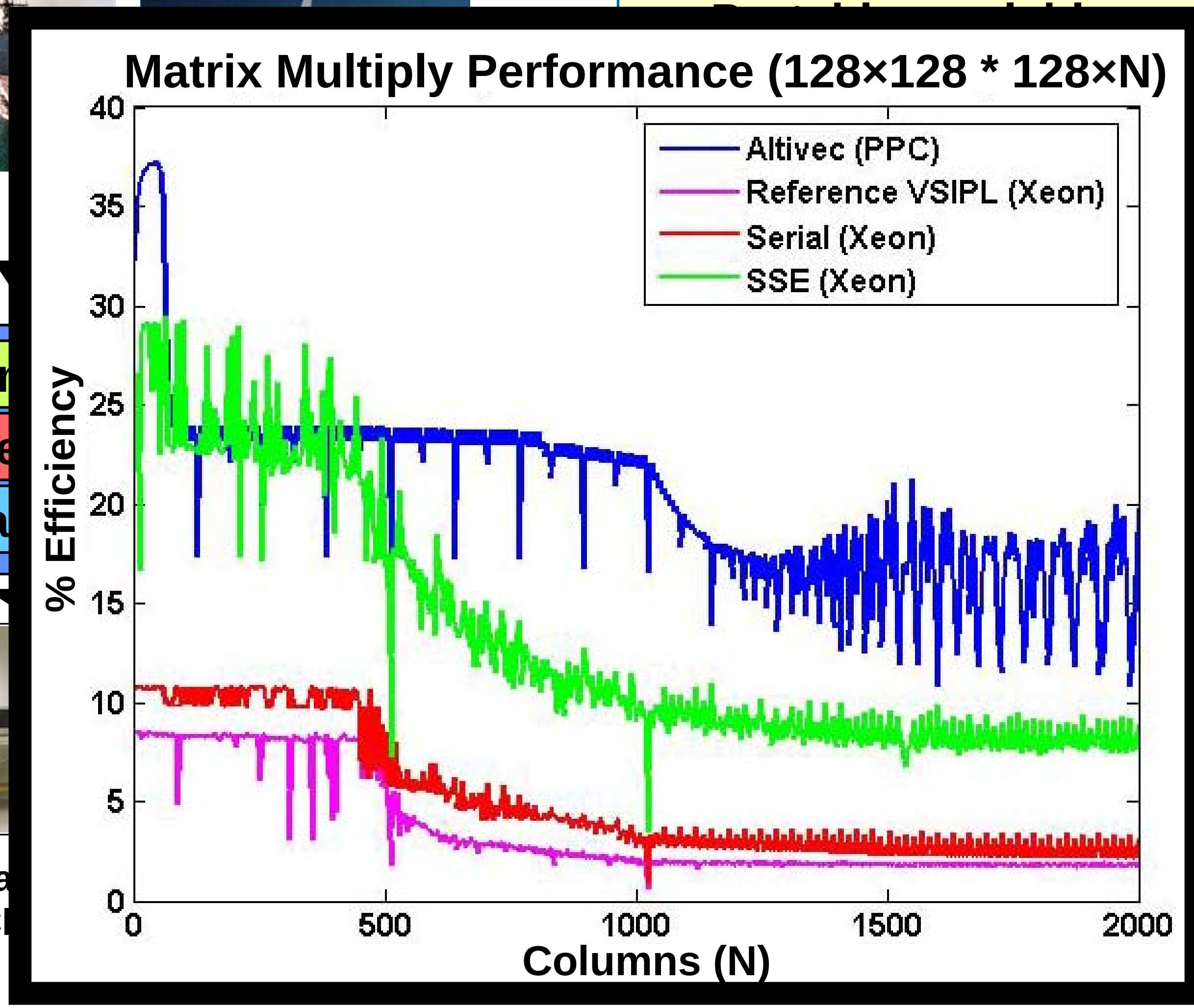
Workstations to “Roll-on/Roll-Off” Rapid Prototyping
- Functionality
- Hardware Mapping
- Real-time



VSIPL = Vector signal image processing library
SSE = Streaming SIMD (Single instruction, multiple data) extensions

# Parallel Vector Library (PVL)



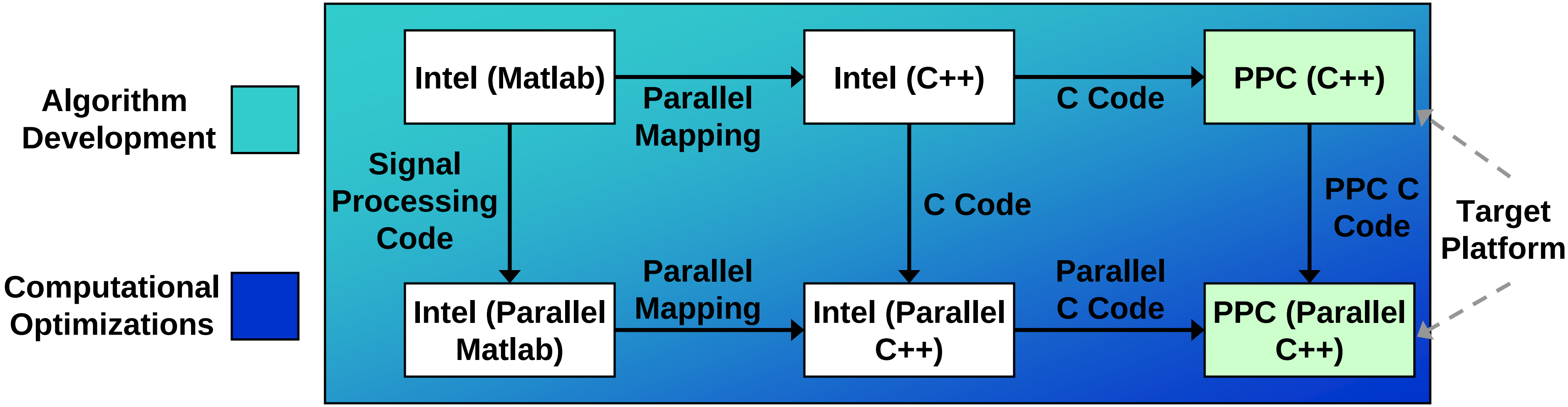



VSIPL = Vector signal image processing library
SSE = Streaming SIMD (Single instruction, multiple data) extensions

# Outline

- Introduction
- Implementation
- Results
- Summary

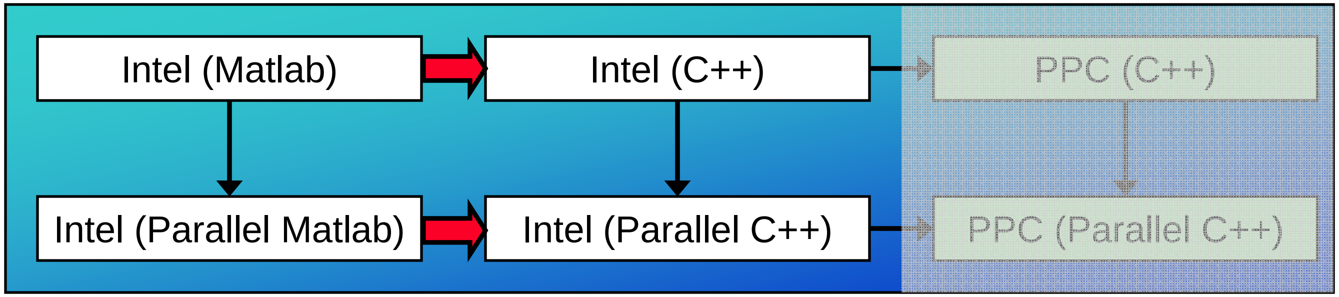
# Development Methodology




Goal: develop real-time stream for Torpedo Adaptive Signal Processor on target platform

- **Serial processing development:**
  - **Finalize Matlab signal processing stream**
  - **Convert Matlab code to serial C/C++ code**
  - **Port serial C/C++ implementation to target PowerPC (PPC) platform**
- **Parallel processing development**
  - **Parallelize Matlab code using pMatlab to determine optimal mappings**
  - **Convert serial C/C++ code to parallel C/C++ code (using pMatlab maps)**
  - **Port parallel C/C++ code to target PPC platform**

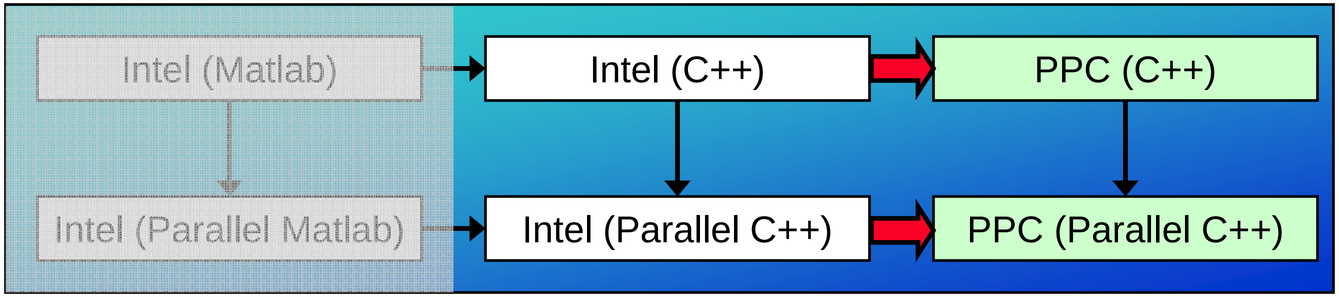
# Development Methodology: Matlab to C++




- **Translate Matlab functionality into PVL/C++**
  - **Cell arrays**
  - **Strided submatrices**
  - **Various mathematical functions (e.g., specgram)**
- **Develop efficient implementations for Matlab memory routines (e.g., reshaping, transposing)**
- **Pre-allocate memory during program initialization for optimal performance (Matlab allocates throughout processing)**

***With PVL we are currently hand converting one line of Matlab code into an average of five lines of C++ code***

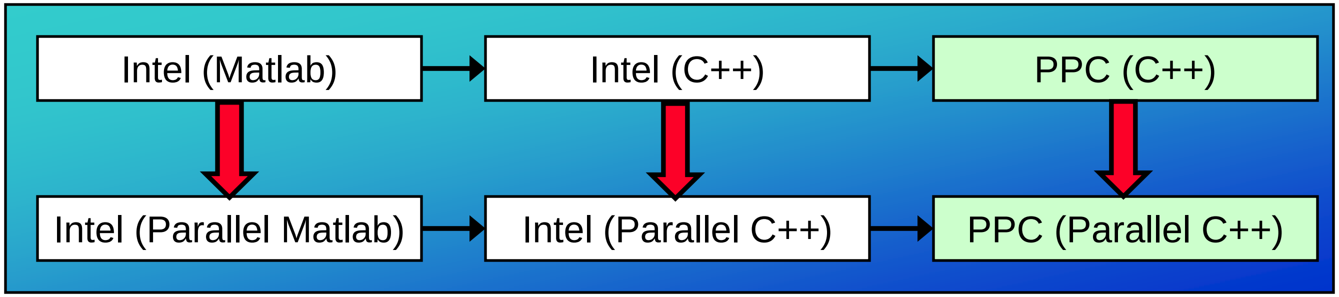
# Development Methodology: Intel to PPC




- **To run intel-based C++ application on target platform, it is necessary to port both application, PVL to Radstone board**

- **Reconcile any compiler issues due to:**
  - **Different header files**
  - **Different or unsupported functions**
  - **Single instruction, multiple data (SIMD) optimizations made on Intel platform that need to be converted to their PPC SIMD equivalents**

# Development Methodology: Serial to Parallel




- **Determine optimal computation distribution mappings using pMatlab (i.e. frequency bands, frequency bins, time, other)**

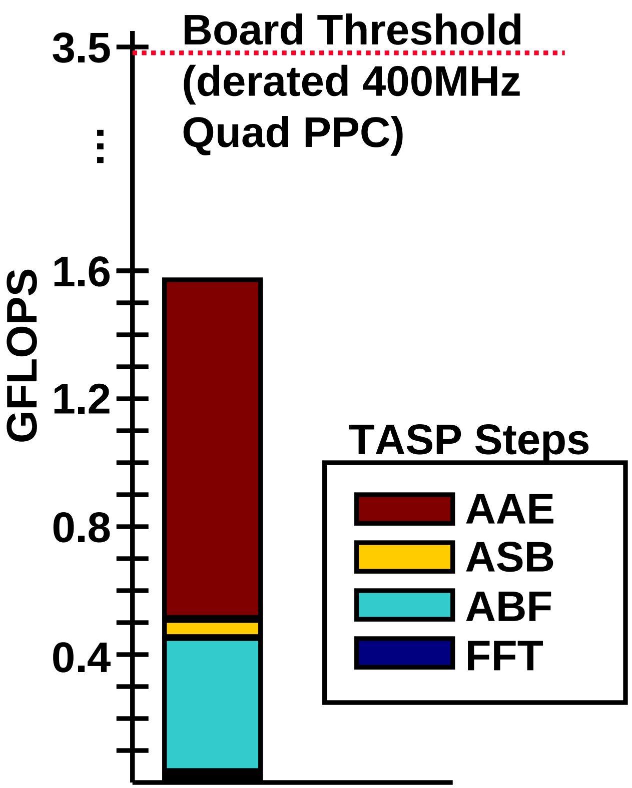
- **Verify that there are no data dependencies among compute nodes**

**Parallel middleware libraries (pMatlab, PVL) automate parallelism**

# Performance Analysis

**Goal: determine how close we are to achieving real-time processing**

**TASP Theoretical Throughput Analysis (Mk 54 Implementation)**




<u>Performance Analysis Steps</u>

Determine complexity of algorithm

↓

Convert workload to theoretical throughput

↓

Evaluate system performance



GFLOPS = Billions of floating point operations per second

# Performance Analysis Simulator Features

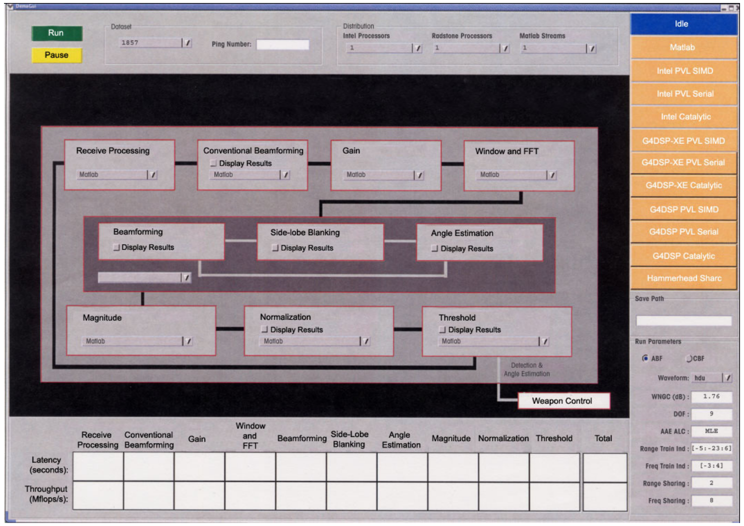
- **PAS is an interactive GUI for algorithm development and analysis**

**PAS Building Blocks**

| Matlab Scripts | Modifiable Complexity Analysis Scripts | Mk 54/48 Merged Baseline | TCP Sockets |
|---|---|---|---|

## Developer Inputs

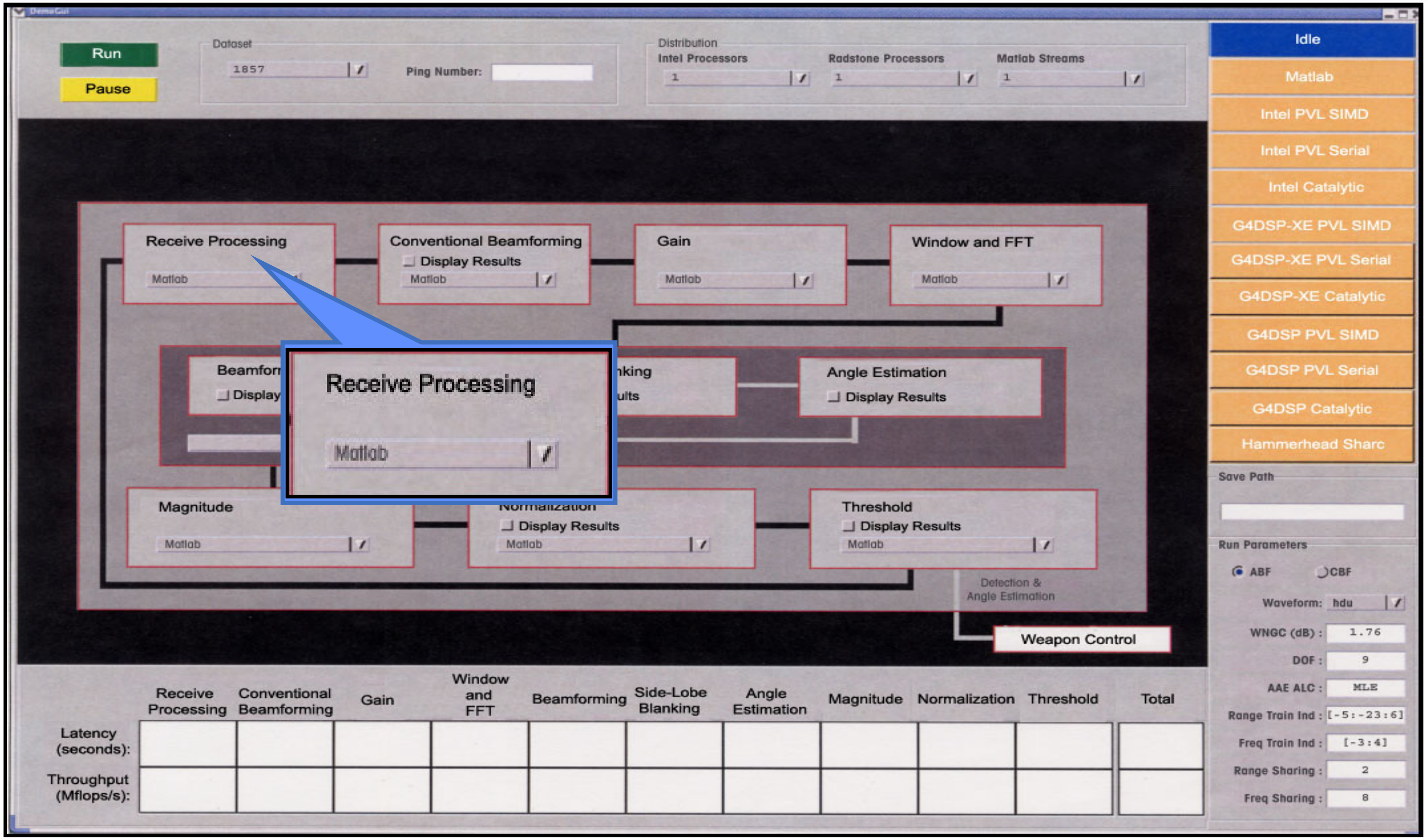
- Software Implementation
- Hardware Platform
- Parameters and Dataset




## GUI Outputs

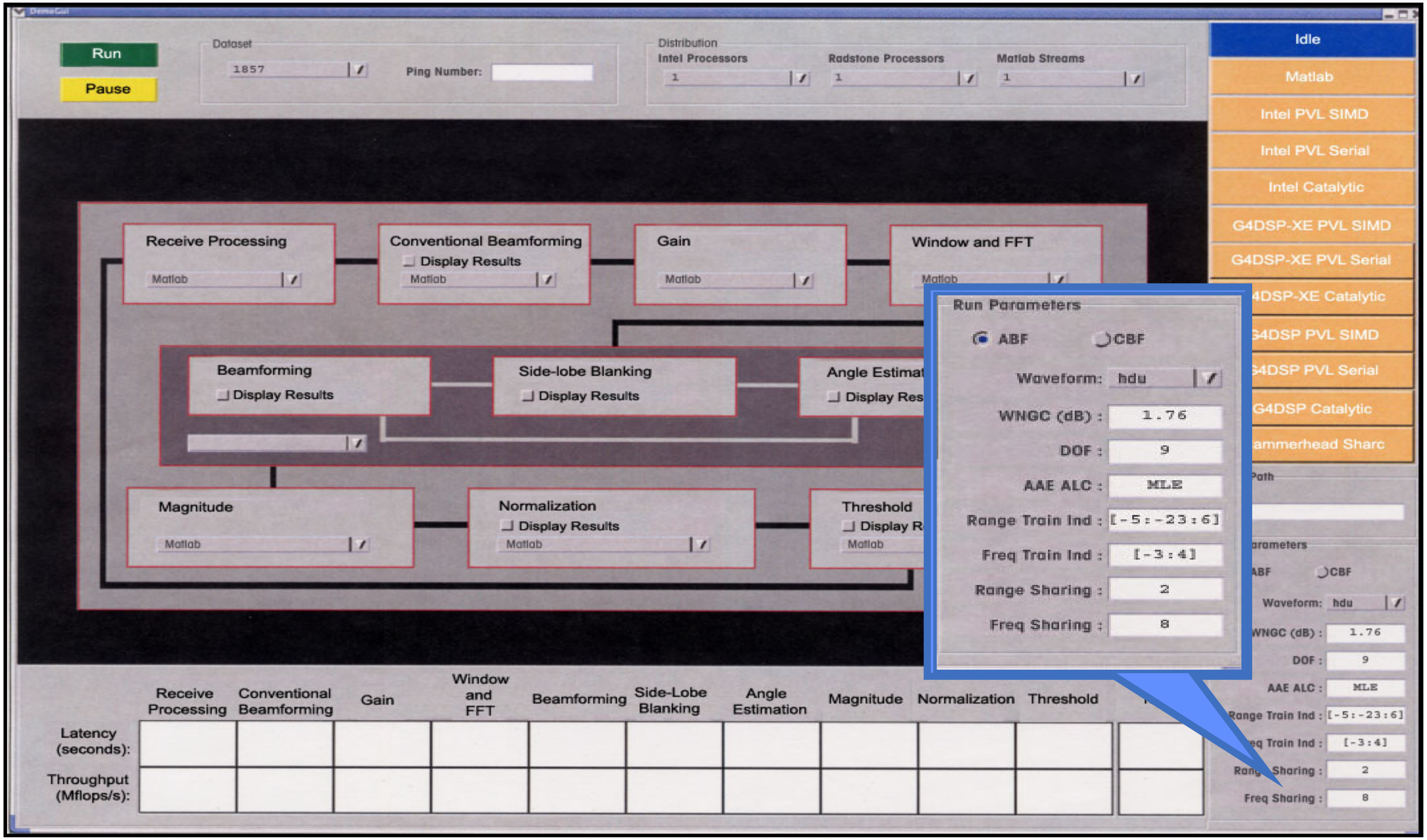
- Processing Results
- Computational Performance Results
- Real-time Achievement Factor

# Torpedo Development Performance Analysis Simulator



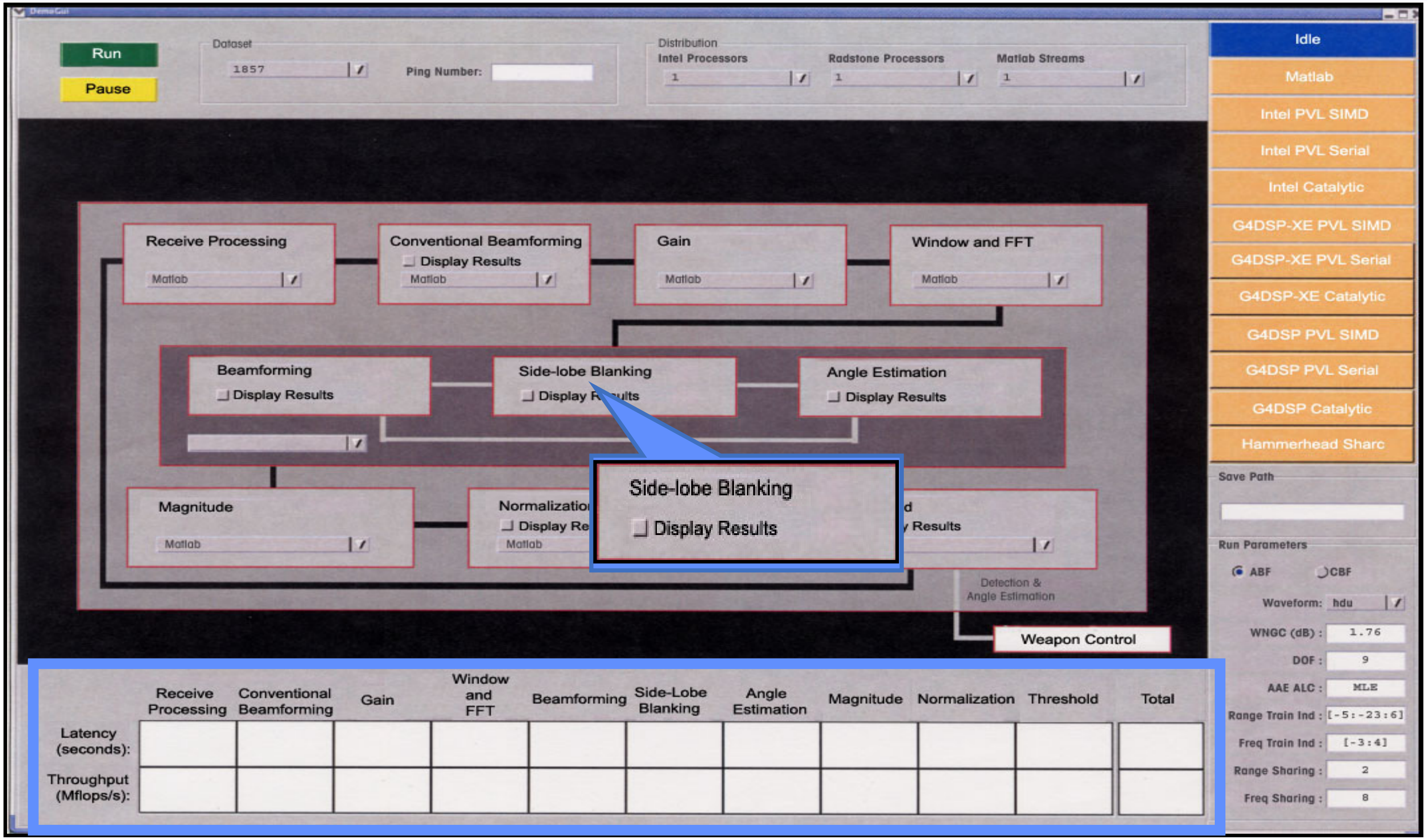

- Algorithm development
- Hardware configuration control
- Algorithm parameter selection
- Data set selection
- "Probe" results
- Latency & throughput results

# Torpedo Development Performance Analysis Simulator




- Algorithm development
- Hardware configuration control
- Algorithm parameter selection
- Data set selection
- "Probe" results
- Latency & throughput results

# Torpedo Development Performance Analysis Simulator




- Algorithm development
- Hardware configuration control ⇨
- Algorithm parameter selection ⇨

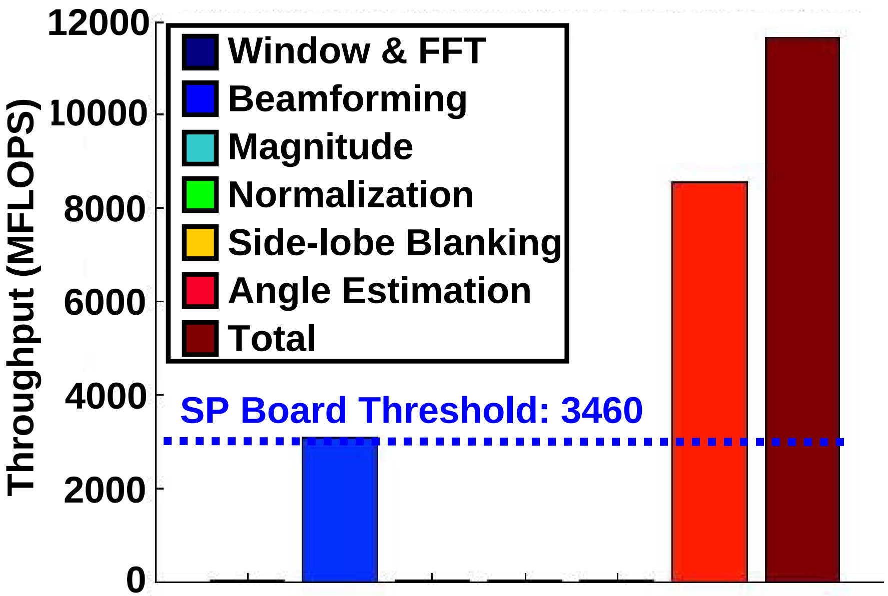
- Data set selection
- “Probe” results
- Latency & throughput results

# Outline

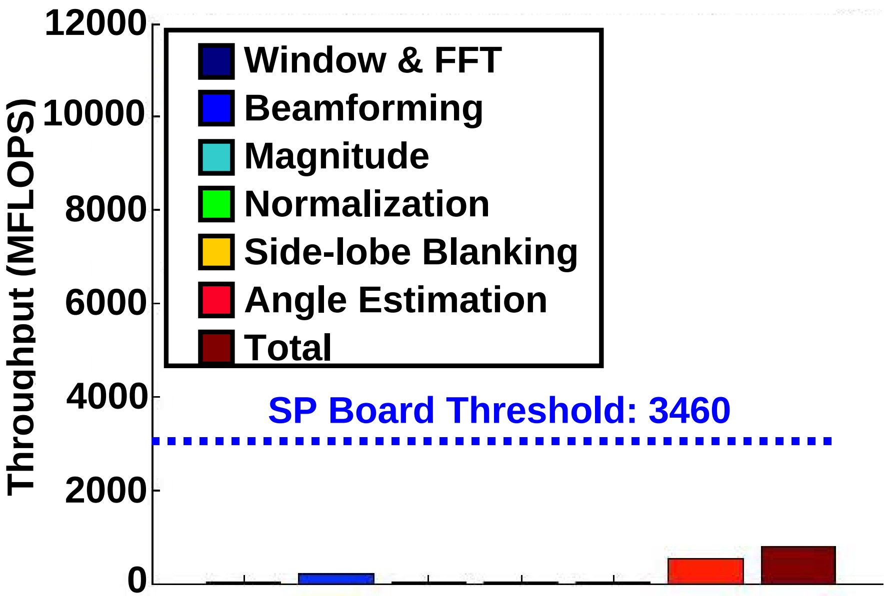
- Introduction
- Implementation
- Results
- Summary

# Complexity Analysis Results

Throughput without weight-sharing




Throughput with Weight-sharing

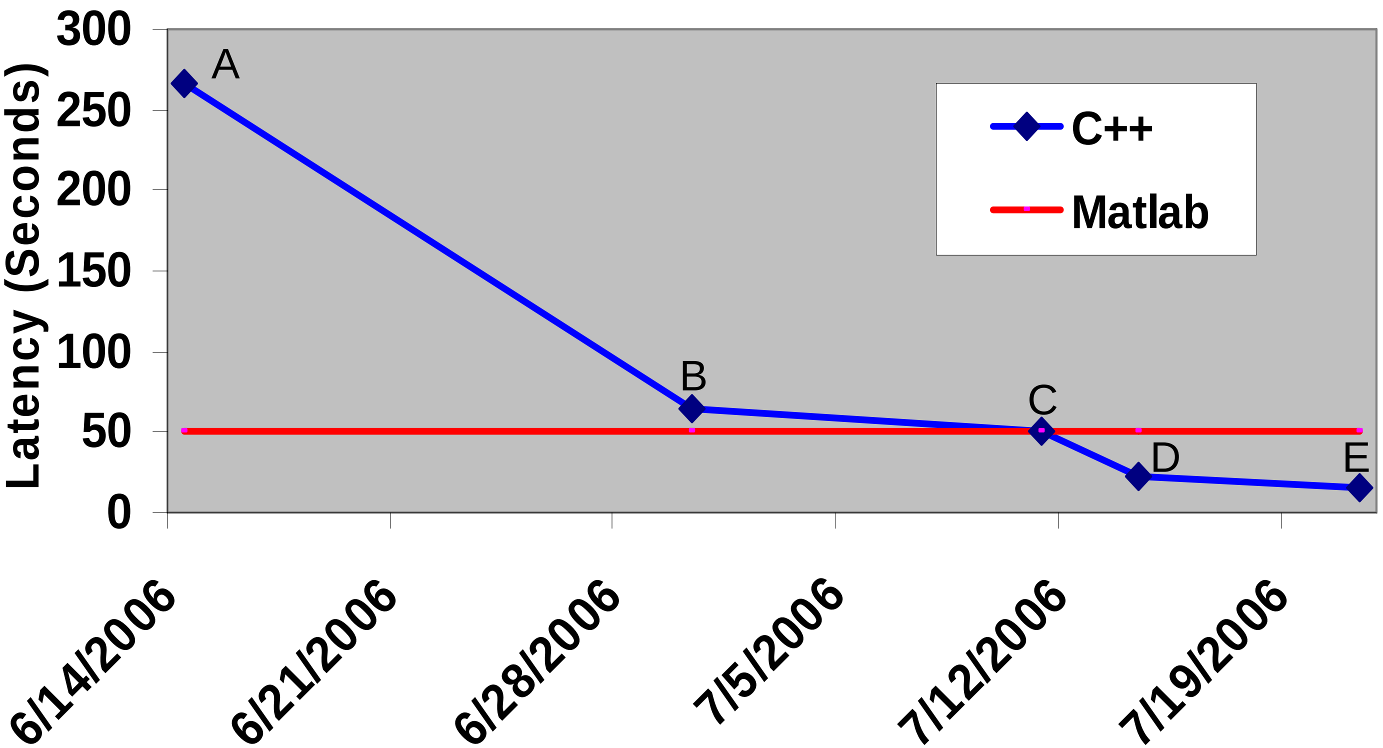



- Computations assume reduced Degrees of Freedom (DOF) ABF with White Noise Gain Constraint (WNGC) loading
- Initial TASP implementation does not fit onto SP board
- Weight sharing allows TASP to fit onto SP board



MFLOPS = Millions of floating point operations per second

# TASP C++ Hand Optimization Results




- A: Verbatim Matlab to C++ translation
- B: Padded data to allow for SIMD optimizations
- C: Developed efficient methods for memory routines (i.e. find(), reshape() )
- D: Used pointers to access & modify non-contiguous data
- E: Preprocessed constant variables (i.e. beamformer coefficients)



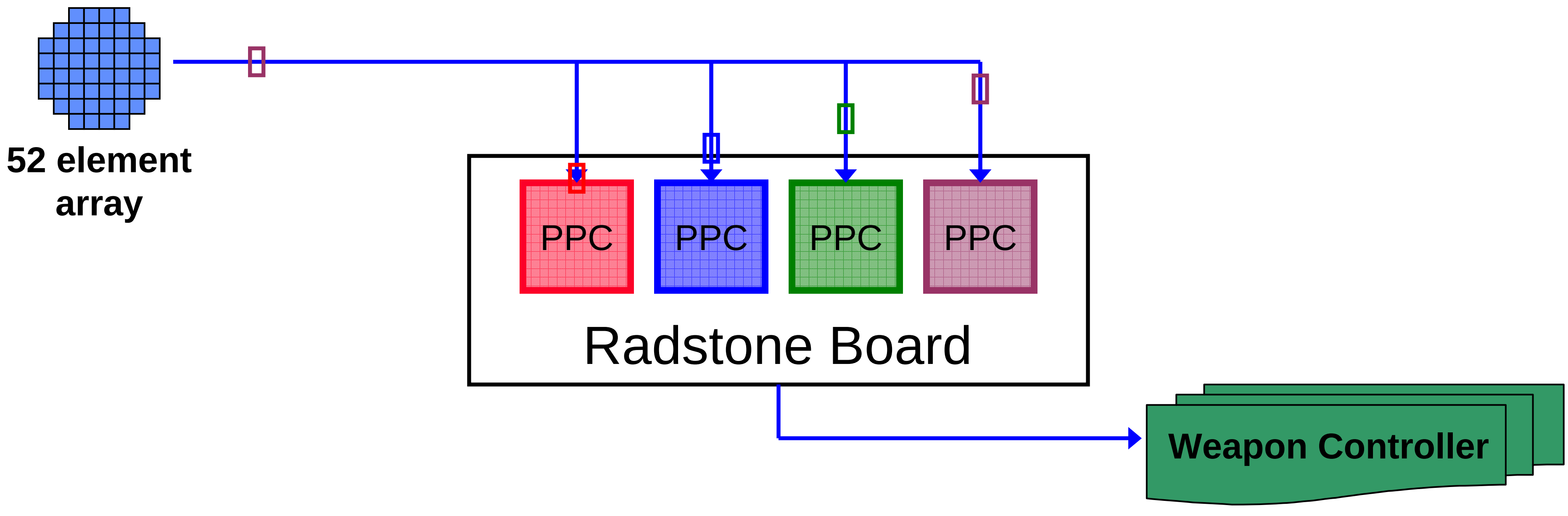


**TASP performed on a single Intel Xeon processor**

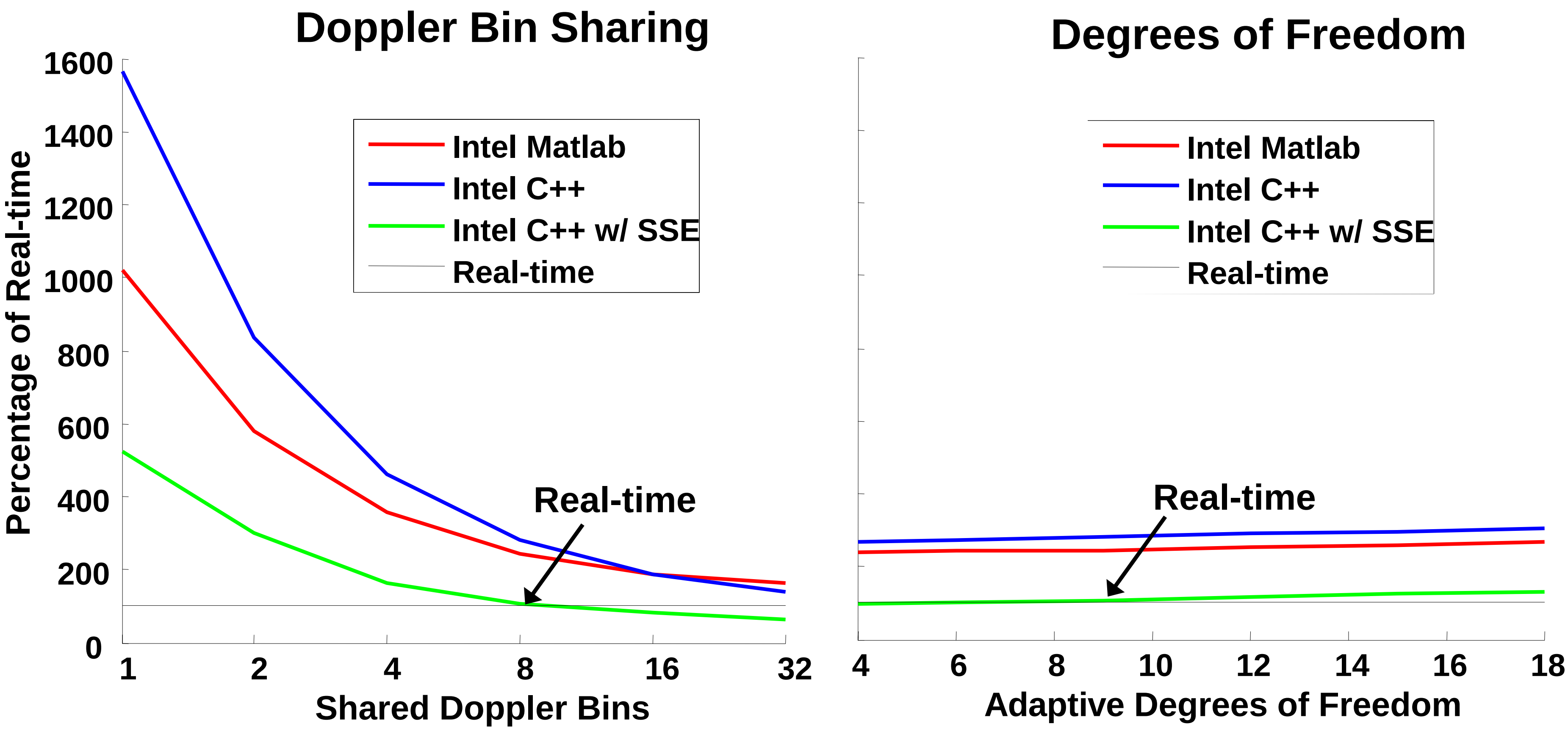
# Parallel Processing Distribution

## Distribution along time dimension

•Processors cycle around and get data from every 4th range gate



# Computational Benchmark Results




- Real-time achieved via Intel C++ with SSE using 8 shared Doppler bins and 9 degrees of freedom
- Results reflect parallel processing

# Outline

- Introduction
- Implementation
- Results
- Summary

# Summary

- **Formulated strategy for achieving real-time processing requirement**
  - **Methodology is being applied to development of Torpedo Adaptive Signal Processor (TASP)**
- **Performed complexity analysis to determine required throughput in order to achieve real-time processing**
- **Developed real-time processing stream in C/C++ using portable/scalable middleware library**
- **Created Performance Analysis Simulator interactive GUI to simplify signal processing development, computation assessment**

# MIT-LL Future Efforts in Torpedo Development Process

- Continue to optimize MIT-LL implementation of torpedo adaptive signal processor (TASP)

- Help torpedo system engineers to integrate MIT-LL implementation of TASP into actual torpedoes
  - Apply lessons learned for optimizing C++ version of TASP
  - Ensure proper communication between torpedo signal processor (adaptive version) and weapon controller

- Support real-time implementation of future algorithm variations of TASP

# BACKUPS

# Platform and Throughput Details

- Signal Processor for ADCAP Mk-48 Mod 7:
  – Radstone G4DSP-7410
  – Quad 400 MHz PowerPC card with 1 Gbyte DDR SDRAM
- Computational capability de-rating:
  – Peak throughput per card: 12.8 GFLOPS (3.2 billion floating point operations per second per processor)
  – Throughput after 50% processor/memory utilization and 54% system efficiency*

**3.5 GFLOPS with 512 Mbytes DDR SDRAM** ← Throughput and memory thresholds for signal processing algorithms.

  – I/O utilization is negligible

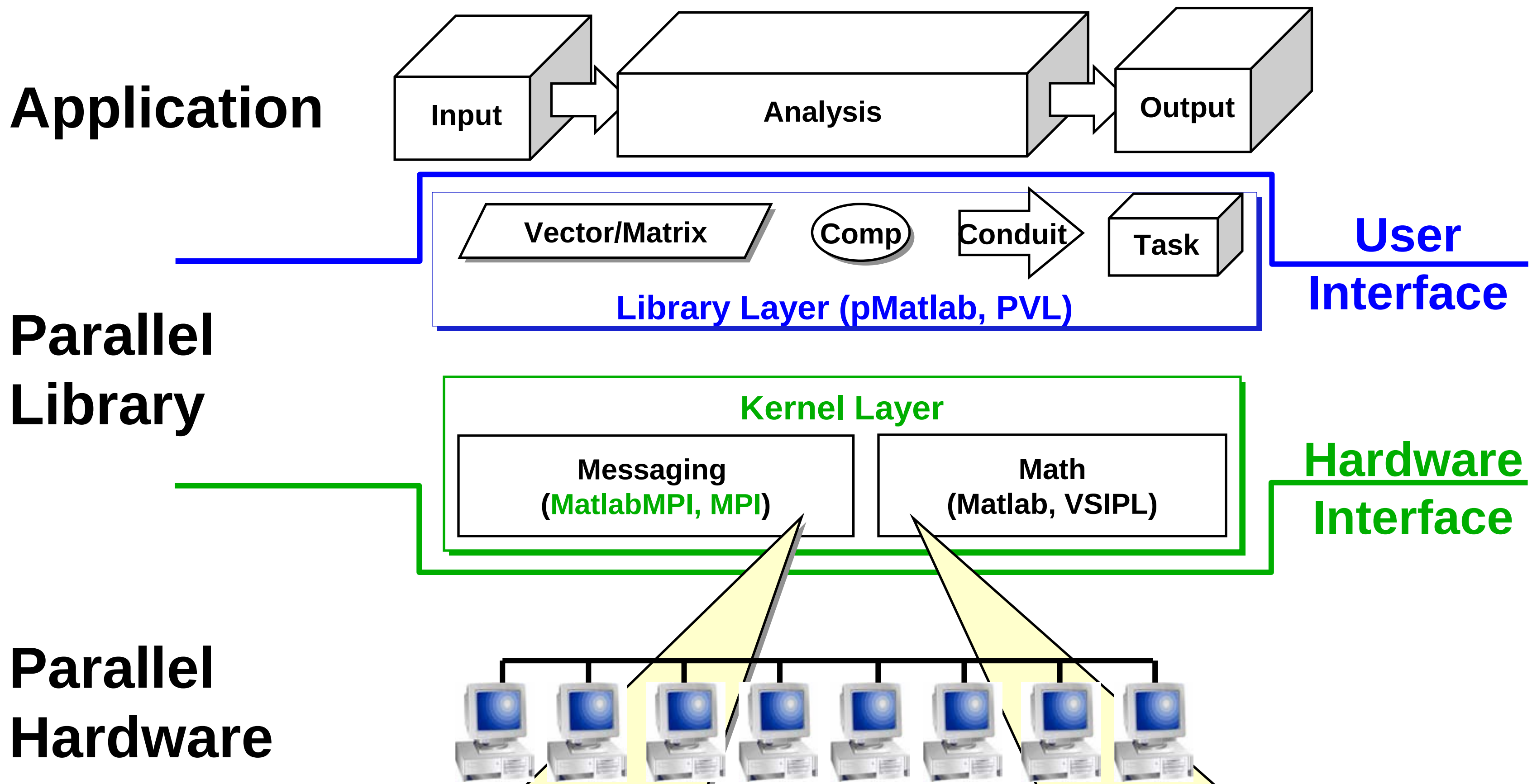
- Algorithm Throughput Calculations:
  – Workload is calculated per batch
  – Throughput = Workload x Range Gate Sampling Rate

GFLOPS = Billions of floating point operations per second



* maximum efficiency taking into account system processes

# Parallel Programming Libraries



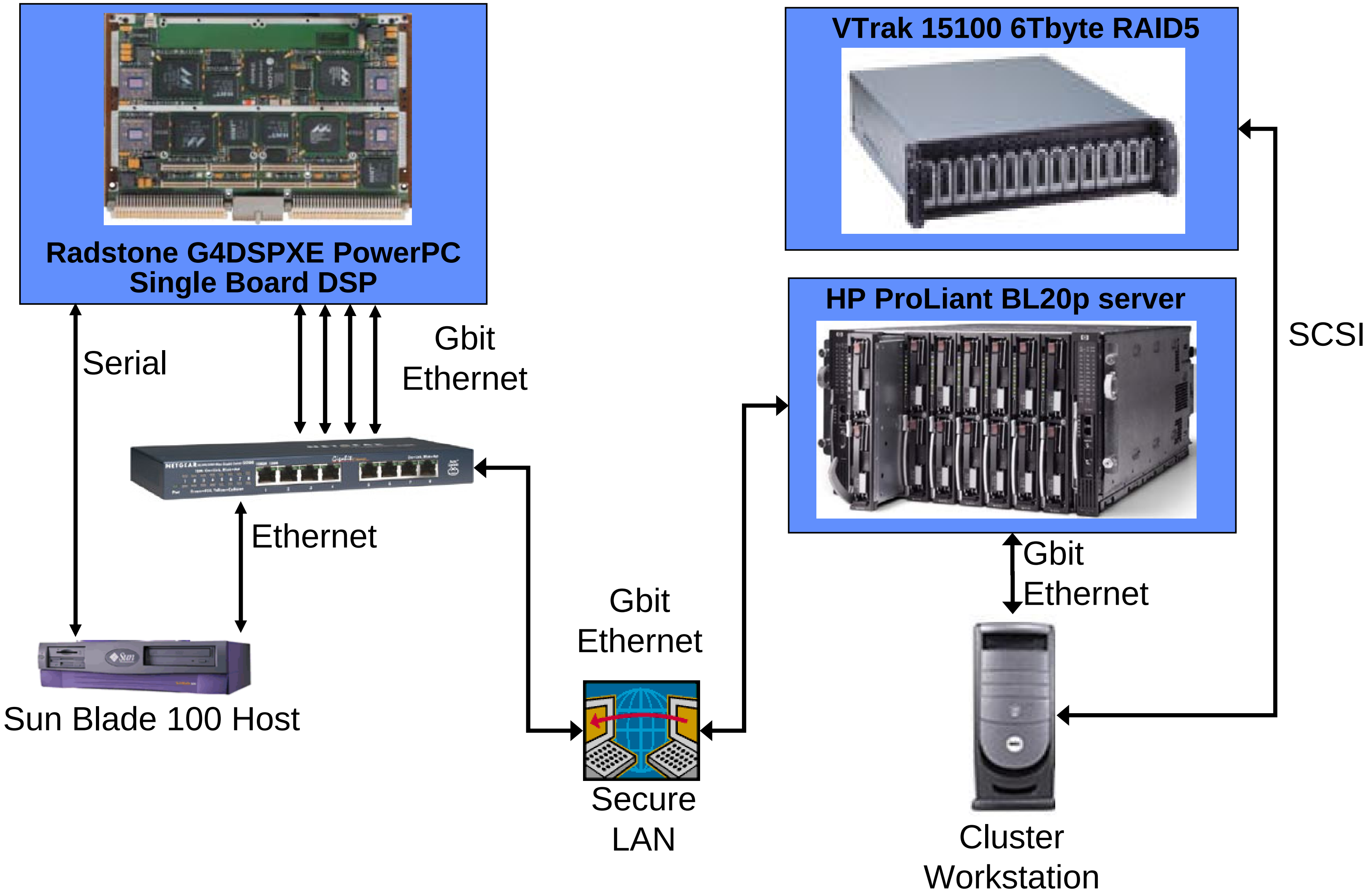

- Can build a parallel library with a few messaging primitives

- Can build a application with a few parallel structures and functions



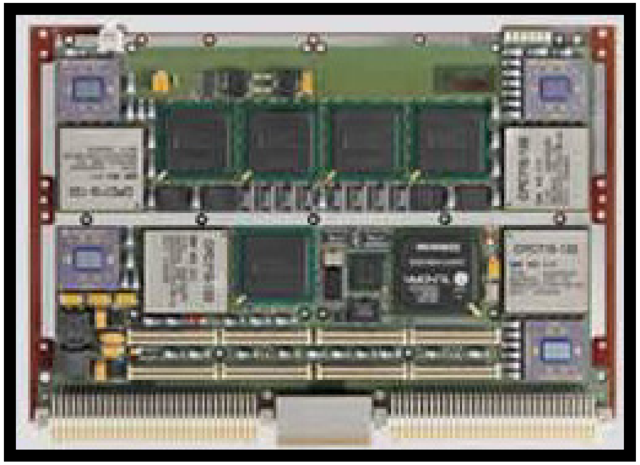
pMatlab = Parallel Matlab PVL = Parallel Vector Library

# MIT/LL TAPB Laboratory Setup



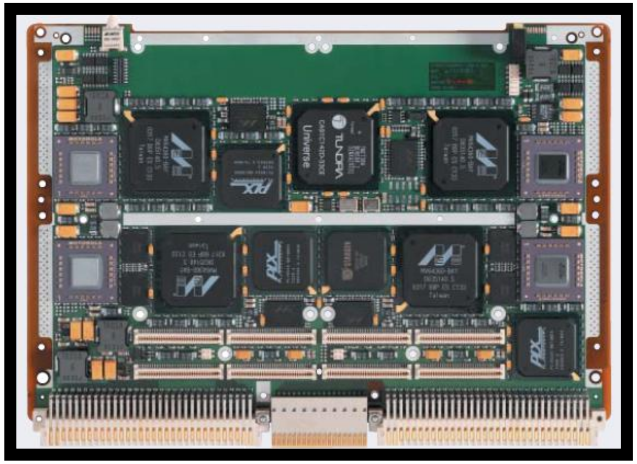
# Signal Processor Prototyping Hardware

## Radstone G4DSP



- Four MPC7410 PowerPC nodes running at 400Mhz
  - Peak throughput: 12.8 GFLOPS (derated to 3.5 GFLOPS)
- 256 Mbytes DDR SDRAM per node
- One 100Base-T ethernet port
  - Peak transfer rate: 12.5 Mbytes/sec
- Consumes <55 Watts
- VxWorks Operating System

## Radstone G4DSP-XE



- Four MPC7447A PowerPC nodes running at 1Ghz
  - Peak throughput: 32 GFLOPS (derated to 8.6 GFLOPS)
- 256 Mbytes DDR SDRAM per node
- Four 1000Base-T ethernet ports
  - Peak transfer rate: 500 Mbytes/sec
- Consumes <55 Watts
- VxWorks Operating System