

*Qualifying Examination*

**A Framework for Real-Time  
High-Throughput Signal and Image  
Processing Systems on Workstations**

**Gregory E. Allen**

**Department of Electrical and Computer Engineering  
The University of Texas at Austin**



**Prof. Craig Chase, chairman  
Prof. Brian Evans, advisor  
Prof. James Browne  
Prof. Lizy John  
Dr. Charles Loeffler**



<http://www.ece.utexas.edu/~allen/>

# Outline

- **Introduction and Motivation**
- **Background**
- **The Proposed Framework**
- **Case Study #1: 3-D Sonar Beamformer**
- **Case Study #2: SAR Processor**
- **Summary**
- **Future Work and Schedule**

# Introduction

- High-performance, low-volume applications (~100 MB/s I/O; 1-20 GFLOPS; under 50 units)
  - Sonar beamforming
  - Synthetic aperture radar (SAR) image processing
  - Multichannel image restoration at video rates
- Current real-time implementation technologies
  - Custom hardware
  - Custom integration using commercial-off-the-shelf (COTS) processors (e.g. 100 digital signal processors in a VME chassis)
- COTS software development is problematic
  - Development and debugging tools are generally immature
  - Partitioning is highly dependent on hardware topology

# Workstation Implementations

- Multiprocessor workstations are **commodity** items
  - Up to 64 processors for Sun Enterprise servers
  - Up to 14 processors for Compaq AlphaServer ES
- Symmetric multiprocessing (**SMP**) operating systems
  - Dynamically load balances many tasks on multiple processors
  - Lightweight threads (e.g. **POSIX Pthreads**)
  - Fixed-priority real-time scheduling
- Leverage native signal processing (**NSP**) kernels
- Software development is faster and easier
  - Development environment and target architecture are same
  - Development can be performed on less powerful workstations

# Parallel Programming

- **Problem: Parallel programming is difficult**
  - Hard to predict deadlock
  - Non-determinate execution
  - Difficult to make scalable software (e.g. rendezvous models)
- **Solution: Formal models for programming**
- **We develop a model that leverages SMP hardware**
  - Utilizes the formal bounded Process Network model
  - Extends with firing thresholds from Computation Graphs
  - Models overlapping algorithms on continuous streams of data
- **We provide a high-performance implementation**

# Motivation

	Custom Hardware	Embedded COTS	Commodity Workstation
<b>Development cost</b>	<b>\$2000K</b>	<b>\$500K</b>	<b>\$100K</b>
<b>Development time</b>	<b>24 months</b>	<b>12 months</b>	<b>6 months</b>
<b>Physical size (m<sup>3</sup>)</b>	<b>3.0</b>	<b>3.0</b>	<b>4.0</b>
<b>Reconfigurability</b>	<b>low</b>	<b>medium</b>	<b>high</b>
<b>Software portability</b>	<b>low</b>	<b>medium</b>	<b>high</b>
<b>Hardware upgradability</b>	<b>low</b>	<b>medium</b>	<b>high</b>

4-GFLOP sonar beamformers; volumes of under 50 units; 1999 technology

# **Outline**

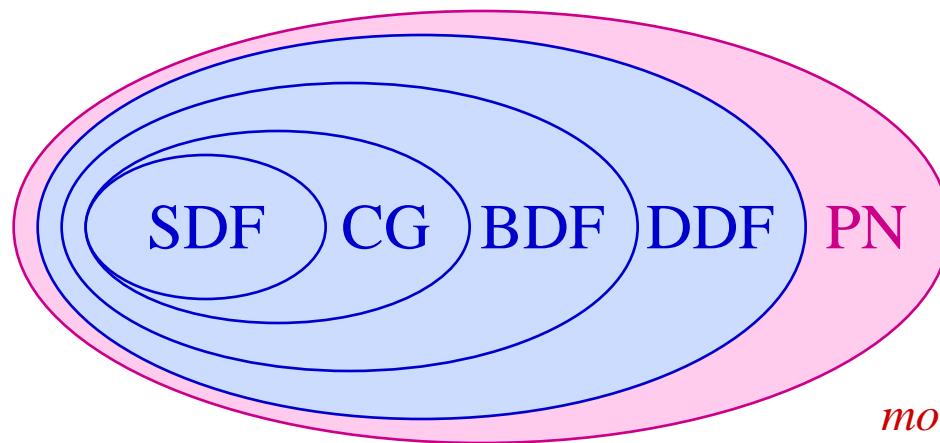
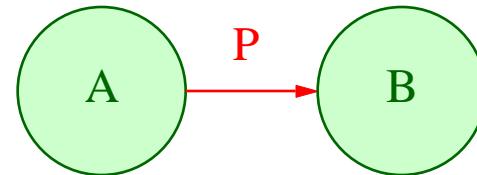
- **Introduction and Motivation**
- **Background**
- **The Proposed Framework**
- **Case Study #1: 3-D Sonar Beamformer**
- **Case Study #2: SAR Processor**
- **Summary**
- **Future Work and Schedule**

# Native Signal Processing

- Single-cycle multiply-accumulate (MAC) operation
  - Vector dot products, digital filters, and correlation  $\sum_{i=1}^N i \chi_i$
  - Missing extended precision accumulation
- Single-instruction multiple-data (SIMD) processing
  - **UltraSPARC Visual Instruction Set (VIS)** and **Pentium MMX**: 64-bit registers, 8-bit and 16-bit fixed-point arithmetic
  - **Pentium III, K6-2 3DNow!**: 64-bit registers, 32-bit floating-point
  - **PowerPC AltiVec**: 128-bit registers, 4x32 bit floating-point MACs
  - Saturation arithmetic (**Pentium MMX**)
- Software data prefetching to prevent pipeline stalls
- Must hand-code using intrinsics and assembly code

# Dataflow Models

- **Models parallelism**
- **A program is represented as a directed graph**
  - **Each node represents a computational unit**
  - **Each edge represents a one-way FIFO queue of data**
- **A node may have any number of input or output edges, and may communicate **only** via these edges**

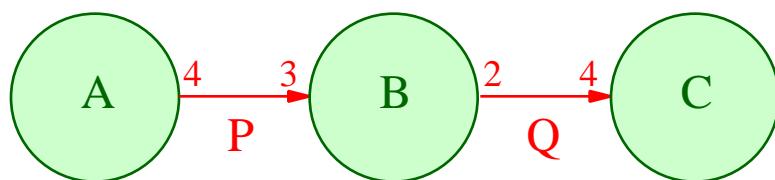


Synchronous Dataflow (SDF)  
Computation Graphs (CG)  
Boolean Dataflow (BDF)  
Dynamic Dataflow (DDF)  
Process Networks (PN)

*more general*

# Synchronous Dataflow (SDF)

- Flow of control and memory usage are known
- Schedule constructed once and repeatedly executed
- Well-suited to synchronous multirate signal processing
- Used in design automation tools (HP EEsof Advanced Design System, Cadence Signal Processing Work System)



Schedule	Memory
AAABBBBBCC	<b>12 + 8</b>
ABABCABCBC	<b>6 + 4</b>

# Computation Graphs (CG)

- **Each FIFO queue is parametrized [Karp & Miller, 1966]**

**A** is number of data words initially present

**U** is number of words inserted by producer on each firing

**W** is number of words removed by consumer on each firing

**T** is number of words in queue before consumer can fire

where **T** = **W**

- **Termination and boundedness are decidable**

- Computation graphs are statically scheduled

- Iterative static scheduling algorithms

- Synchronous Dataflow is  $T = W$  for every queue

# Process Networks (PN)

- A **networked set of Turing machines**
- **Concurrent model (superset of dataflow models)**
- **Mathematically provable properties [Kahn, 1974]**
  - **Guarantees correctness**
  - **Guarantees determinate execution of programs**
  - **Enables easier development and debugging**
- **Dynamic firing rules at each node**
  - **Suspend execution when trying to consume data from an empty queue (blocking reads)**
  - **Never suspended for producing data (non-blocking writes) so queues can grow without bound**

# Bounded Scheduling

- Infinitely large queues cannot be realized
- Dynamic scheduling to always execute the program in bounded memory if it is possible [Parks, 1995]:
  1. Block when attempting to read from an empty queue
  2. Block when attempting to write to a full queue
  3. On artificial deadlock, increase the capacity of the smallest full queue until its producer can fire
- Preserves formal properties: liveness, correctness, and determinate execution
- Maps well to a threaded implementation (one node maps to one thread)

# **Outline**

- **Introduction and Motivation**
- **Background**
- **The Proposed Framework**
- **Case Study #1: 3-D Sonar Beamformer**
- **Case Study #2: SAR Processor**
- **Summary**
- **Future Work and Schedule**

# Contribution #1: Framework

- Utilize the **Process Network** model [Kahn, 1974]
  - Captures concurrency and parallelism
  - Provides correctness and determinate execution
- Utilize **bounded scheduling** [Parks, 1995]
  - Permits realization in finite memory
  - Preserves properties regardless of which scheduler is used
- Extend this model with **firing thresholds**
  - Models overlapping algorithms on continuous streams of data (e.g. digital filters)
  - Decouples computation (node) from communication (queue)
  - Allows compositional parallel programming

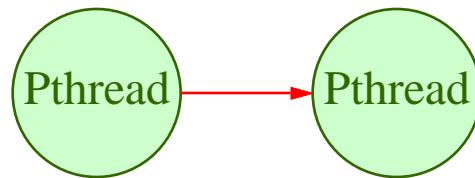
# **Contribution #2: Implementation**

- **Designed for real-time high-throughput signal processing systems based on proposed framework**
- **Implemented in C++ with template data types**
- **POSIX Pthread class library**
  - Portable to many different operating systems
  - Optional fixed-priority real-time scheduling
- **Low-overhead, high-performance, and scalable**
- **Publicly available source code**

<http://www.ece.utexas.edu/~allen/PNSourceCode/>

# Implementation: Nodes

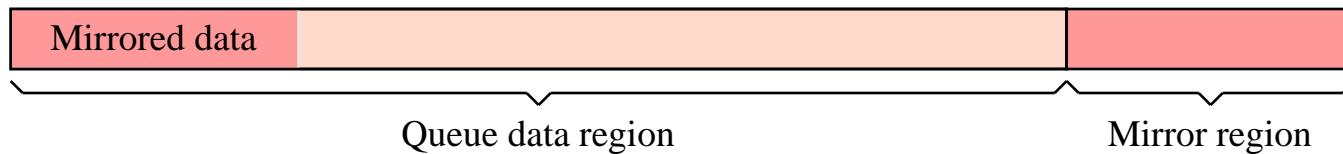
- **Each node corresponds to a Pthread**



- **Node granularity larger than thread context switch**
  - Context switch is about 10  $\mu\text{s}$  in Sun Solaris operating system
  - Increasing node granularity reduces overhead
- Thread scheduler **dynamically** schedules nodes as the flow of data permits
- Efficient utilization of multiple processors (**SMP**)

# Implementation: Queues

- Nodes operate directly on queue memory to avoid unnecessary copying
- Queues use mirroring to keep data contiguous

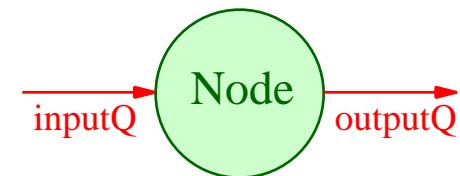


- Compensates for lack of hardware support for circular buffers (e.g. modulo addressing in DSPs)
- Queues tradeoff memory usage for overhead
- Virtual memory manager keeps data circularity in hardware

# A Sample Node

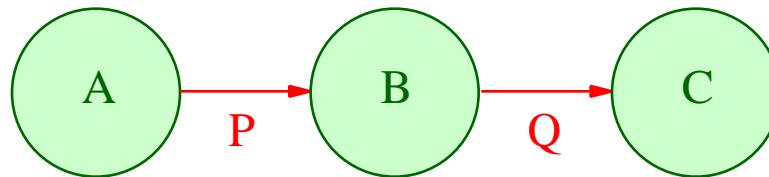
- A queue transaction uses pointers
  - Decouples communication and computation
  - Overlapping streams without copying

```
typedef float T;  
while (true) {  
    // blocking calls to get in/out data pointers  
    const T* inPtr = inputQ.GetDequeuePtr(inThresh);  
    T*       outPtr = outputQ.GetEnqueuePtr(outThresh);  
  
    DoComputation( inPtr, inThresh, outPtr, outSize );  
  
    // complete node transactions  
    inputQ.Dequeue(inSize);  
    outputQ.Enqueue(outSize);  
}
```



# A Sample Program

- Programs currently constructed in C++
- Could be built with a graphical user interface
  - Compose system from a library of nodes
  - Rapid development of real-time parallel software

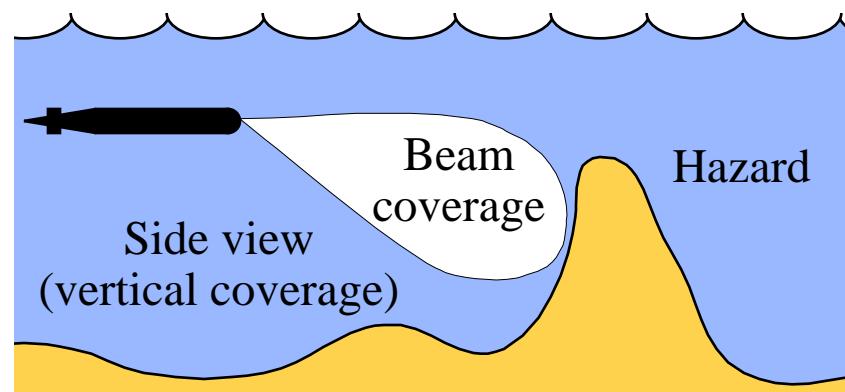
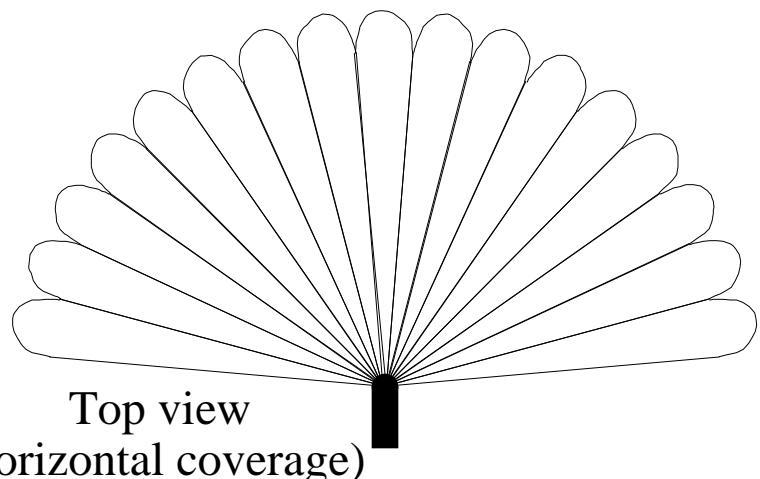
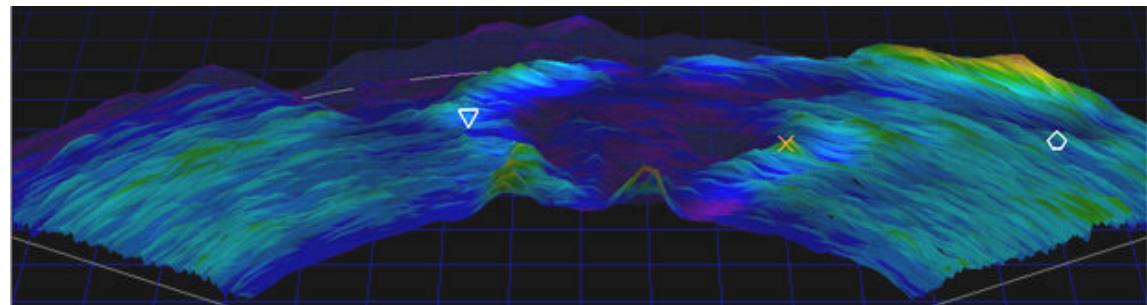


```
int main() {
    PNThresholdQueue<T> P (queueLen, maxThresh);
    PNThresholdQueue<T> Q (queueLen, maxThresh);
    MyProducerNode      A (P);
    MyTransmuterNode    B (P, Q);
    MyConsumerNode      C (Q);
}
```

# Case Study #1: Sonar Beamformer

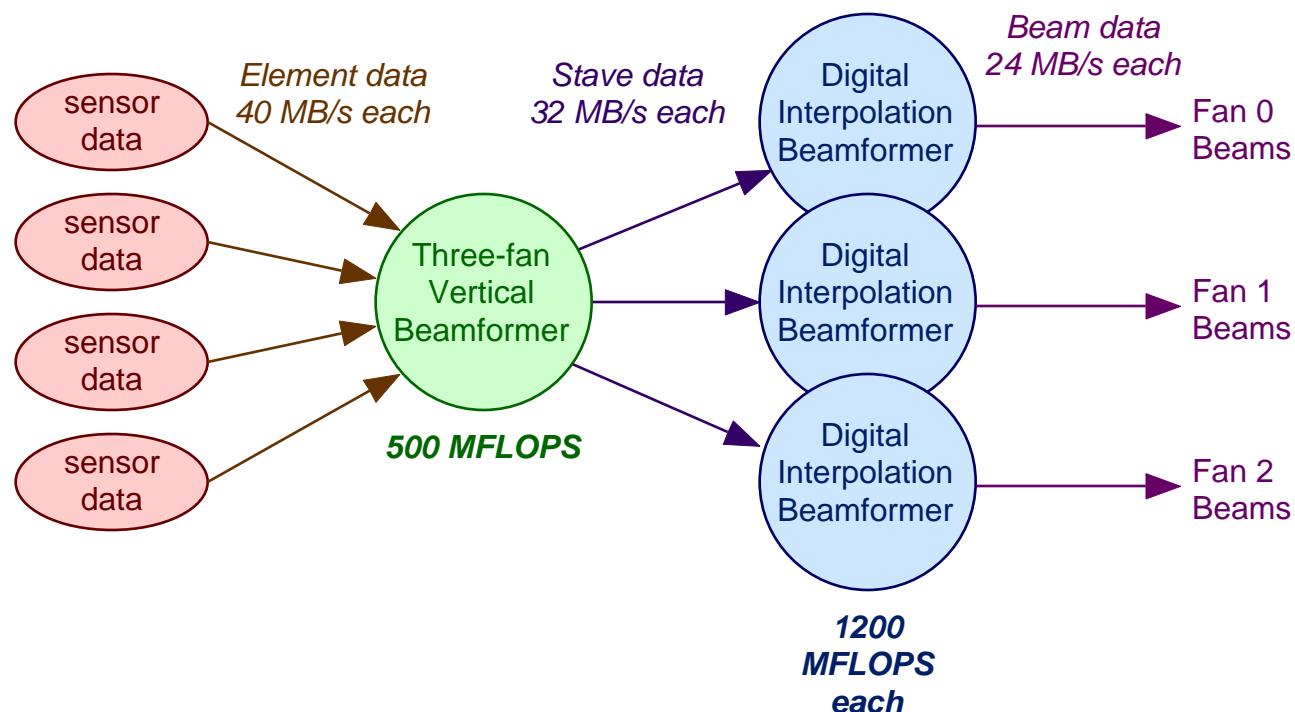


- Collaboration with UT Applied Research Laboratories

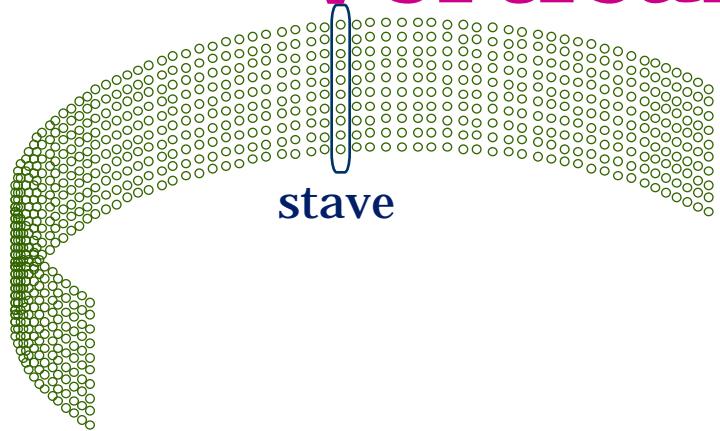


# 4-GFLOP 3-D Beamformer

- 80 horizontal x 10 vertical sensors, data at 160 MB/s
- Collapse vertical sensors into 3 sets of 80 staves
- Do horizontal beamforming, 3 x 1200 MFLOPS



# Vertical Beamformer

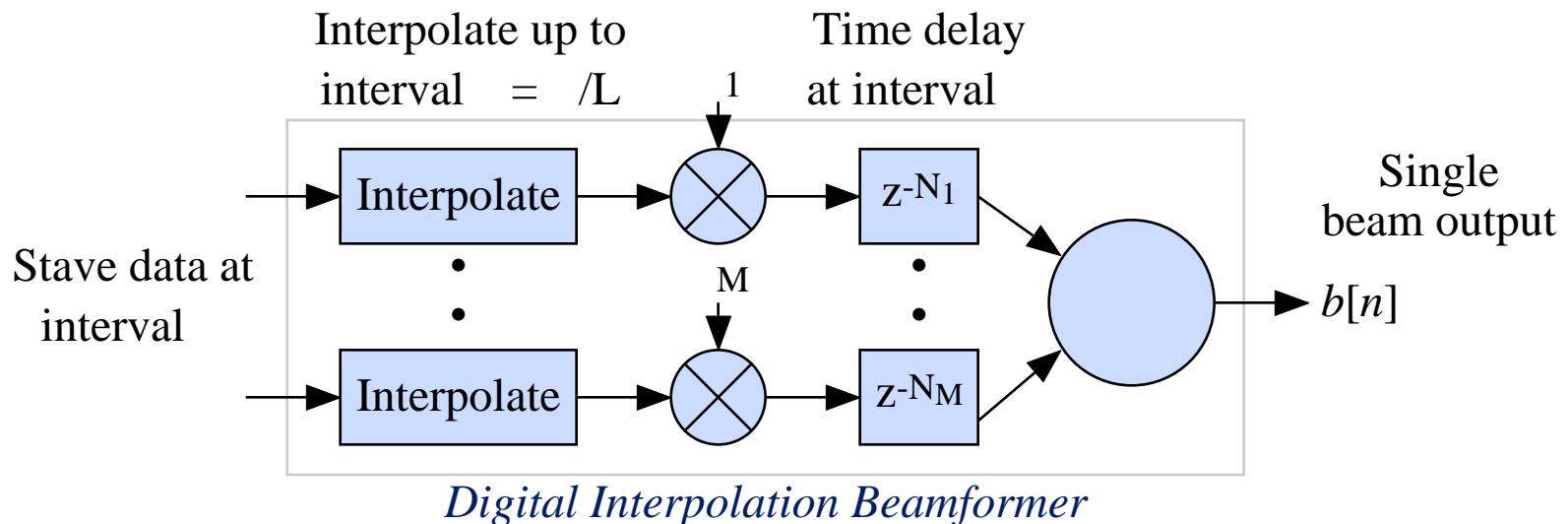


***Multiple vertical transducers  
for every horizontal position***

- Vertical columns combined into 3 **stave** outputs
  - Multiple integer dot products
  - Convert integer to floating-point for following stages
  - Interleave output data for following stages
- Kernel implementation on UltraSPARC-II
  - VIS for fast dot products and floating-point conversion
  - Software data prefetching to hide memory latency
  - Operates at 313 MOPS at 336 MHz (93% of peak)

# Horizontal Beamformer

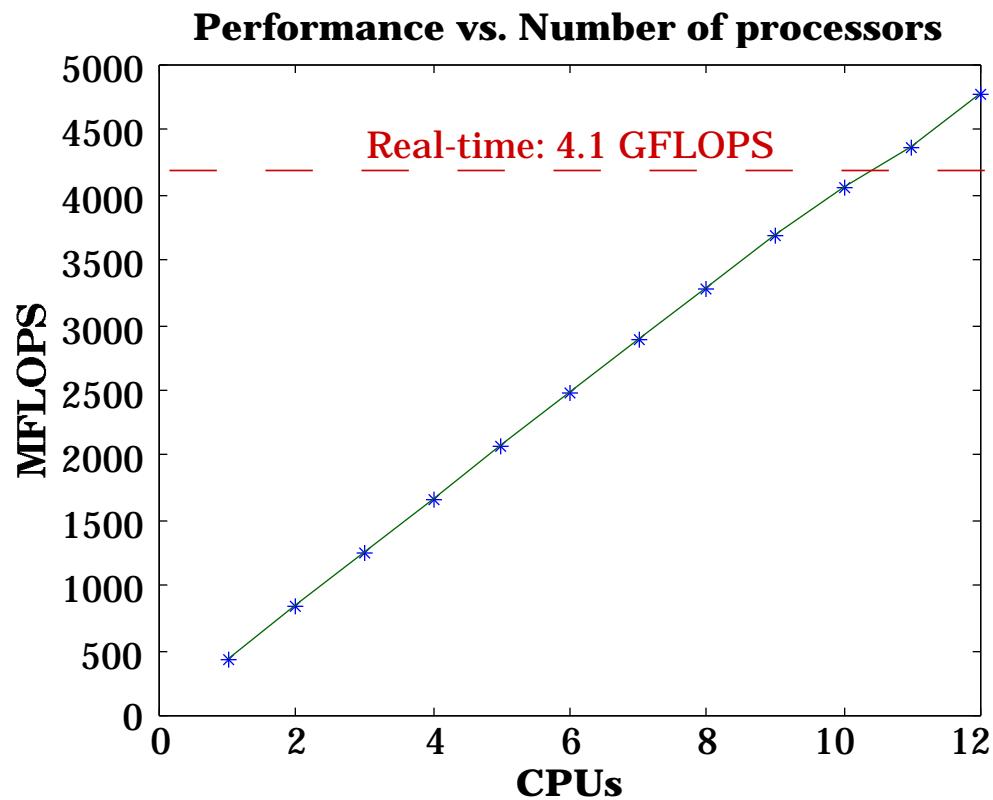
- Sample at just above the Nyquist rate, **interpolate** to obtain desired time delay resolution



- Different beams formed from same data
- Kernel implementation on UltraSPARC-II
  - Highly optimized C++ (loop unrolling and SPARCompiler5.0DR)
  - Operates at 440 MFLOPS at 336 MHz (60% of peak)

# Performance Results

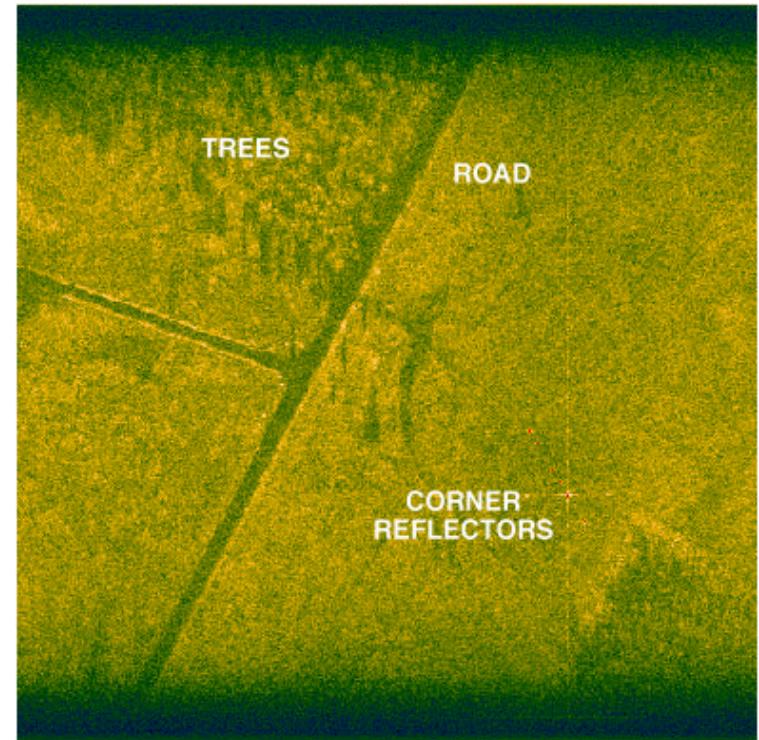
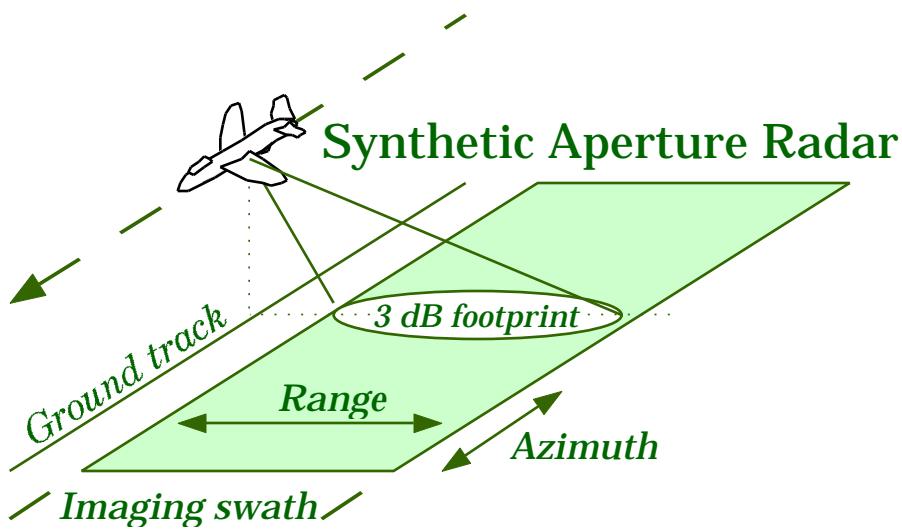
- Sun Ultra Enterprise 4000 with twelve 336-MHz UltraSPARC-IIIs, 3 Gb RAM, running Solaris 2.6
- Compare to sequential case and thread pools



- On one CPU,  
slowdown < 0.5%
- 8 CPUs vs. thread pool
  - 7% faster
  - 20% less memory
- On 12 CPUs
  - Speedup is 11.28 and  
efficiency of 94%
  - Runs real-time +14%

# Case Study #2: SAR Processor

- Combines return signals received by moving radar to improve accuracy
- Deployed from aircraft or spacecraft



- Terrain mapping
- Landscape classification

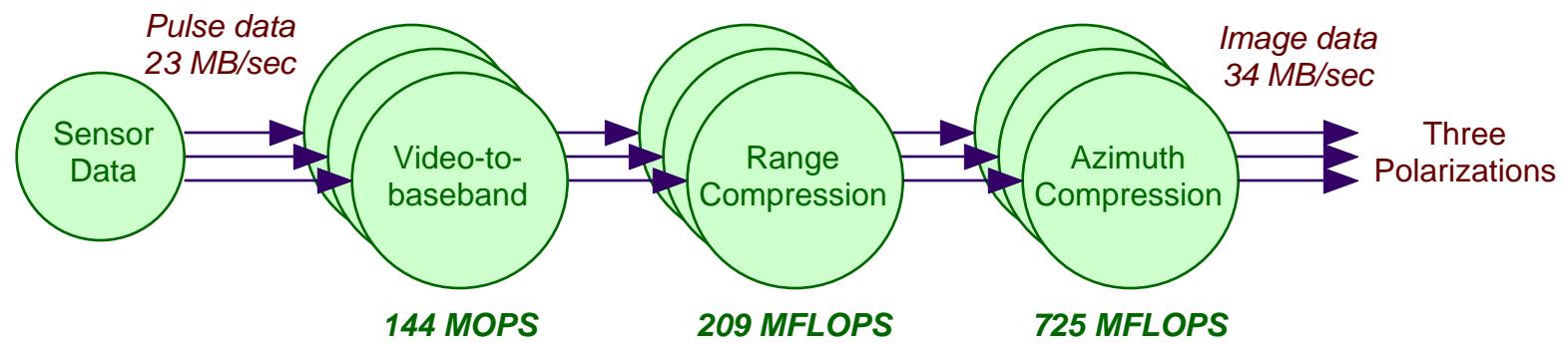
# **Advanced Detection Technology Sensor**

## **MIT Lincoln Laboratory**

- Aircraft-based 35 GHz (Ka) SAR data at 23 MB/s
- Rapid prototyping of Application Specific Signal Processors (RASSP) Benchmark-1
  - 1.1 GOPS real-time system
  - Simulator source code and data sets available for download
  - <http://www.ll.mit.edu/llrassp/>
- Lockheed Martin ATL implementation
  - 1 custom fiber I/O board, 4 COTS boards in a 6U VME rack
  - 68040 controller, 3 quad SHARC DSP boards, RACE Interface

# SAR Processor Block Diagram

- Video-to-baseband (demodulate and FIR filter)
- Range compression (weighting and FFT)
- Azimuth compression (FFT, multiply, and  $\text{FFT}^{-1}$ )
- Three independent channels (polarizations)



- Collaboration with UT Center for Space Research

# Summary

- **Bounded Process Network** model extended with **firing thresholds** from **Computation Graphs**
  - Provides correctness and determinate execution
  - Naturally models parallelism in system
  - Models overlapping algorithms on continuous streams of data
- **Multiprocessor workstation implementation**
  - Designed for high-throughput data streams
  - Native signal processing on general-purpose processors
  - SMP operating systems, real-time lightweight POSIX Pthreads
  - Low-overhead, high-performance and scalable
- **Reduces implementation time and cost**

# Future Work

- Framework enhancements and extensions
  - Multiple identical channels – busses of streams
  - Multiple executions of a program graph
- Formal analysis of framework properties
  - Verify that the use of thresholds preserves PN properties
  - Analyze the consequences and limitations of multiple executions
- Case studies
  - Complete SAR processor case study
  - Multimedia applications
  - Multichannel image restoration
- Integration with rapid prototyping tools

# Schedule

- ***Summer 1999:*** Complete SAR case study
- ***Fall 1999:*** Extend framework with busses of streams, investigate multiple executions; Implement a multimedia case study
- ***Spring 2000:*** Analyze the formal consequences of the use of thresholds, and of multiple executions
- ***Summer, Fall 2000:*** Investigate design automation tools, and begin integration
- ***Spring 2001:*** Case study which utilizes design automation tools
- ***Fall 2001:*** Write and defend dissertation

# Graduate Coursework

Semester	Course	Title	Instructor	Grade	Area
Fall 1993	EE 380L.6	Interfacing to Operating Systems	Lipovski	A	1
Spring 1994	EE 382N.10	Parallel Computer Architecture	Chase	A	2
Fall 1994	EE 381K.3	Digital Filtering and Data Smoothing	Smith	A	(TISE)
Spring 1995	EE 381K.8	Digital Signal Processing	Bovik	A	(TISE)
Fall 1995	EE 380L.5	Engineering Programming Languages	Chase	A	1
Spring 1996	EE 380L	Neural Networks for Pattern Recognition	Ghosh	A	3
Fall 1996	EE 382M.1	Fault Tolerant Computing I	Abraham	A	4
Spring 1997	EE 382C	Embedded Software Systems	Evans	A	3
Fall 1997	EE 397K.1	Conference Course	Evans	A	-
Spring 1998	CS 395T	Real-time Systems	Mok	A	(CS)
Fall 1998	EE 382M	Comp Perf Eval and Benchmarking	John	A	2
Fall 1998	EE 397K.1	Conference Course	Evans	A	-
Spring 1999	CS 392C	Meth and Tech for Parallel Programming	Browne	A	(CS)
<i>Fall 1999</i>	<i>CS 384M</i>	<i>Multimedia Systems</i>	<i>Vin</i>	<i>-</i>	<i>(CS)</i>
<i>Spring 2000</i>	<i>M 365C</i>	<i>Real Analysis</i>	<i>-</i>	<i>CR/NC</i>	<i>(Math)</i>
<i>Fall 2000</i>	<i>EE 381J</i>	<i>Probability and Stochastic Processes</i>	<i>-</i>	<i>-</i>	<i>(TISE)</i>

# Publications

G. E. Allen, B. L. Evans, and D. C. Schanbacher, “Real-Time Sonar Beamforming on a Unix Workstation Using Process Networks and POSIX Pthreads”, *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, Nov. 1-4, 1998, Pacific Grove, CA, invited paper, vol. 2, pp. 1725-1729.

G. E. Allen and B. L. Evans, “Real-Time High-Throughput Sonar Beamforming Kernels Using Native Signal Processing and Memory Latency Hiding Techniques”, *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, Oct. 25-28, 1999, Pacific Grove, CA, submitted.

G. E. Allen and B. L. Evans, “Real-Time Sonar Beamforming on a Unix Workstation Using Process Networks and POSIX Threads”, *IEEE Transactions on Signal Processing*, accepted for publication.

# Extras!

# Time-Domain Beamforming

- Delay-and-sum weighted sensor outputs
- Geometrically project the sensor elements onto a line to compute the time delays

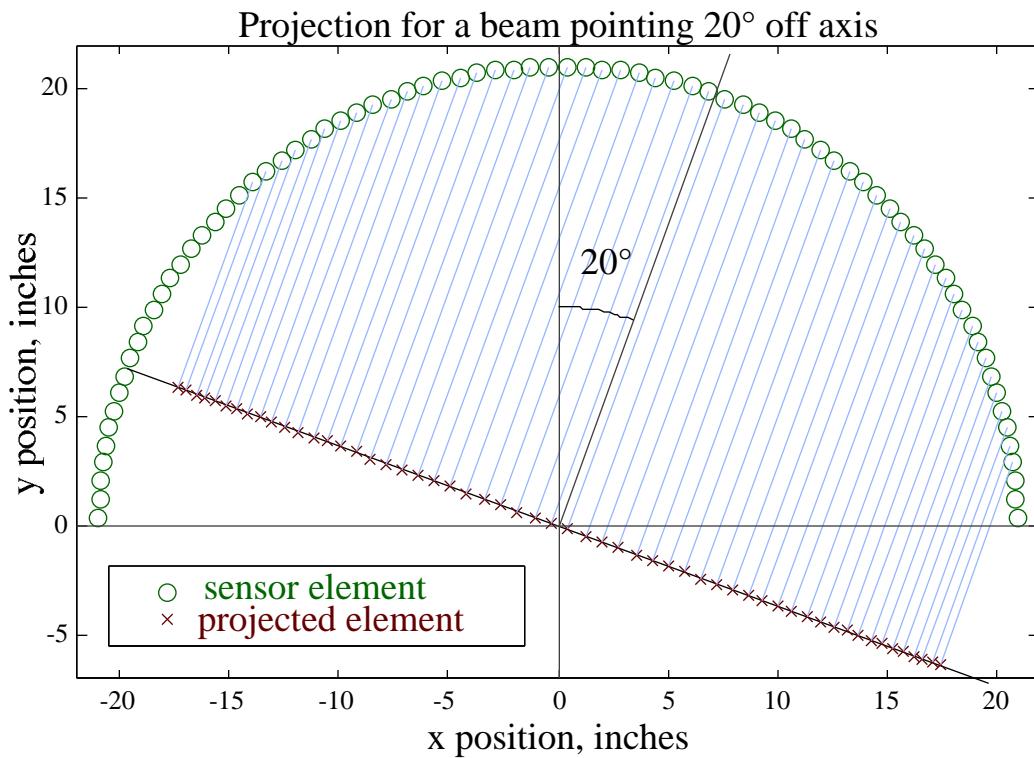
$$b(t) = \sum_{i=1}^M w_i x_i(t - \tau_i)$$

$b(t)$  beam output

$x_i(t)$   $i^{\text{th}}$  sensor output

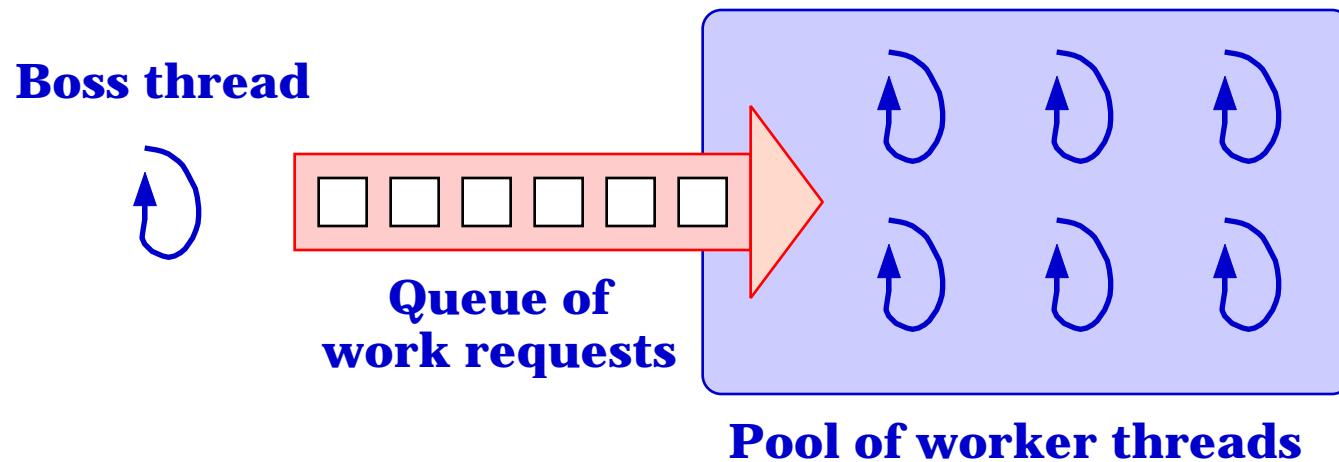
$\tau_i$   $i^{\text{th}}$  sensor delay

$w_i$   $i^{\text{th}}$  sensor weight



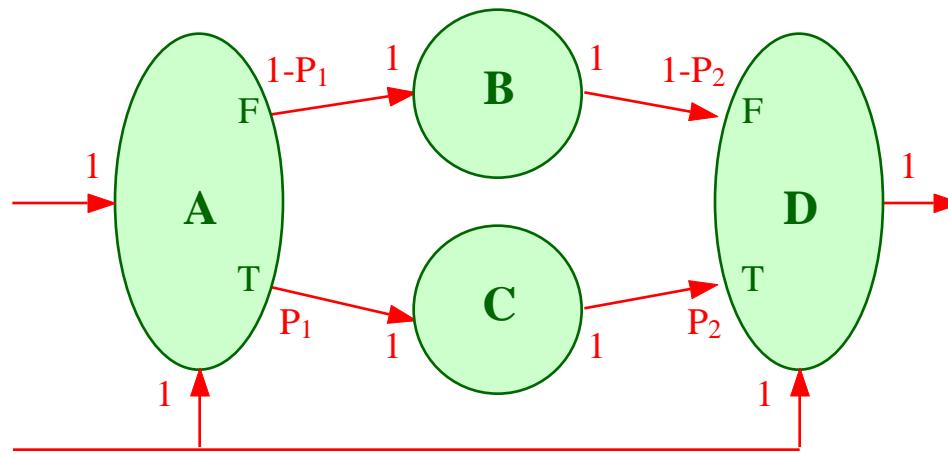
# Thread Pools

- A **boss / worker model for threads**
- A **fixed number of worker threads are created at initialization time**
- The **boss inserts work requests into a queue**
- **Workers remove and process the requests**

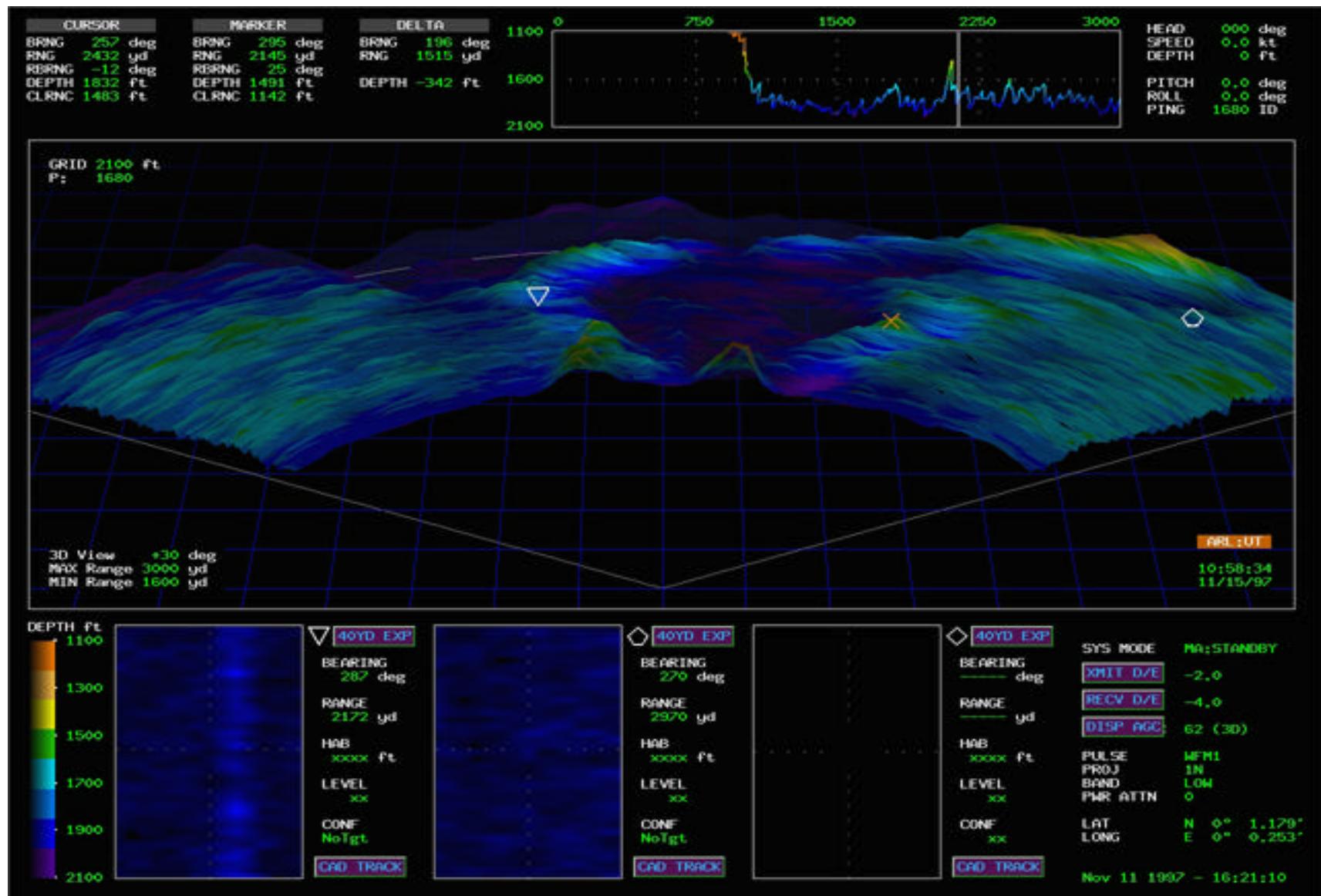


# Boolean Dataflow (BDF)

- Turing complete
- Adds switch and select – provides if/then/else, for loops
- Termination and boundedness are undecidable
- Quasi-static scheduling with clustering of SDF



# Sample Sonar Display



# Process Network Nodes

- Full system: 4 GFLOPS of computation
- A single processor (thread) cannot achieve real-time performance for any one node
- Use a thread pool to allow each node to execute in real-time (data parallelism)
- The number of worker threads is set as processing performance dictates
- For this implementation:
  - The vertical beamformer node uses 4 worker threads
  - Each horizontal beamformer node uses 4 worker threads

# **A Framework for Real-Time High-Throughput Signal and Image Processing Systems on Workstations**

**Prof. Brian L. Evans**

**in collaboration with**

**Gregory E. Allen and K. Clint Slatton**

**Department of Electrical and Computer Engineering  
The University of Texas at Austin**



<http://www.ece.utexas.edu/~bevans/>