Computational Process Networks

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

Gregory E. Allen

EE 382C - Embedded Software Systems

17 February 2000

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

Outline

- Introduction and Motivation
- Modeling Background
- Computational Process Networks
- Application: Sonar Beamforming
- 4-GFLOP 3-D Sonar Beamformer
- Summary

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**
  - **Seismic volume processing**
- **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

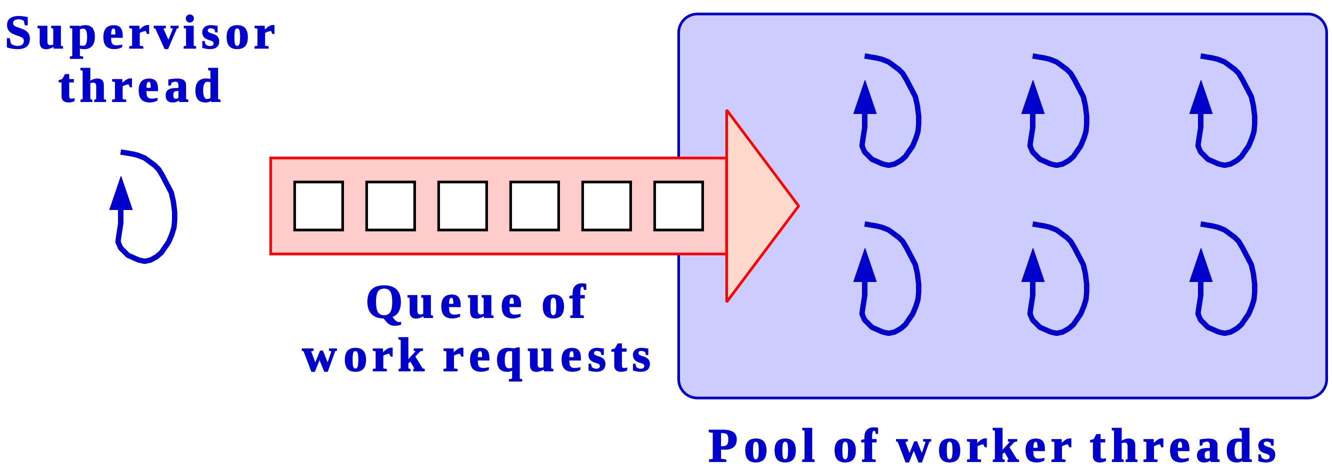
- **Multiprocessor workstations are commodity items**
  - Up to 64 processors for Sun Enterprise servers
  - Up to 14 processors for Compaq AlphaServer ES
- **Symmetric multiprocessing (SMP) operating systems**
  - Dynamically load balances many tasks on multiple processors
  - Lightweight threads (e.g.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

Native Signal Processing

- **Single-cycle multiply-accumulate (MAC) operation**
  - **Vector dot products, digital filters, and correlation** $\sum_{i=1}^{N} \alpha_i x_i$
  - **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**
- **Software data prefetching to prevent pipeline stalls**
- **Must hand-code using intrinsics and assembly code**

Thread Pools

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



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 algorithms on overlapping continuous streams of data**
- **We provide a high-performance implementation**

Motivation

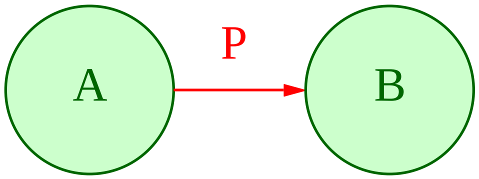
	Custom Hardware	Embedded COTS	Commodity Workstation
Development cost	$2000K	$500K	$100K
Development time	24 months	12 months	6 months
Physical size (m^3)	0.067	0.067	0.090
Reconfigurability	low	medium	high
Software portability	low	medium	high
Hardware upgradability	low	medium	high

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

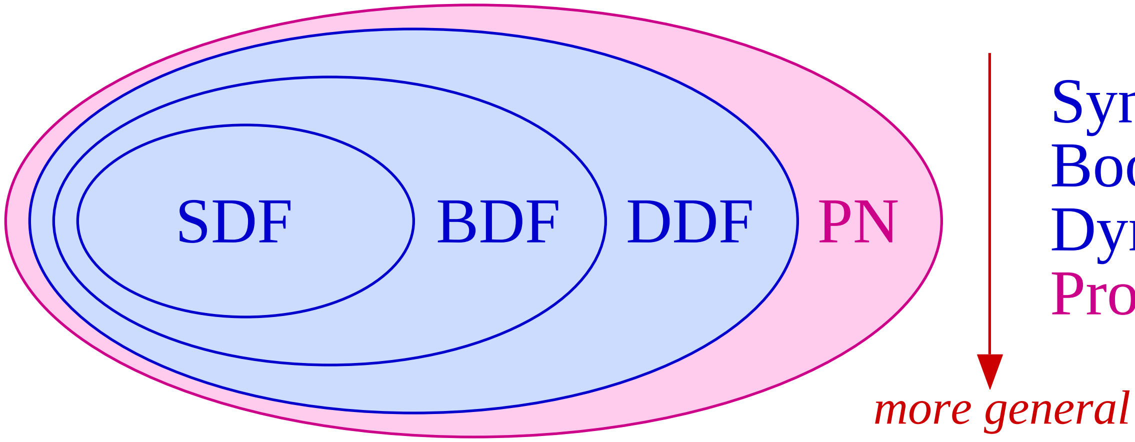
Outline

- Introduction and Motivation
- Modeling Background
- Computational Process Networks
- Application: Sonar Beamforming
- 4-GFLOP 3-D Sonar Beamformer
- Summary

Dataflow Models



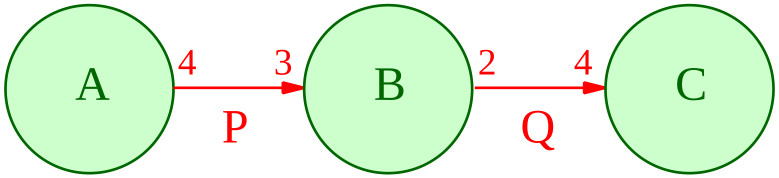
- **Models functional parallelism**
- **A program is represented as a directed graph**
  - **Each node represents a computational unit**
  - **Each edge represents a one-way FIFO queue of data**
- **A node may have any number of input or output edges and may communicate only via these edges**



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

Synchronous Dataflow (SDF)

- **Flow of control and memory usage are known at compile time [Lee, 1986]**
- **Schedule constructed once and repeatedly executed**
- **Well-suited to synchronous multirate signal processing on fixed topologies**
- **Used in design automation tools (HP EEsof Advanced Design System, Cadence Signal Processing Work System)**



Schedule	Memory
AAABBBBCC	12 + 8
ABABCABBC	6 + 4

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**

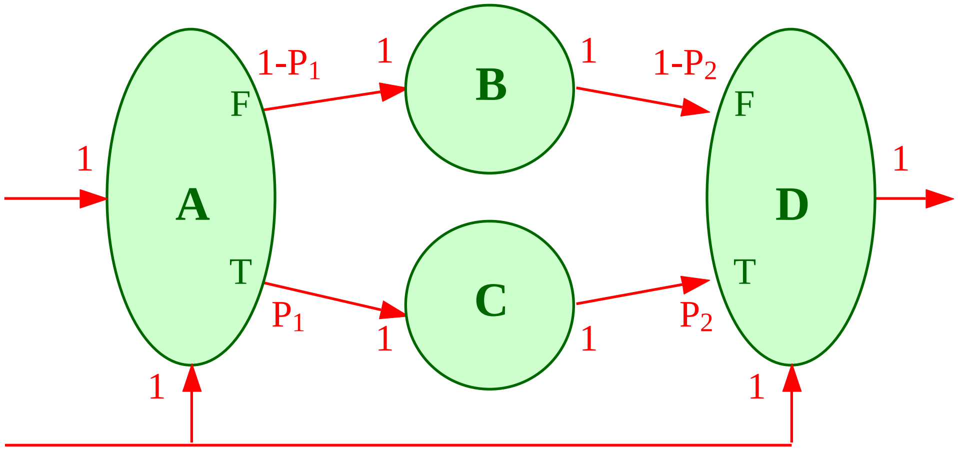
  T **is number of words in queue before consumer can fire**

  where $T \geq W$

- **Termination and boundedness are decidable**
  - **Computation graphs are statically scheduled**
  - **Iterative static scheduling algorithms**
  - **Synchronous Dataflow is** $T = W$ **for every queue**

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



Process Networks (PN)

- A networked set of Turing machines
- Concurrent model for functional parallelism
- Mathematically provable properties [Kahn, 1974]
  - Guarantees correctness
  - Guarantees determinate execution of programs
- 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
- Modeling Background
- Computational Process Networks
- Application: Sonar Beamforming
- 4-GFLOP 3-D Sonar Beamformer
- Summary

Computational Process Networks

- **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 algorithms on overlapping continuous streams of data, e.g. digital filters and fast Fourier transforms (FFTs)
  - Decouples computation (node) from communication (queue)
  - Allows compositional parallel programming

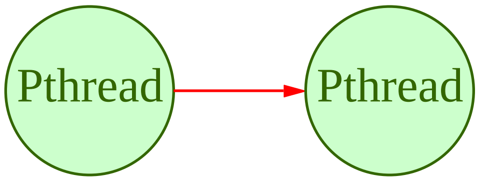
Implementation

- 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

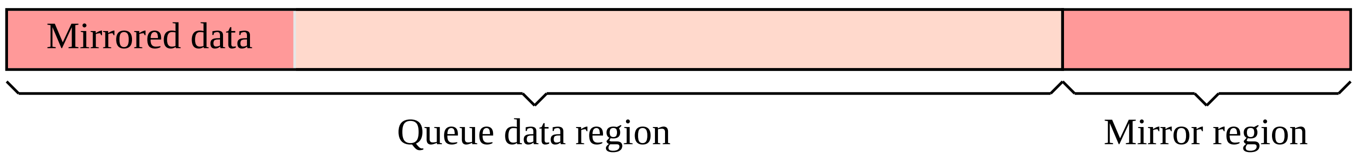
- **Each node corresponds to a Pthread**



- **Node granularity larger than thread context switch**
  - **Context switch is about 10 μs 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

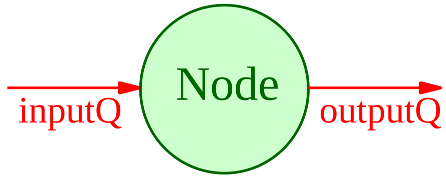
- Queues have input and output firing thresholds
- Nodes operate directly on queue memory to avoid unnecessary copying
- Queues use mirroring to keep data contiguous



- 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**



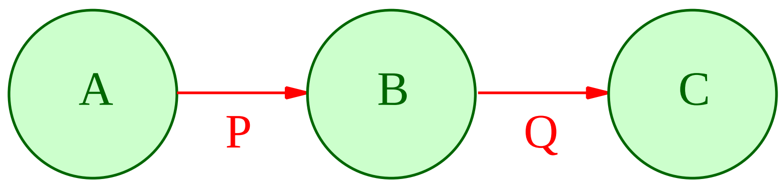

```
typedef float T;
while (true) {
   // blocking calls to get in/out data pointers
   const T* inPtr  = inputQ.GetDequeuePtr(inThresh);
   T*       outPtr = outputQ.GetEnqueuePtr(outThresh);

   DoComputation( inPtr, inThresh, outPtr, outThresh );

   // complete node transactions
   inputQ.Dequeue(inSize);
   outputQ.Enqueue(outSize);
}
```

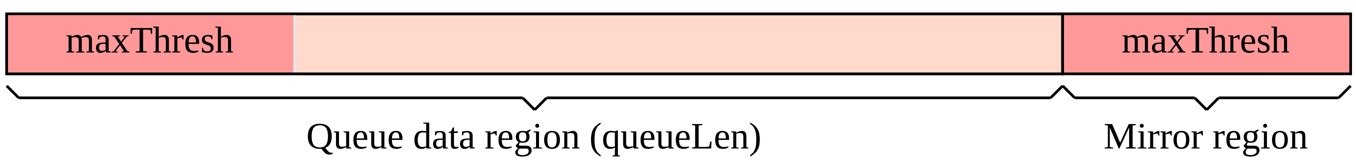
A Sample Program

- **Compose system from a library of nodes**
- **Rapid development of real-time parallel software**

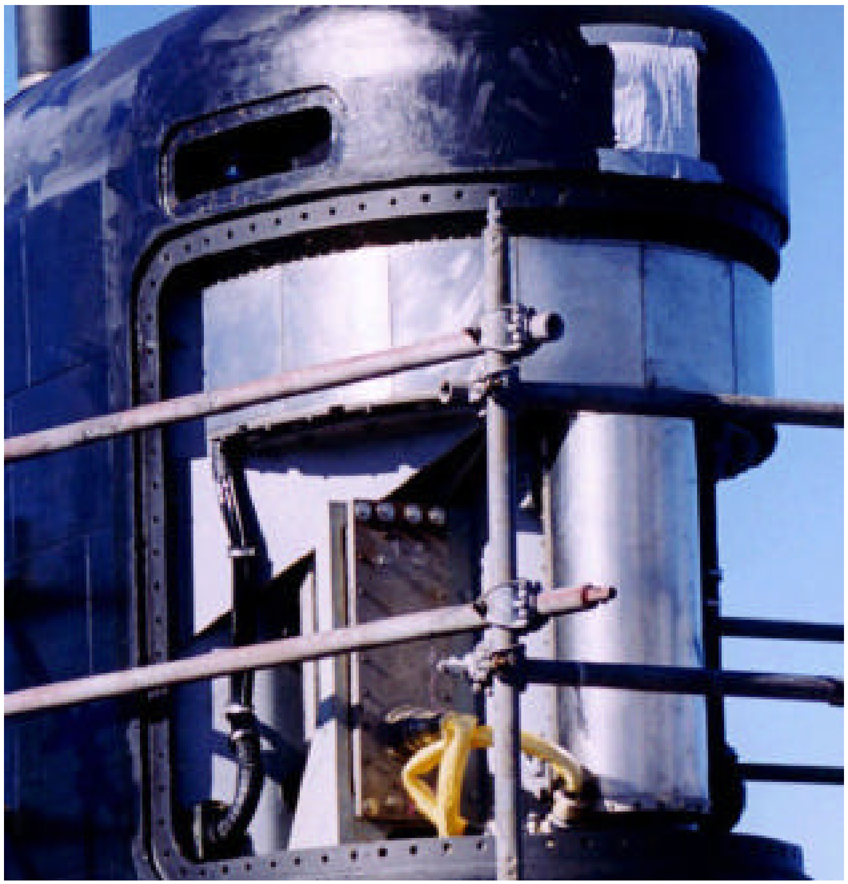


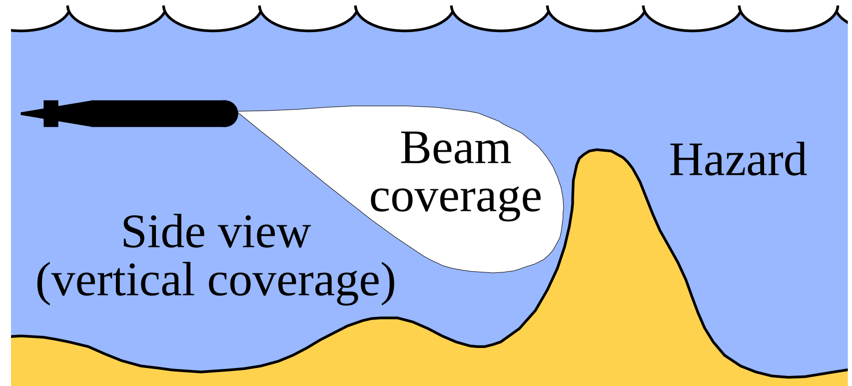
- **Programs currently constructed in C++**

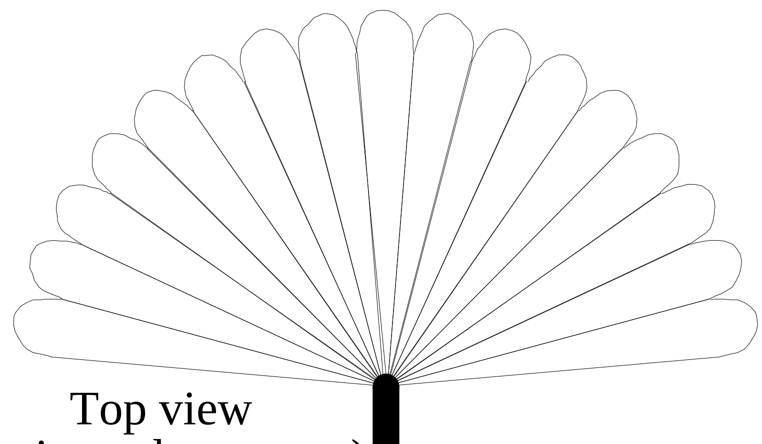
```
int main() {
    PNThresholdQueue<T> P (queueLen, maxThresh);
    PNThresholdQueue<T> Q (queueLen, maxThresh);
    MyProducerNode      A (P);
    MyTransmuterNode    B (P, Q);
    MyConsumerNode      C (Q);
}
```



Application: Sonar Beamforming



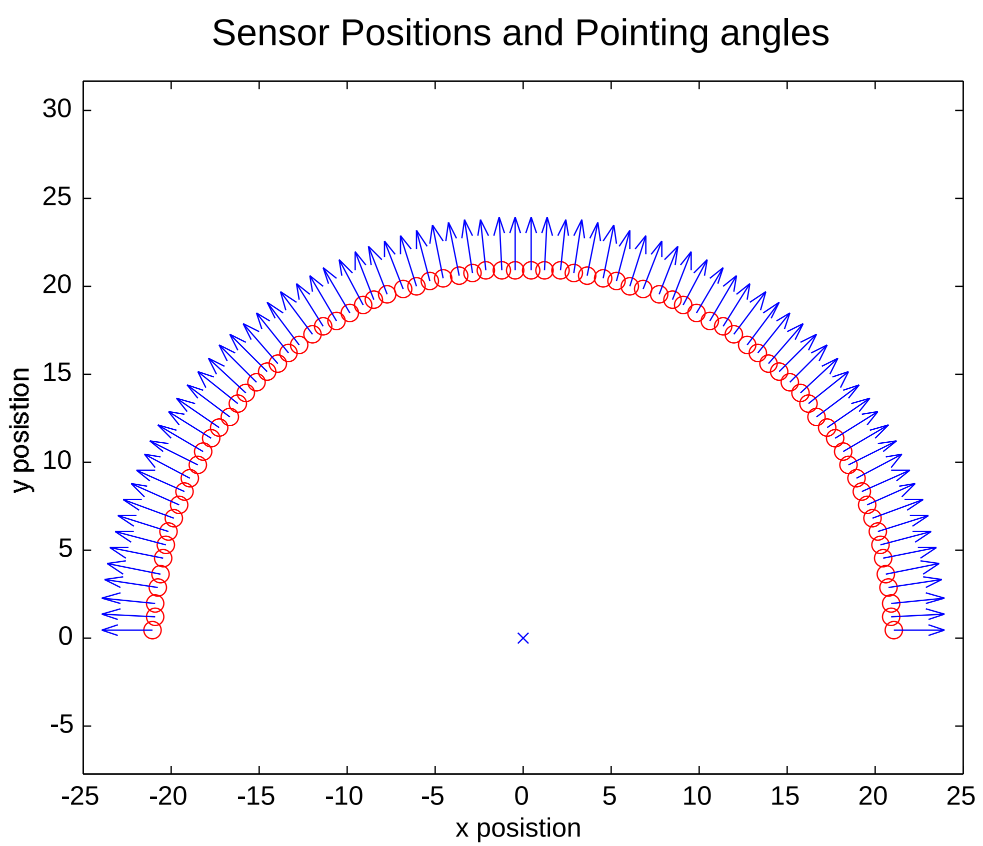


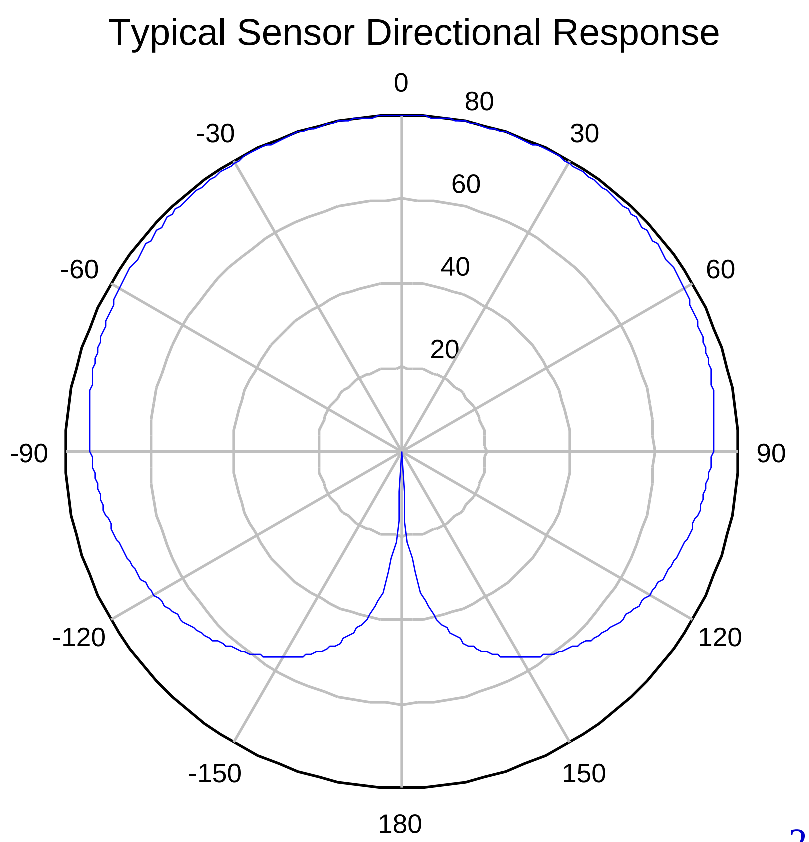


Collaboration with UT Applied Research Laboratories

Sonar Hydrophone Array

- **Array of directional hydrophone sensors**
- **Each sensor has a wide directional response**

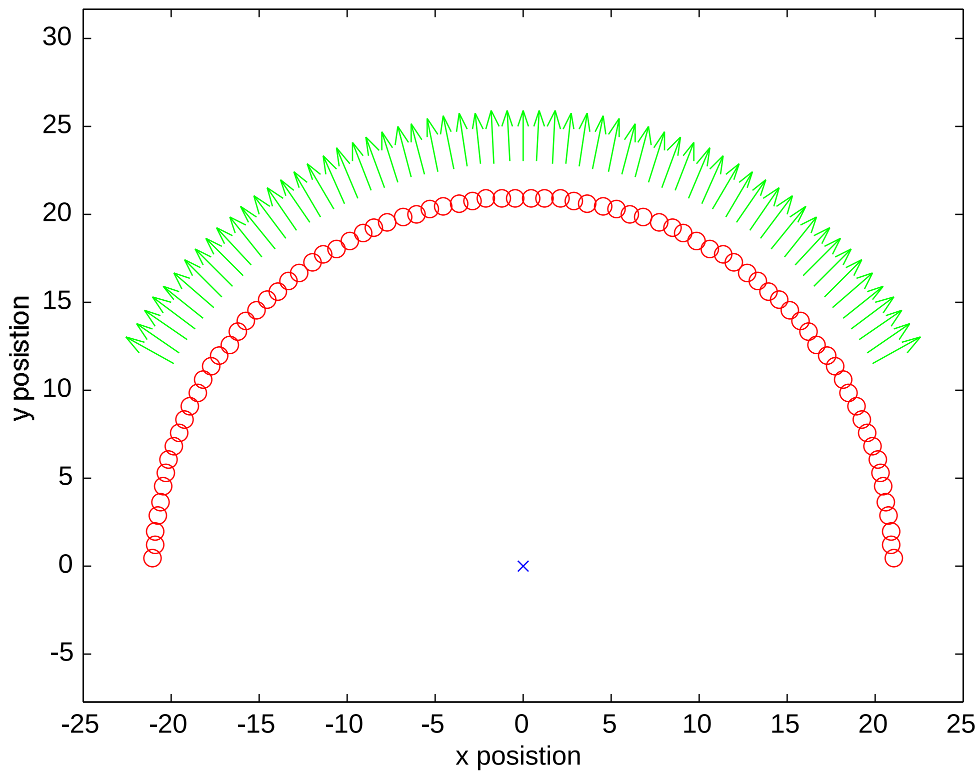




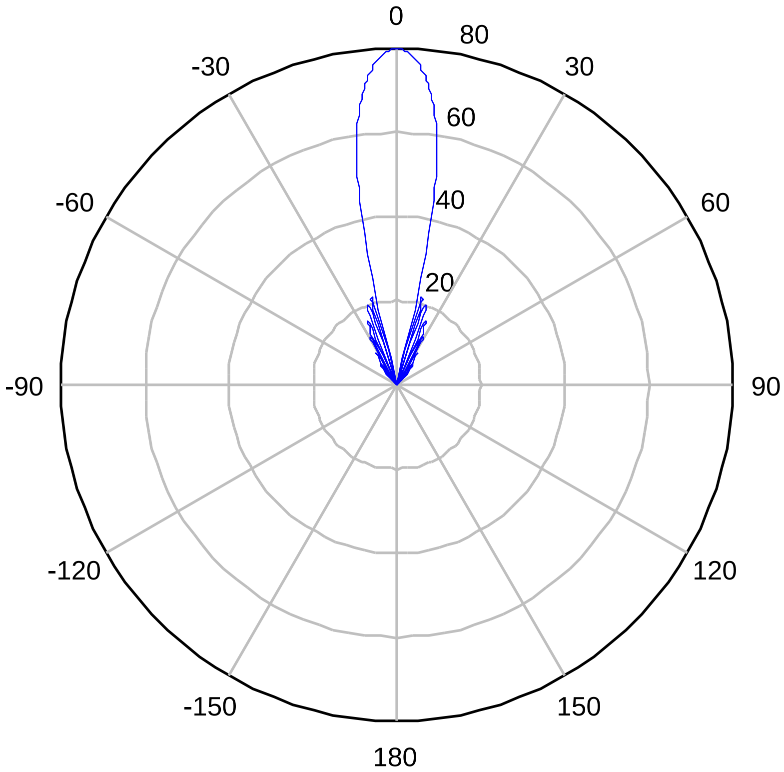
Sonar Beamforming

- A beamformer is a directional (spatial) filter
- Beams with a narrow response pattern are formed

Desired Beam Pointing Angles



Typical Beam Directional Response

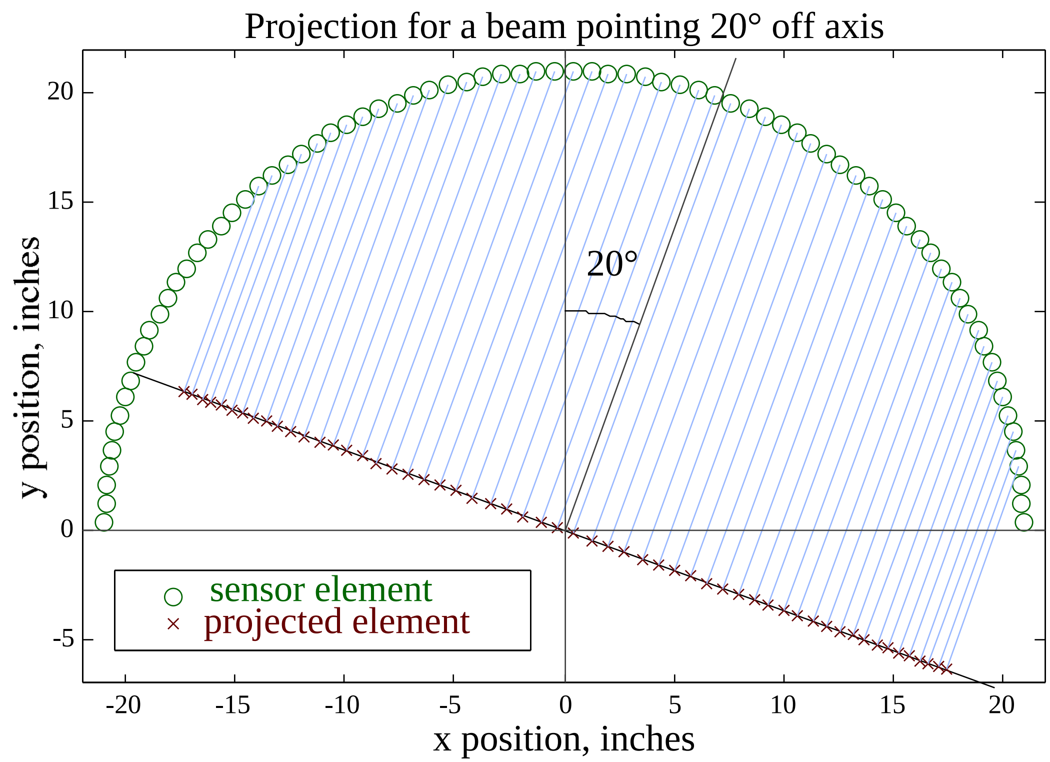


Time-Domain Beamforming

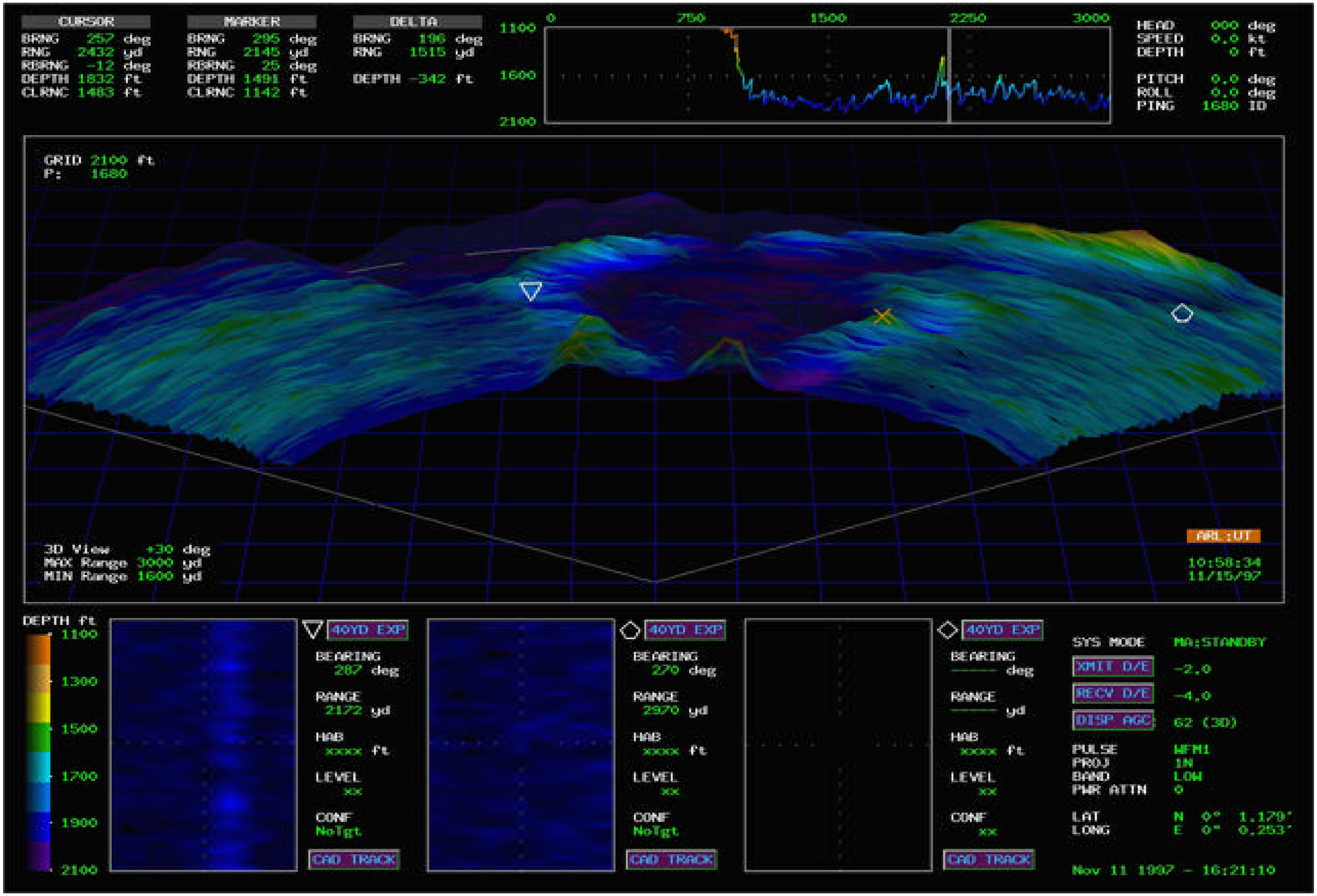
- **Delay-and-sum weighted sensor outputs**
- **Geometrically project the sensor elements onto a line to compute the time delays**

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

$b(t)$	beam output
$x_i(t)$	i^{th} sensor output
τ_i	i^{th} sensor delay
α_i	i^{th} sensor weight

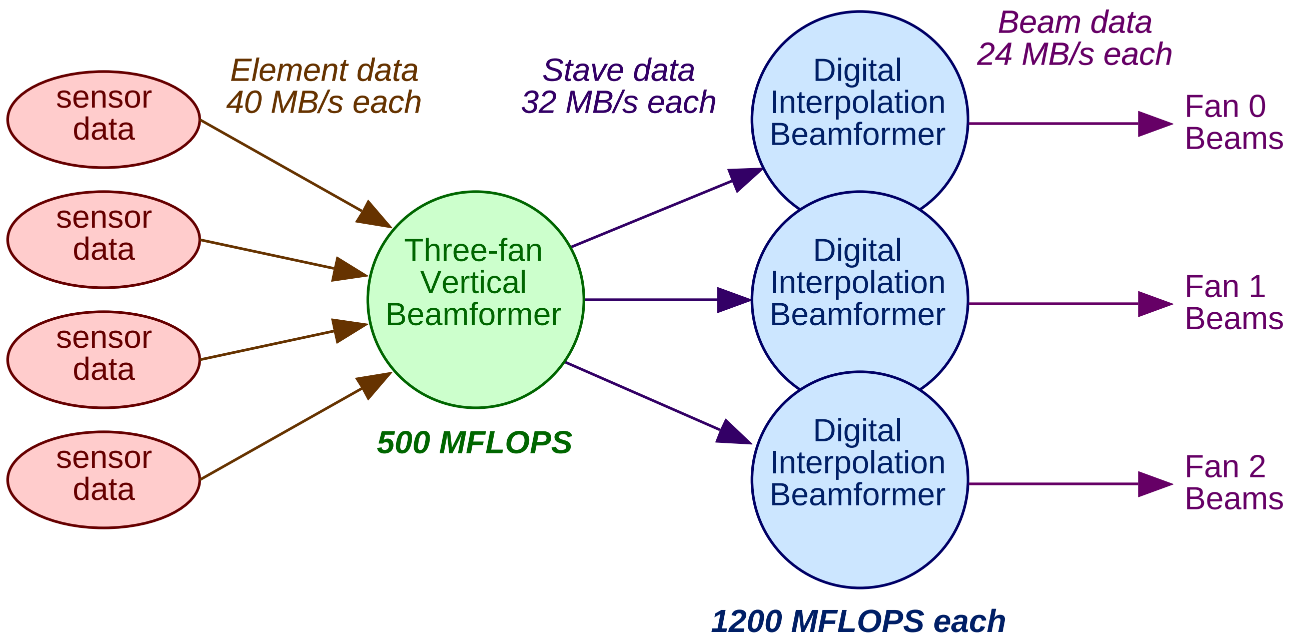


Sample Sonar Display

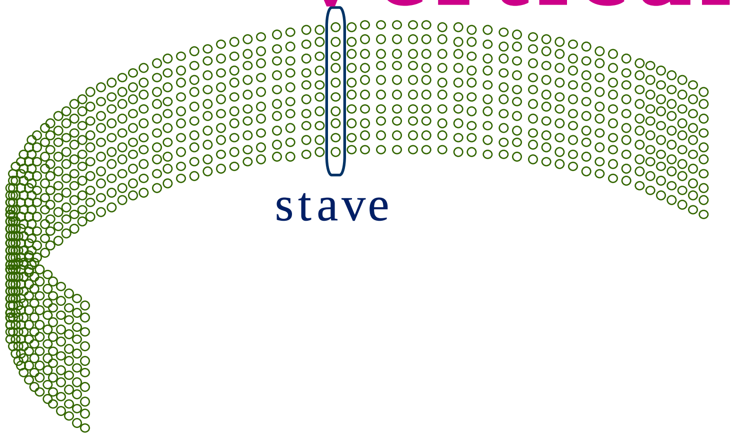


4-GFLOP 3-D Beamformer

- **80 horizontal x 10 vertical sensors**
- **Data at 160 MB/s input, 72 MB/s output**
- **Collapse vertical sensors into 3 sets of 80 staves**
- **Do horizontal beamforming, 3 x 1200 MFLOPS**



Vertical Beamformer

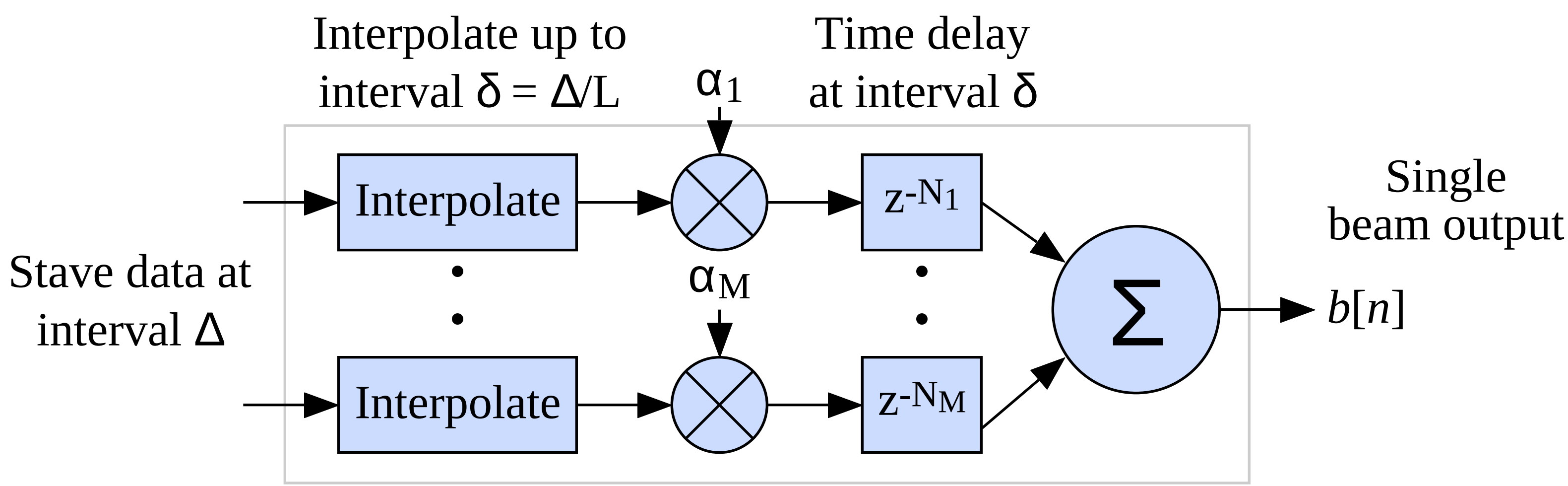


Multiple vertical transducers for every horizontal position

- **Vertical columns combined into 3 stave outputs**
  - **Multiple integer dot products (16x16-bit multiply, 32-bit add)**
  - **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 to preserve frequency content, interpolate to obtain desired time delay resolution**



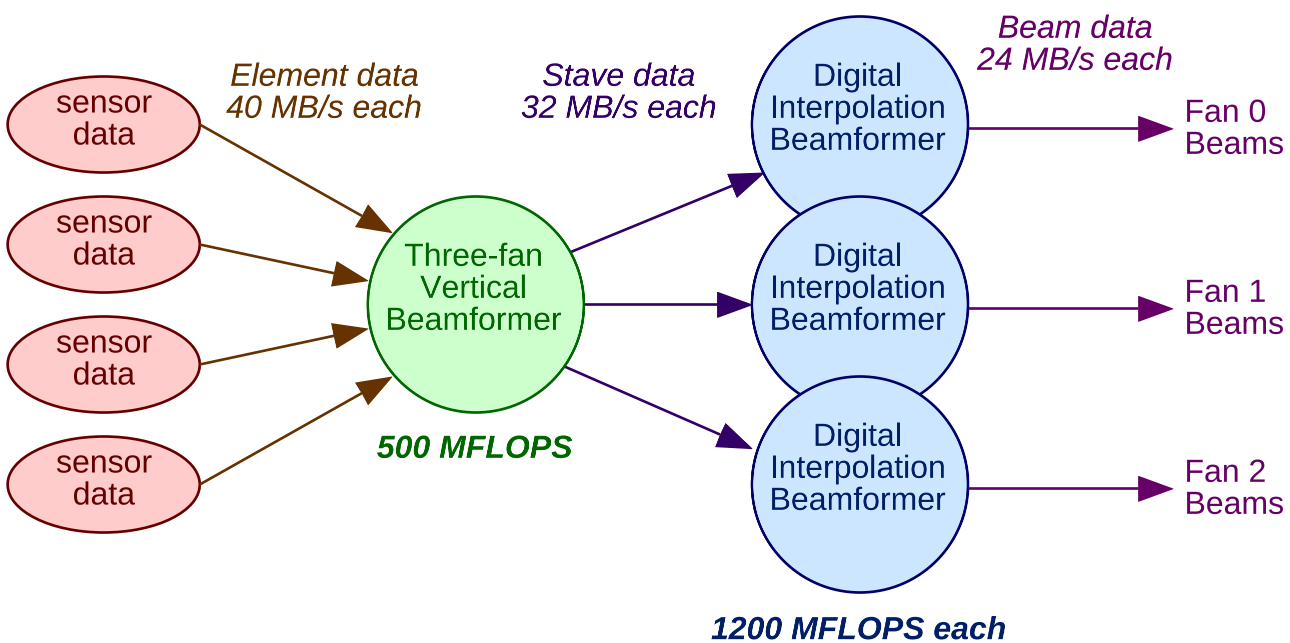
Digital Interpolation Beamformer

- **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)**

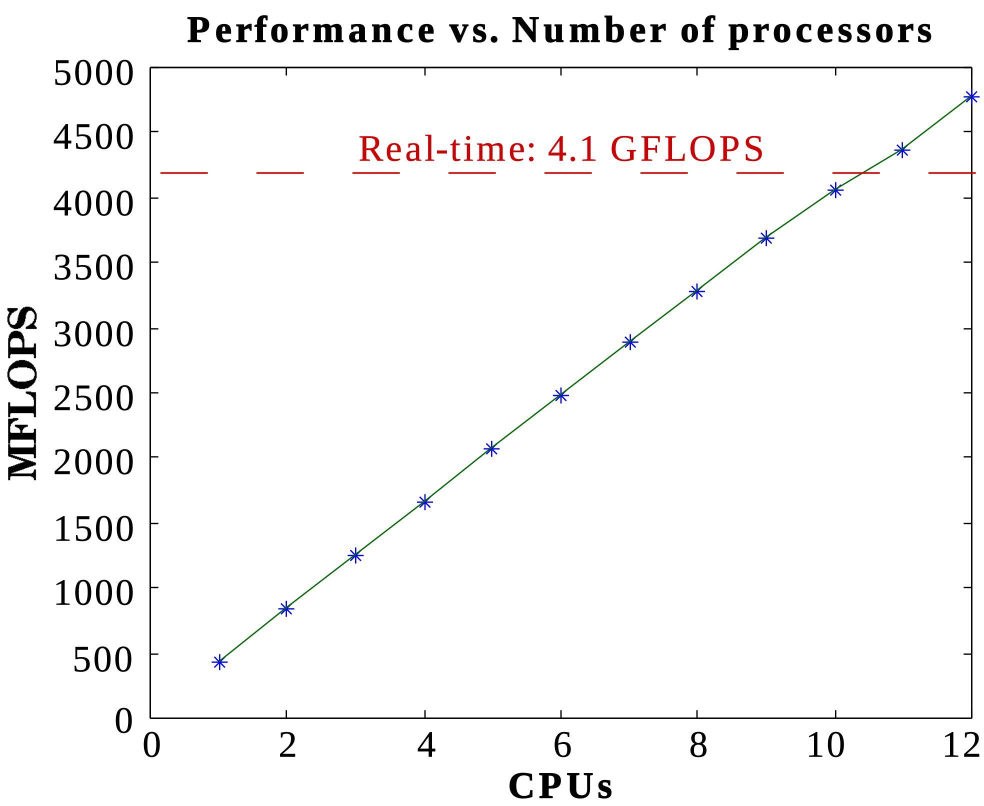
Integration with Framework

- A single processor (thread) cannot achieve real-time performance for any one node
- Each beamformer node utilizes a pool of 4 threads (data parallelism)
- Performance dictates number of worker threads



Performance Results

- Sun Ultra Enterprise 4000 with twelve 336-MHz UltraSPARC-IIs, 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%

Summary

- Bounded Process Network model extended with firing thresholds from Computation Graphs
  - Provides correctness and determinate execution
  - Naturally models parallelism in system
  - Models algorithms on overlapping continuous streams of data
- Multiprocessor workstation implementataion
  - 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