Real-Time Sonar Beamforming on a Unix Workstation using Process Networks and POSIX Threads

Gregory E. Allen [1,2]
Brian L. Evans [1]
David C. Schanbacher [1]

[1] **Embedded Signal Processing Laboratory**
The University of Texas at Austin

[2] ARL
The University of Texas at Austin

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

Motivation

- Beamforming is computationally intensive (GFLOPS).
- Traditionally limited to expensive custom hardware.
- Real-time software implementation on a workstation.
  - Multi-processor workstations.
  - Real-time threads supported by modern operating systems.
  - Native signal processing.

Objectives

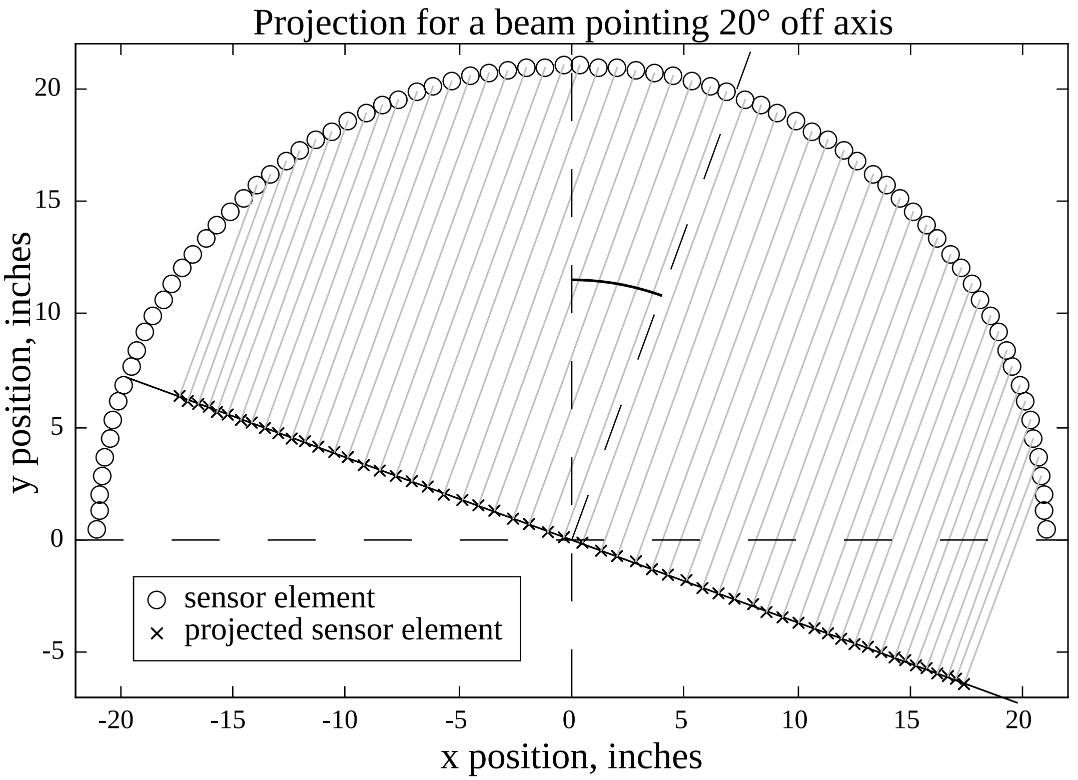
- Implement a 4 GFLOP sonar beamformer in software.
  - Evaluate the performance of sonar beamforming algorithms.
  - Capture parallelism and guarantee determinate bounded execution.
  - Use lightweight threads on a multiprocessor workstation.
- Assess feasibility of replacing a real-time custom hardware beamformer with a Unix workstation.

Time-Domain Beamforming

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

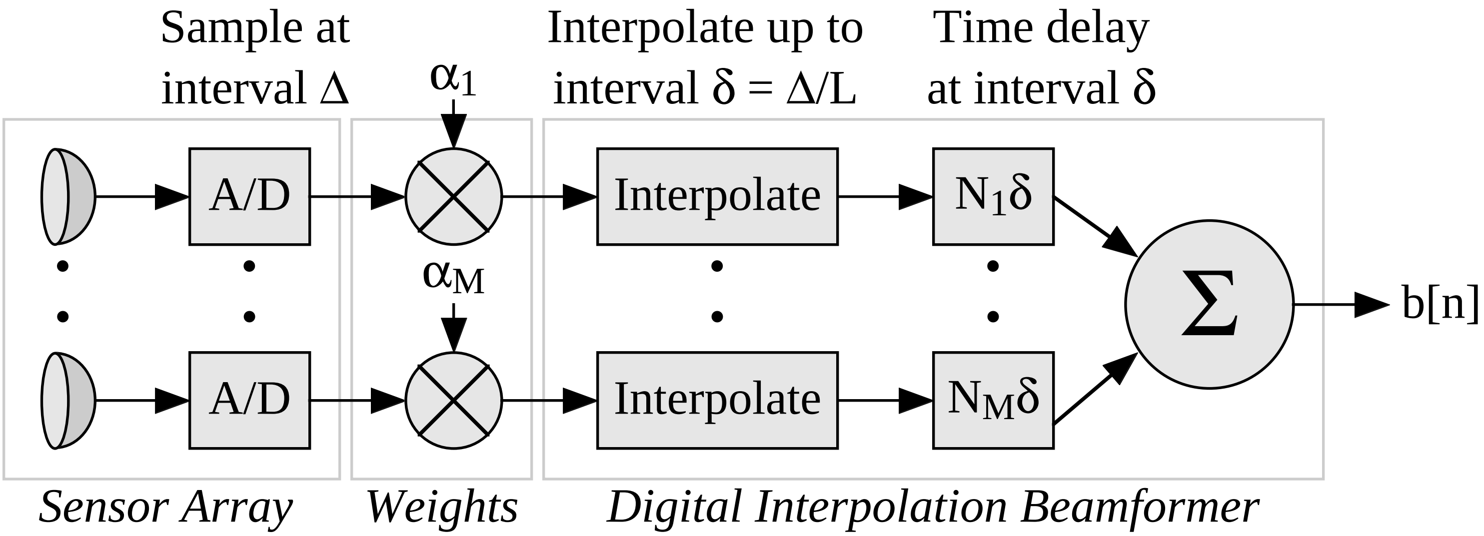
$$b(t) = \sum_{i=1}^{M} \alpha_i \, x_i(t-\tau_i)$$

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


Projection for a beam pointing 20° off axis

Interpolation Beamforming

- Quantized time delays perturb beam pattern.
- Sample at just above the Nyquist rate.
- Interpolate to obtain desired time-delay resolution.



Interpolation Beamforming

- Modeled as a sparse FIR filter:

• M	total sensors in array	(80)
• S	sensors used to calculate beam	(50)
• D	maximum geometry delay	(31)
• P	points for interpolation filter	(2)
• B	number of beams calculated	(61)

Coefficient filter length: $K = (D+P-1)\, M$ (2560)

Non-zero coefficients: $C = P\ S$ (100)

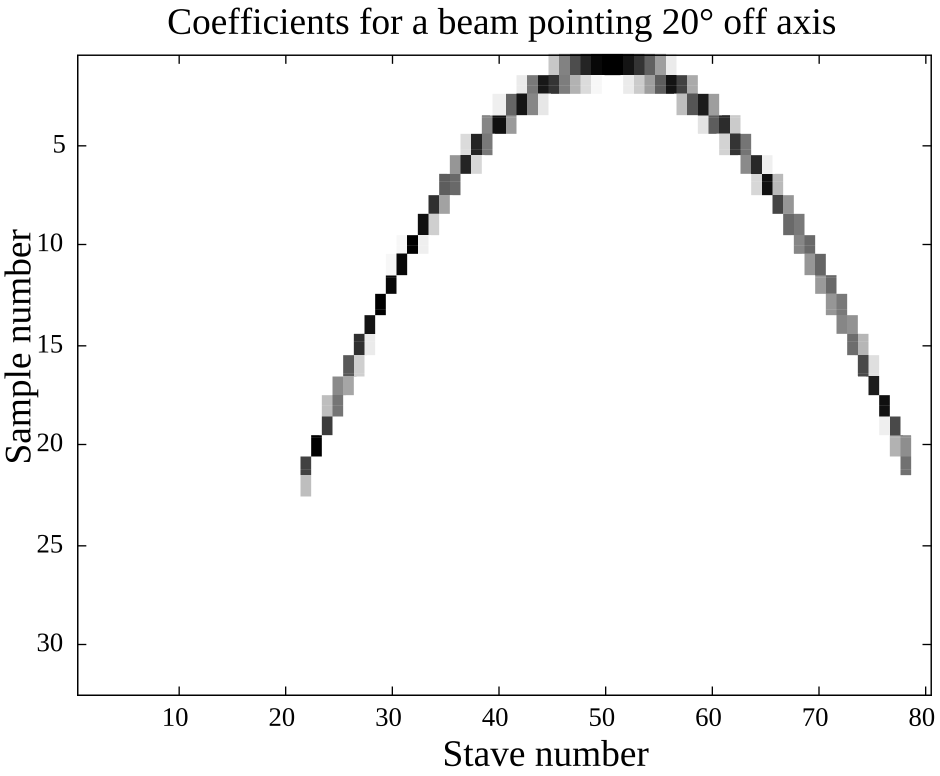
Sparsity = $1-C/K$ (96%)

MACs per sample = $\boldsymbol{B\ C}$ **(6100)**

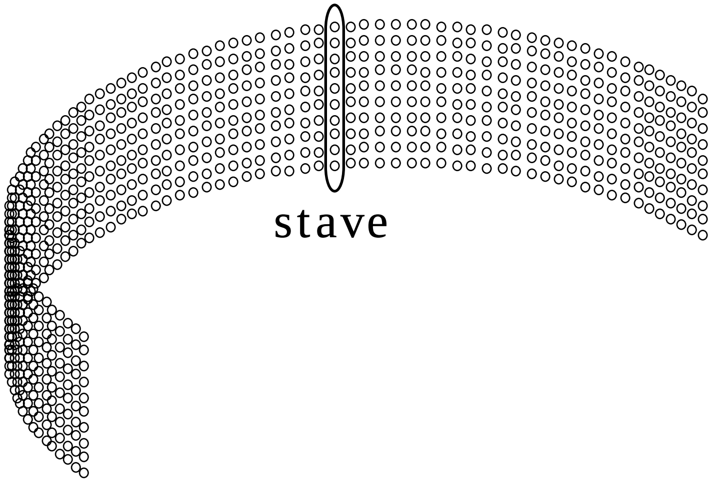
$$\underset{(1\text{ by }K)}{\begin{bmatrix}\text{Incoming Data}\end{bmatrix}} \times \underset{(K\text{ by }B)}{\begin{bmatrix}\text{Beam} & & \text{Beam}\\ 1 & \cdots & \text{B}\\ \text{coefs} & & \text{coefs}\end{bmatrix}} = \underset{(1\text{ by }B)}{\begin{bmatrix}\text{Beam Data}\\ \text{(1 sample)}\end{bmatrix}}$$

Interpolation Beamformer

- Performed in floating-point to preserve dynamic range.
- Generate sparse FIR beam coefficients using Matlab.
  - 2560-point sparse FIR filter viewed in 2-D.
  - Zero-valued coefficients are white, non-zero coefficients are black.
  - Array shape is visible in beam coefficients.



Vertical Beamforming

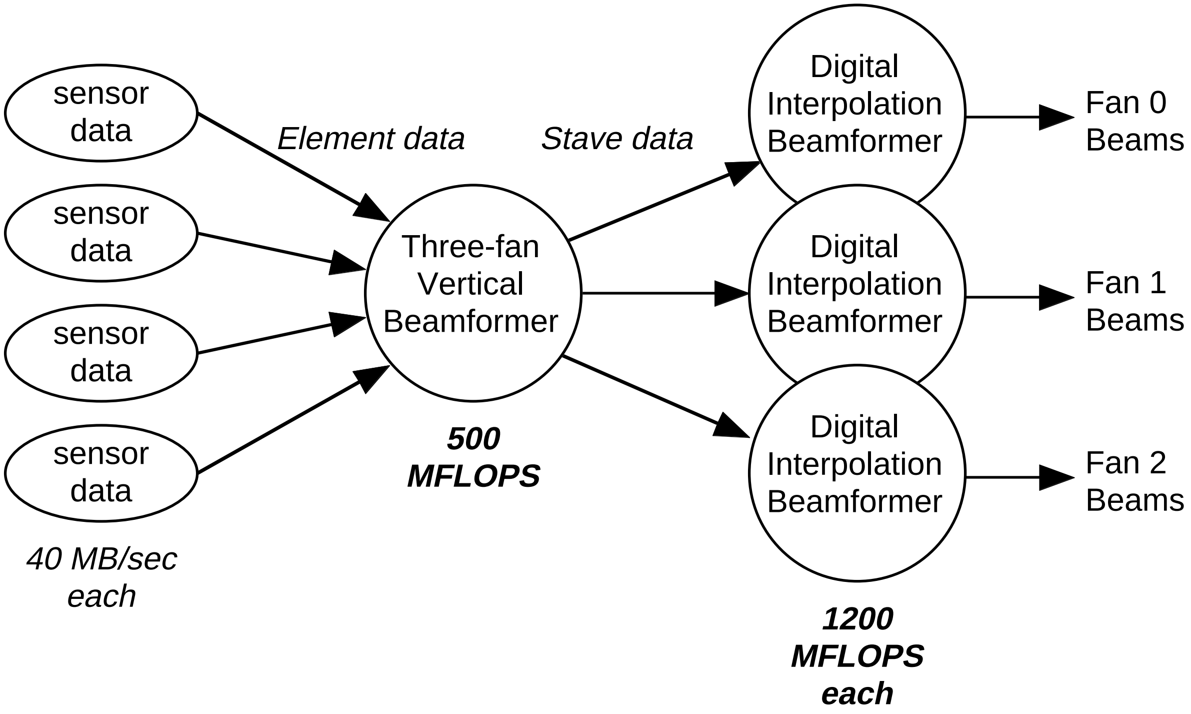


Multiple vertical transducers for every horizontal position.

- Each vertical sensor column is combined into a *stave*.
  - No time delay or interpolation is required.
  - Staves are calculated by a simple dot product.
  - Integer-to-float conversion must be performed.
  - Output data must be interleaved.

System Block Diagram

- Vertical beamformer forms 3 sets of 80 staves from 10 vertical elements each.
- Each horizontal beamformer forms 61 beams from the 80 staves, using a two-point interpolation filter.

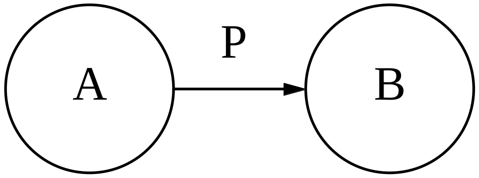


Formal Design Methodology

- The *Process Network* model [Kahn, 1974].
- Superset of dataflow models of computation.
- Captures concurrency and parallelism.
- Provides correctness.
- Guarantees determinate execution of the program.

The Process Network Model

- A program is represented as a directed graph
  - Each node represents an independent process.
  - Each edge represents a one-way FIFO queue of data.



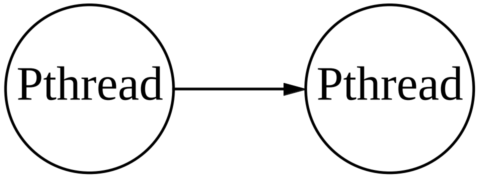
- A node may have any number of input or output edges, and may communicate only via these edges.
- A node suspends execution when it tries to consume data from an empty queue (blocking reads).
- A node is never suspended for producing, so queues can grow without bound (non-blocking writes).

Bounded Scheduling

- Infinitely large queues cannot be implemented.
- The following scheduling policy will 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 the producer associated with it can fire.
- Fits the thread model of concurrent programming.

Process Network Implementation

- Implemented in C++ using POSIX Pthreads.
- Each node corresponds to a thread.

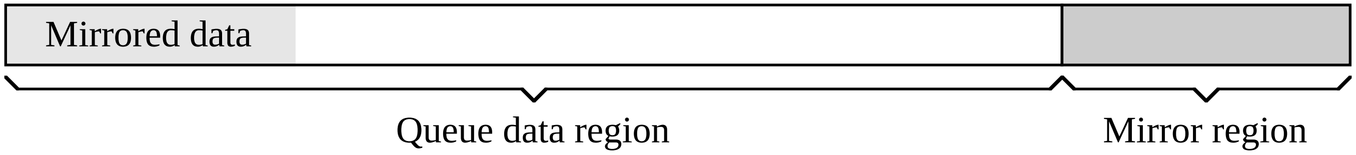


- Low-overhead, high-performance, scalable.
- Granularity larger than a thread context switch.
- Symmetric multiprocessing operating system dynamically schedules threads.
- Efficient utilization of multiple processors.

Process Network Queues

- Nodes operate directly on queue memory, avoiding unnecessary copying.

- Queues use mirroring to keep data contiguous.



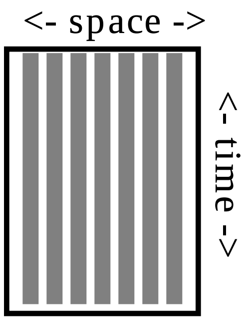
- Compensates for the lack of circular address buffers.

- Queues tradeoff memory usage for overhead.

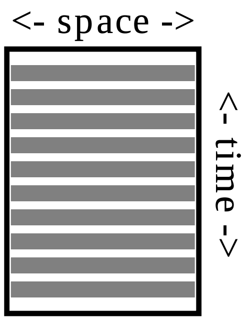
- Virtual memory manager maintains data circularity.

Exploiting Parallelism

divide by beam vs. divide by time



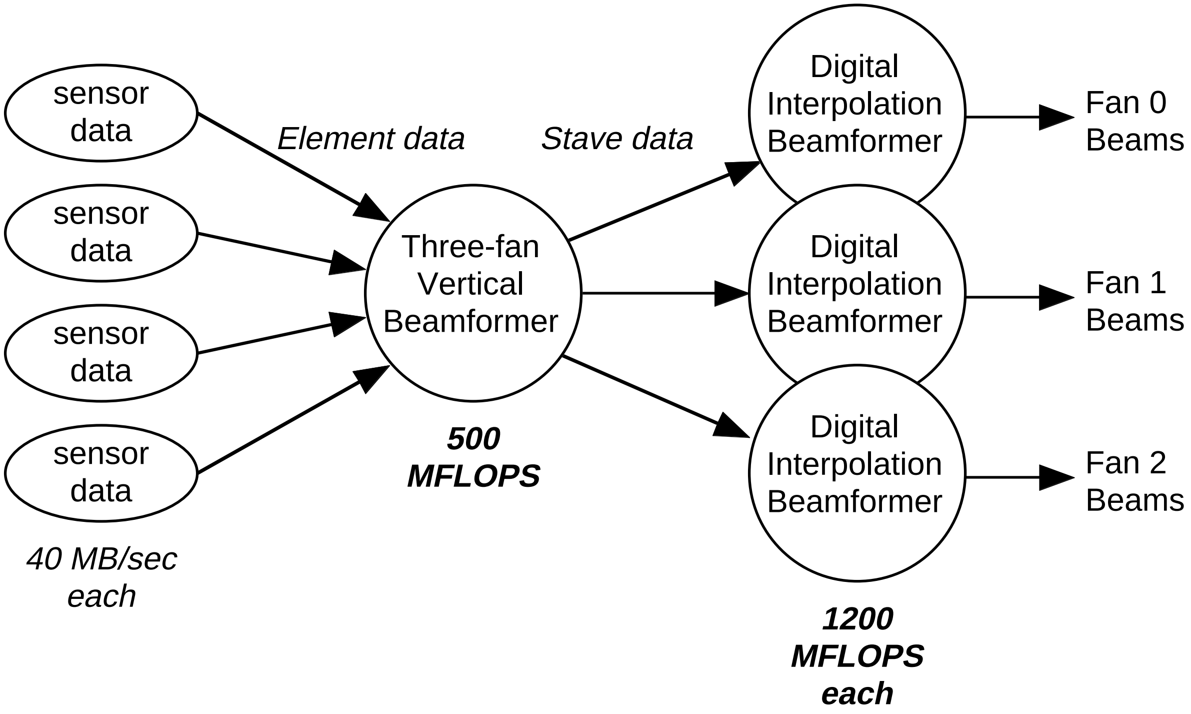
low	*Latency*	high
low	*Memory Usage*	high
poor	*Cache Usage*	good
partial	*Style*	batch
embedded	*Target*	workstation



- Strategies for high performance on a workstation
  - Throughput is more importatant than memory usage or latency.
  - Keep kernel calculations smaller than the cache.
  - Calculate as much as possible while the data is in cache.

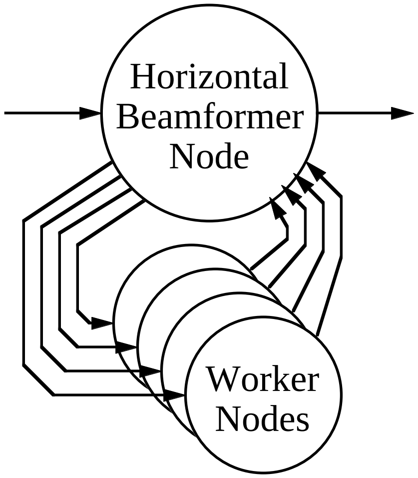
System Implementation

- Vertical beamformer forms 3 sets of 80 staves from 10 vertical elements each.
- Each horizontal beamformer forms 61 beams from the 80 staves, using a two-point interpolation filter.



Integration with Process Networks

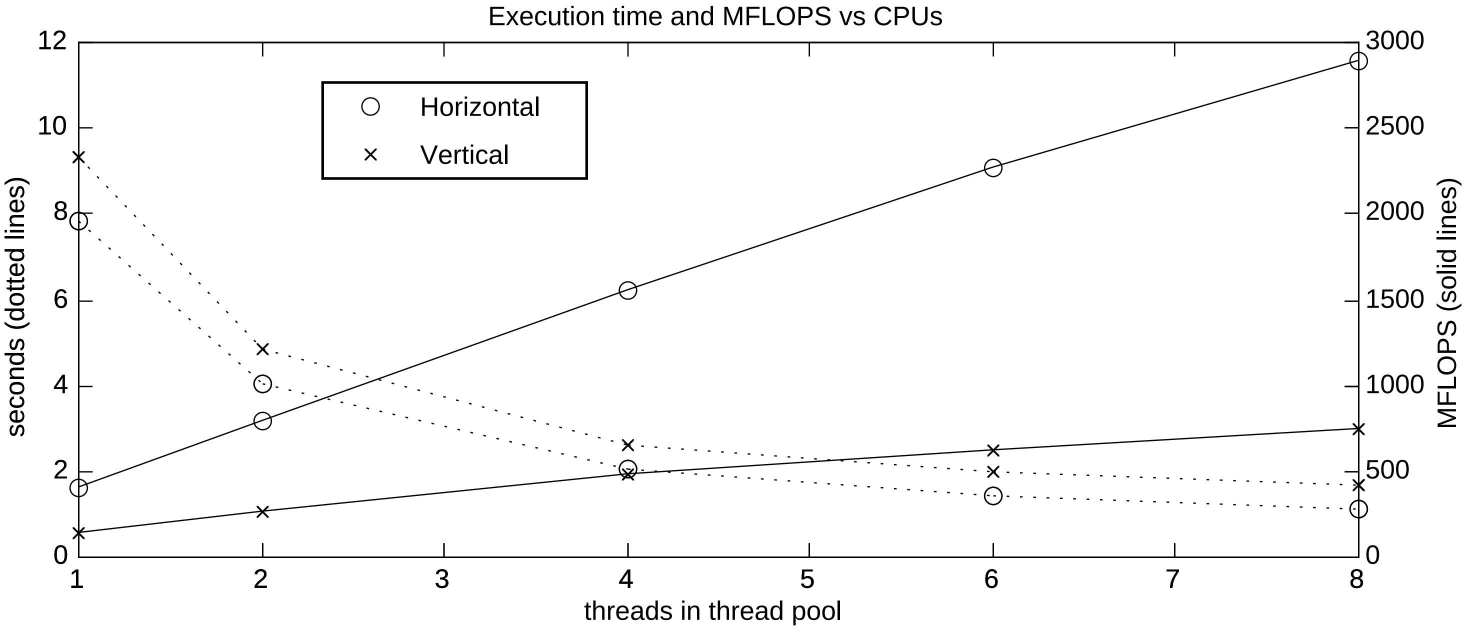
- A single CPU cannot achieve real-time performance.
- A horizontal beamformer node manages multiple worker nodes.



- The number of worker nodes is set as performance requirements dictate.
- Similar to the traditional thread pool model.

Kernel Performance Results

- Ten trial mean execution time for 2.6 seconds of data.
- Sun Ultra Enterprise 4000 with 8 UltraSPARC-II CPUs at 336 MHz, running Solaris 2.6.



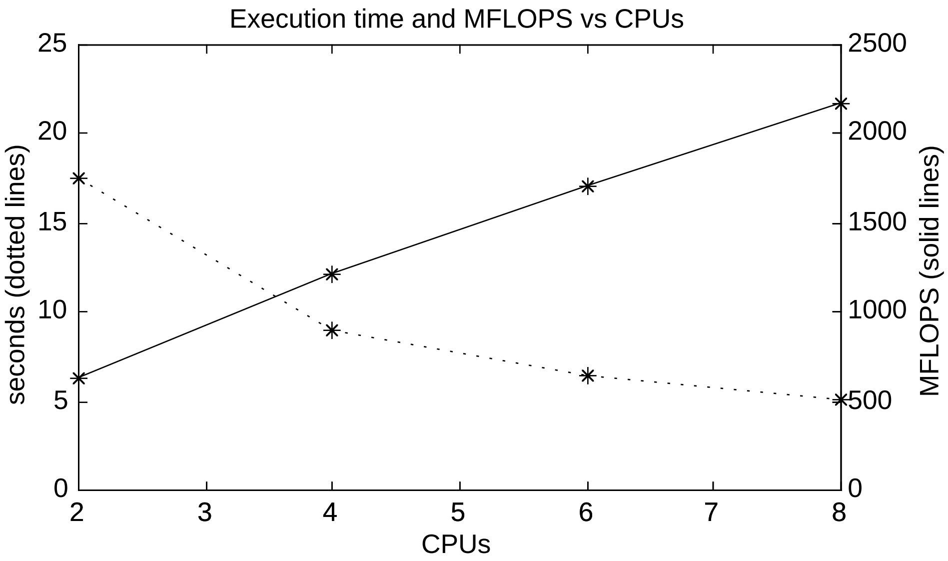
	kernel performance	scalability
Horizontal	good at 1.22 FLOPS per cycle	good
Vertical	poor at 0.40 FLOPS per cycle	poor

System Performance Results

- Process network and thread pool results are within 1%, overhead is small.

Type	Seconds	MFLOPS
thread pool	5.053	2159.0
process network	5.024	2171.5

- Process network uses 25% less memory with lower latency.



- Scalability is evaluated by disabling CPUs.
- Process network scalability is good.
- Will continue to scale as more CPUs are added.

Conclusion

- Implemented a 4 GFLOP software sonar beamformer.
    - Divide the computation by time and not by beam.
    - Use the Process Network model of computation.
    - POSIX Pthreads and a symmetric multiprocessing workstation.
- This 4 GFLOP beamforming system could execute in real time with 16 UltraSPARC-II CPUs at 336 MHz.
- We achieve real-time beamforming at a substantial savings in development cost and time.