A Comparison of Parallel Workstation Sonar Beamforming Implementations

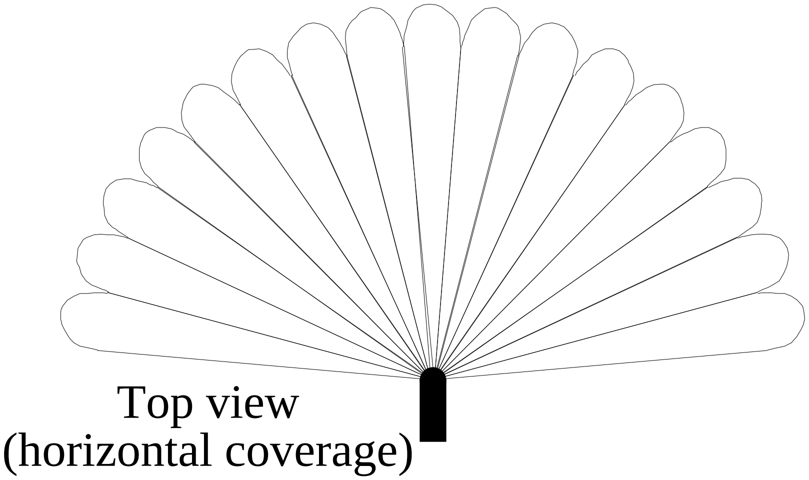
Gregory E. Allen

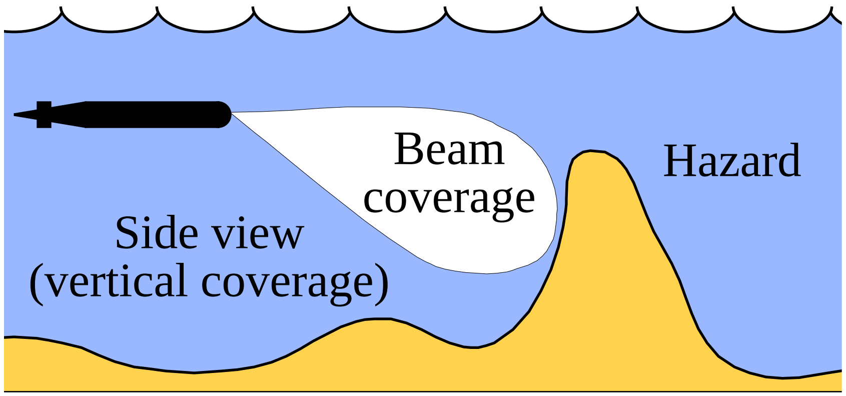
Applied Research Laboratories
The University of Texas at Austin

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

What is Beamforming?

- **A *beamformer* is a spatial filter that operates on the output of an array of sensors**
  - **Enhances sound from a desired direction, rejects others**
  - **Filter design chooses which direction a beam points**
  - **We can determine from which direction a sound is coming**
- **Many beams formed, each in a different direction**



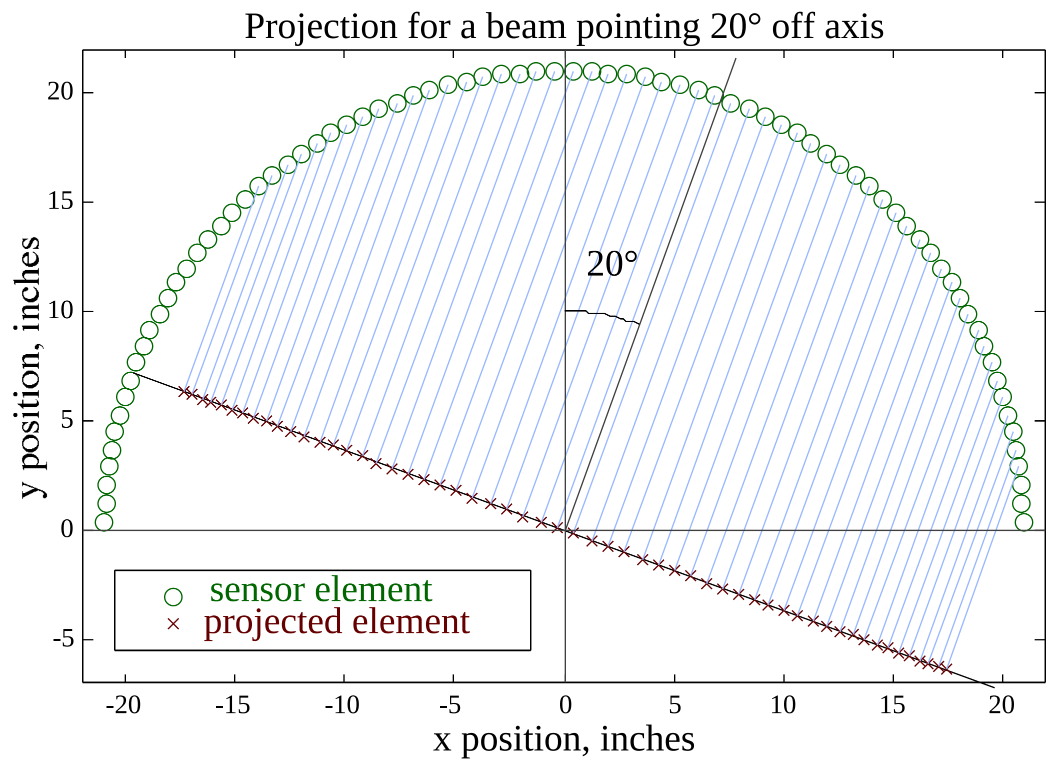


Time-Domain Beamforming

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

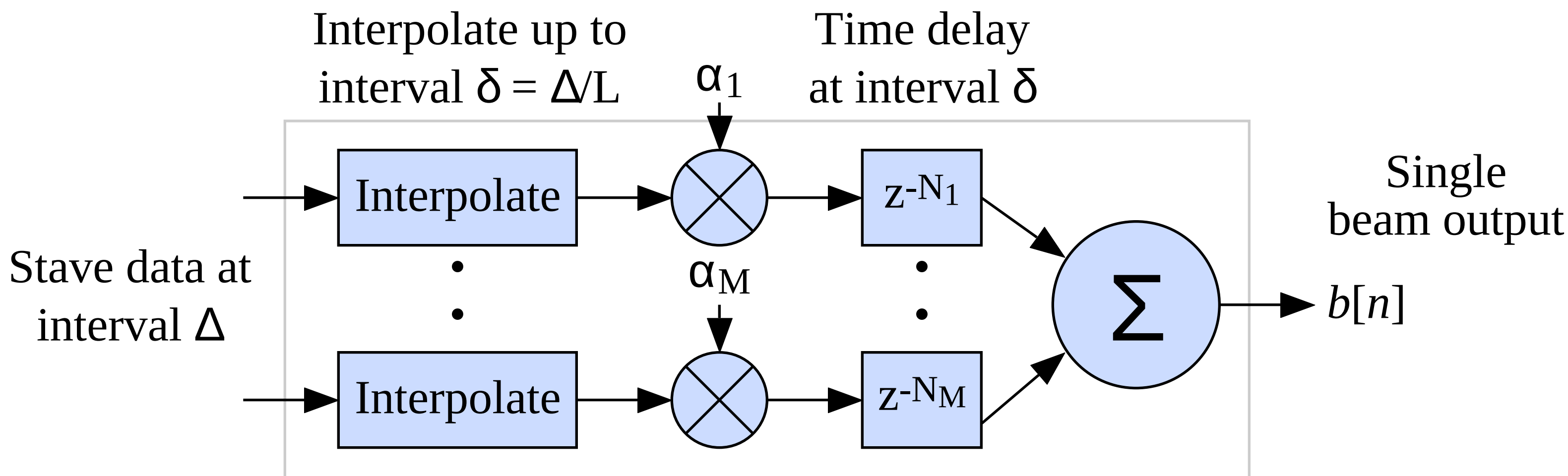
$$b(t) = \sum_{i=1}^{M} \alpha_i \, x_i(t - \tau_i)$$

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



Interpolation Beamforming

- **Quantized time delays distort the beam pattern**
- **Sample at just above the Nyquist rate, interpolate to obtain desired time delay resolution**



Digital Interpolation Beamformer

- **Multiple different beams formed from same data**

Motivation

- **High-performance, low production volume (~100 MB/s I/O; 1-20 GFLOPS; under 50 units)**
- **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)**
- **Wish to target commodity workstations**
  - **Symmetric multiprocessing (SMP) operating systems**
  - **Leverage native signal processing (NSP) kernels**
  - **Development environment and target architecture are same**
  - **Concurrent development on less powerful workstations**
  - **Reduce development time and cost**

Objective

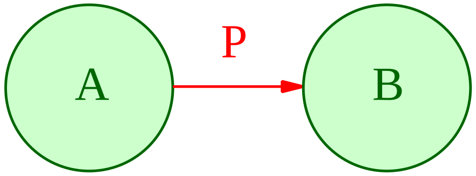
- **Given an UltraSPARC-II beamforming kernel...**
  - **Highly optimized C++ (loop unrolling and SPARCompiler5.0)**
  - **Operates at 440 MFLOPS at 336 MHz (60% of peak)**
- **Compare beamforming perfomance using different frameworks for parallelism**
  - **Message Passing Interface (MPI)**
  - **Computational Process Networks (CPN)**
  - **Extend CPN implementation with MPI: a hybrid**
- **Measure performance on Sun Ultra Enterprise 4000**
  - **Eight 336 MHz UltraSPARC-II processors**
  - **2 GB RAM, Solaris 2.6**

Message Passing Interface

- **A standard interface for**
  - **Explicit message passing in application programs**
  - **MIMD distributed memory concurrent computers**
- **A library for C or Fortran, developed by about 80 people from 40 organizations (edu, gov, com)**
- **Intended to be portable and easy to use**
- **Many implementations exist, free and commercial**
- **Using MPI from Sun's HPC 2.0 Package**
  - **Based on MPICH (public from ANL)**
  - **Claims to be thread safe**

Computational Process Networks

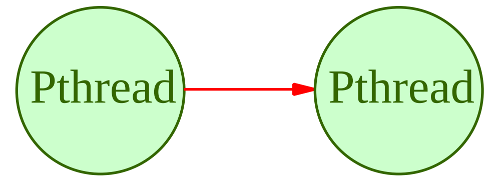
- **Based on formal Process Network model [Kahn, 1974]**
  - **Program is represented as a directed graph**
  - **Captures concurrency and parallelism**
  - **Provable model provides correctness and determinate execution**



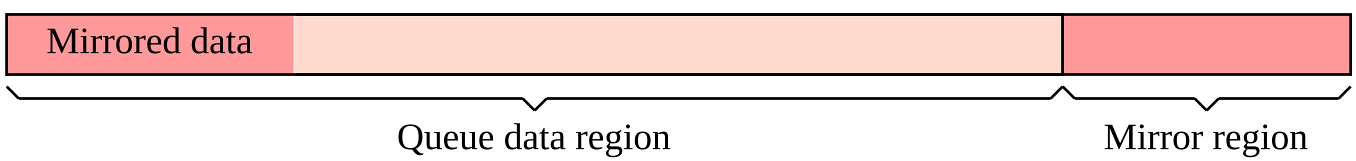
- **Leverage bounded scheduling [Parks, 1995]**
  - **Permits realization in finite memory, regardless of scheduler**
- **Extend this model with firing thresholds, similar to Computation Graphs [Karp & Miller, 1966]**
  - **Models algorithms on overlapping continuous streams of data, e.g. digital filters and fast Fourier transforms (FFTs)**
  - **Decouples computation (node) from communication (queue)**
  - **Allows compositional parallel programming**

High-Performance Implementation

- **Designed for real-time high-throughput systems**
- **Uses POSIX lightweight Pthreads**



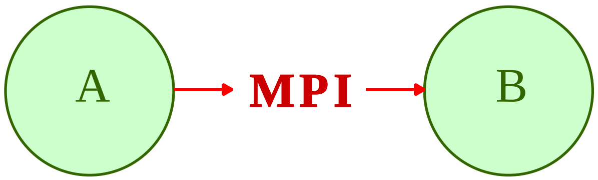
  - **Each node corresponds to a thread**
  - **Portable to many different operating systems**
  - **Optional fixed-priority real-time scheduling**
- **Operate directly on queue memory to avoid copying**
- **Queues use mirroring to keep data contiguous**



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

Process Networks with MPI

- Threads require shared memory, MPI does not
- Allow a queue to be implemented across MPI

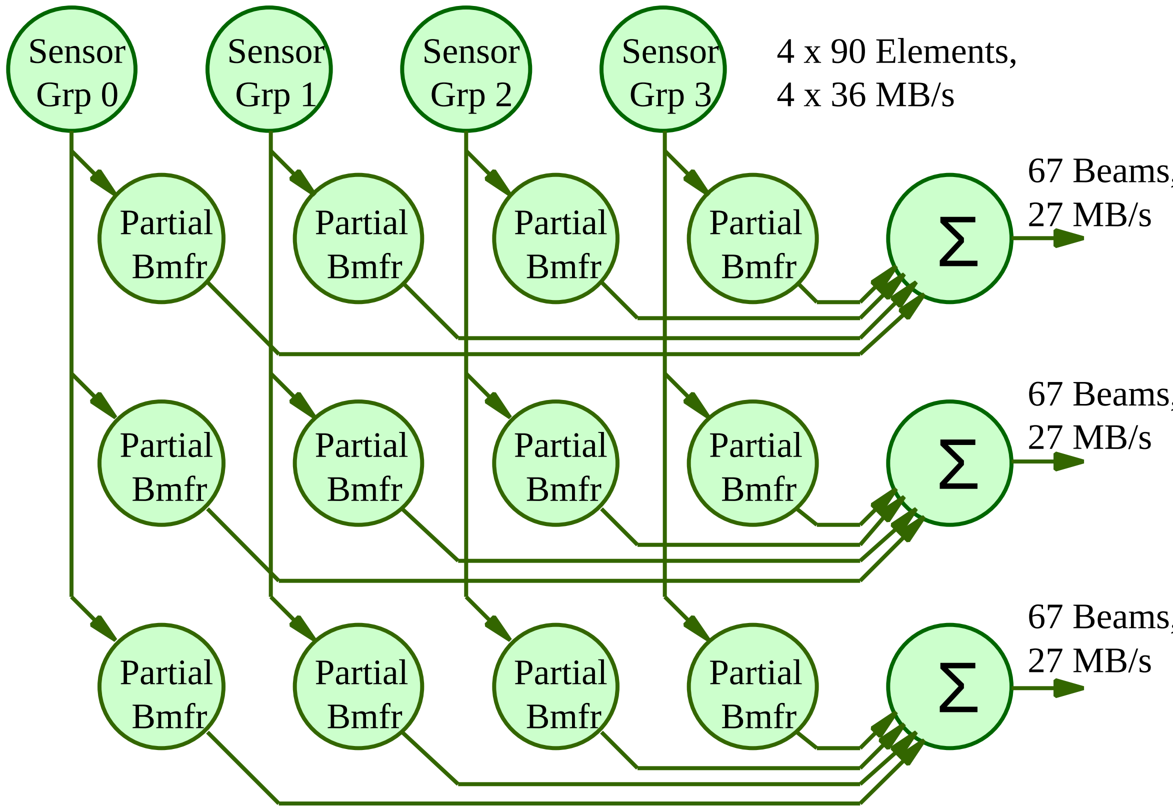


  - Easy to implement, but
  - Programming style must change (threaded vs. SPMD)
  - Mapping nodes across processes and matching up queues is difficult, and best left to automated tools
- Other point-to-point technologies may be a better fit for Process Networks (sockets, RACEway)

Beamformer Block Diagram

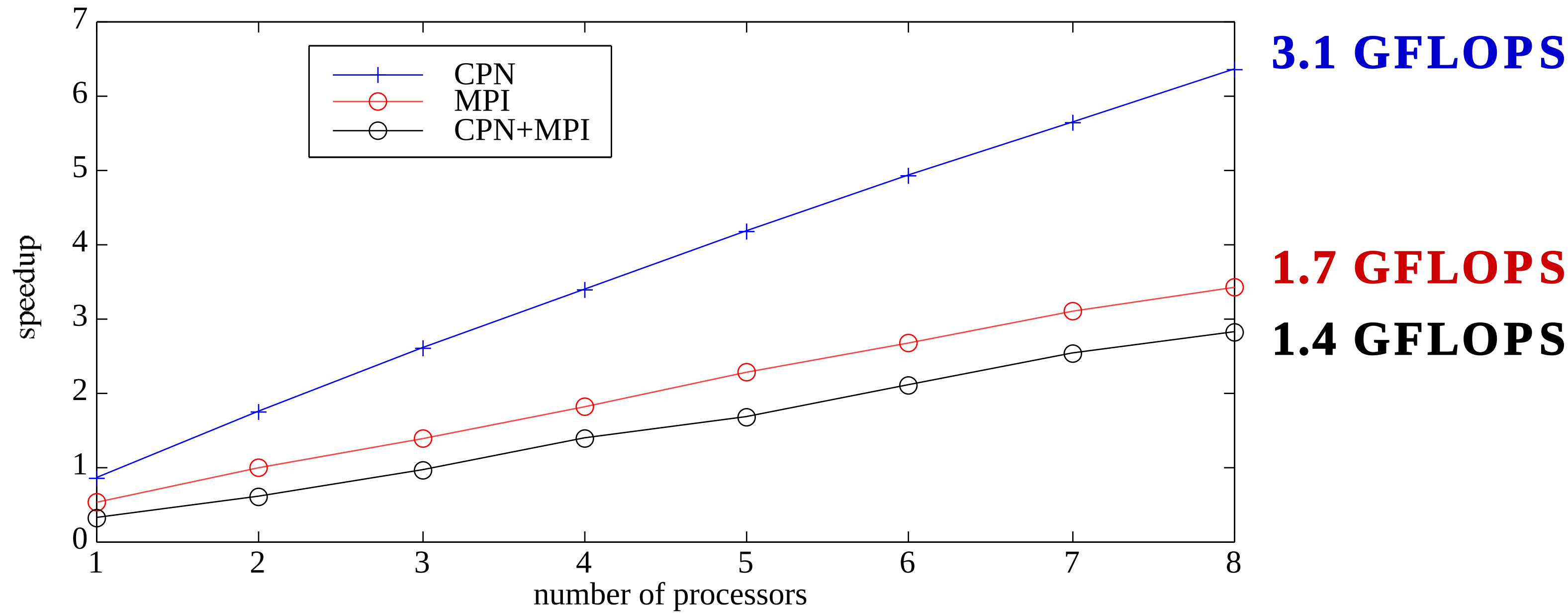
- **Calculate 201 beams from 360 sensor elements**
  - **21 GFLOPS total, requires about fifty 336 MHz UltraSPARC-IIs**
  - **144 MB/s sensor data in, 81 MB/s beam data out**

- ***Partial beamforming* divides the problem functionally**
- **Each node calculates different part, results are summed**
- **Average node is 1.75 GFLOPS (2.4 max)**
- **Each part needs to operate at real-time**
- **Workstation cluster**



Performance Results

- **Benchmarked on a single SMP machine**
- **Results as compared to sequential case (480 MFLOPS)**
- **Slowdown on one processor: CPN 16%, MPI 84%**
- **Speedup on eight processors: CPN 6.5, MPI 3.5**



My Comments on MPI

- **MPI is straightforward to use, and a step in the right direction, but**
  - **It lacks any formal methodology**
  - **It needs a C++ class interface (added in MPI-2)**
  - **Type handling is very messy**
  - **Does not leverage lightweight threads (smallest unit of computation is the process)**
  - **Strangely absent from the commercial embedded real-time community**
- **Sun's Implementation of MPI**
  - **$375 run-time license per CPU (with educational discount)**
  - **Breaks CPN hardware data circularity (~10% penalty)**

Conclusion

- **Benchmark on eight processor 336 MHz SMP Sun**
  - **Use highly optimized beamforming kernel (single processor)**
  - **Compare performance within several parallel frameworks**
- **Process Network outperforms MPI significantly**
  - **Slowdown on one processor: CPN 16%, MPI 84%**
  - **Speedup on eight processors: CPN 6.5, MPI 3.5**
  - **CPN is 86% faster than MPI on eight processors**
  - **The hybrid CPN / MPI is 18% slower than MPI alone**
  - **CPN is 58% of *peak* performance (2 FLOPS x 336 MHz x 8 CPUs)**
- **If I needed to develop a real-time implementation today, I probably would not use MPI**