Introduction

- **Problem:** Beamforming is computationally intensive
- **Example:** High-resolution sonar beamformers (GFLOPS)
  - Generation 1: expensive custom digital hardware (large state machines)
  - Generation 2: custom integration of programmable digital signal processors on commercial-off-the-shelf hardware (e.g. 120 DSPs in a VME rack)
- **Objective:** Unix workstation beamformers
  - Analysis: evaluate performance of beamforming kernels and systems
  - Modeling: capture parallelism, guarantee determinate bounded execution
  - Implementation: use portable, scalable software to achieve real-time performance on commodity hardware and lower development costs
- **Solution:** Real-time beamforming on workstations
  - Analysis: optimize kernels and profile beamformers to measure scalability
  - Modeling: Process Networks
  - Implementation: Real-time POSIX threads using C++ on symmetric multiprocessor UltraSPARC-II workstation with native signal processing
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
- $\tau_i$: $i$th sensor delay
- $\alpha_i$: $i$th sensor weight

Interpolation Beamforming

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

- Kernel implementation on UltraSPARC-II
  - Highly optimized C++ (loop unrolling and SPARCompiler5.0EA)
  - Currently operating at 60% of peak, which is 2 FLOPs per cycle
### 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 integer dot product
  - Integer-to-float conversion must be performed
  - Output data must be interleaved
- Kernel implementation on UltraSPARC-II
  - Native signal processing with Visual Instruction Set (VIS)
  - Software data prefetching to hide memory latency and keep the pipeline full

### 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
- A program is represented as a directed graph
  - Each node is an independent process
  - Each edge is a one-way queue of data
- Blocking reads, non-blocking writes, infinite queues
- Scheduling for bounded queues is possible [Parks, 1995]
  - Blocking reads and writes
  - Dynamically increase queue capacities to prevent *artificial deadlock*
- Fits the thread model of concurrent programming
Process Network Implementation

• Each node corresponds to a thread
  • Implemented in C++ using POSIX Pthreads
  • Low-overhead, high-performance, scalable
  • Granularity larger than a thread context switch (~10 us)
  • Symmetric multiprocessing operating system dynamically schedules threads
  • Efficient utilization of multiple processors

• Optimize queues for high-throughput signal processing
  • Nodes operate directly on queue memory, avoiding copying
  • Queues use mirroring to keep data contiguous

  ![Queue layout](image)

  • Compensates for the lack of circular address buffers
  • Queues trade-off memory usage for overhead
  • Virtual memory manager maintains data circularity

System Implementation

• Vertical beamformer forms 3 sets of 80 staves from 10 vertical elements per stave

• Each horizontal beamformer forms 61 beams from the 80 staves, using a two-point interpolation filter

• 4 GFLOPS total computation
Performance Results

- 100 trial mean execution time for 2.6 seconds of data
- Sun Ultra Enterprise 4000 with eight 336-MHz UltraSPARC-IIs, 2 Gb RAM, running Solaris 2.6

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time (s)</th>
<th>MFLOPS</th>
<th>Mbytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread pool</td>
<td>3.607</td>
<td>3024.8</td>
<td>832</td>
</tr>
<tr>
<td>process network</td>
<td>3.354</td>
<td>3252.8</td>
<td>654</td>
</tr>
</tbody>
</table>

- Process network is 7% faster than thread pool, overhead is small
- Process network uses 20% less memory with lower latency
- Process network scalability is nearly linear
- Will continue to scale with additional CPUs
- Real-time performance achievable with 12 CPUs

**Conclusion**

- **Third generation beamformers:** *Workstation* hardware
  - Commodity hardware saves development/manufacturing costs
  - Multiprocessor servers, native signal processing
  - Upgradable hardware, Moore’s Law
- **Software model:** *Process Networks*
  - Captures parallelism, guarantees determinate bounded execution
  - Portable, reusable, scalable C++ code
  - High-performance, low overhead POSIX threads
  - Symmetric multiprocessing operating system
- The example 4-GFLOPS 1-Gb 3-D sonar beamforming system does execute in real time using a Sun Ultra Enterprise 4000 server with twelve 336 MHz UltraSPARC-II CPUs with 14.5% to spare

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