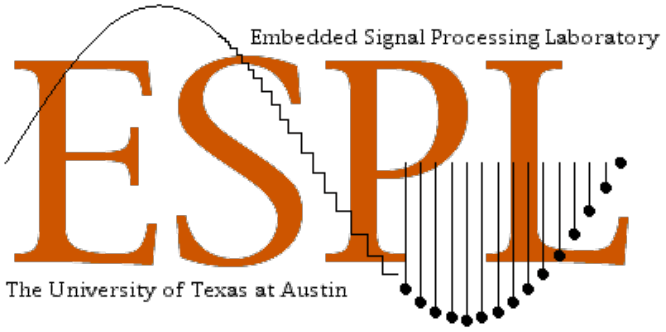
# Zero-copy Queues for Native Signal Processing using the Virtual Memory System

Gregory Allen, Paul Zucknick, Brian Evans
Applied Research Laboratories, and
Dept. of Electrical and Computer Engineering
The University of Texas at Austin




# Introduction

- Modern general-purpose CPUs are capable of substantial digital signal processing performance
  - Tens of GFLOPS with 32-bit floating-point
  - Single-Instruction Multiple-Data (SIMD) instruction sets (SSE3/AltiVec)
  - Highly optimized libraries (VSIPL, Intel MKL)
  - Multiple cores and/or multiple CPUs
- Dubbed "Native Signal Processing" (NSP)

# Bottleneck: Memory

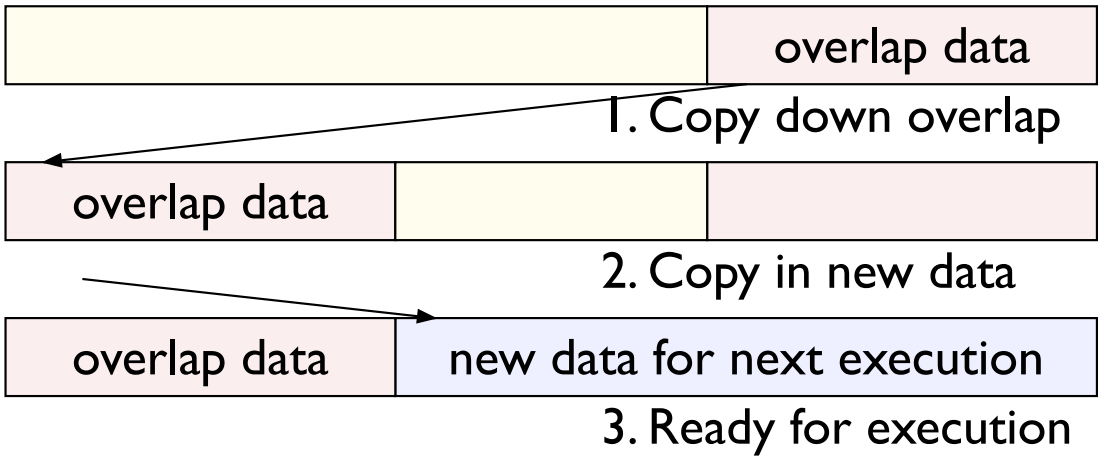
- Major bottleneck in data-dominated applications
- CPUs often waiting on memory -- latency or BW
- Performance depends on organization in memory
- Many algorithms require specific data arrangement
- Copying and re-arranging data is expensive
- Time spent copying is time not doing DSP

# "Sliding Window" Algorithms

- Overlapping, continuous streams of data
- Common in DSP algorithms
  - FIR filters
  - Overlap-and-save FFTs
- "Circular" memory buffers for queues of data
- Hardware DSPs use modulo addressing modes
  - No copying is necessary to maintain circularity

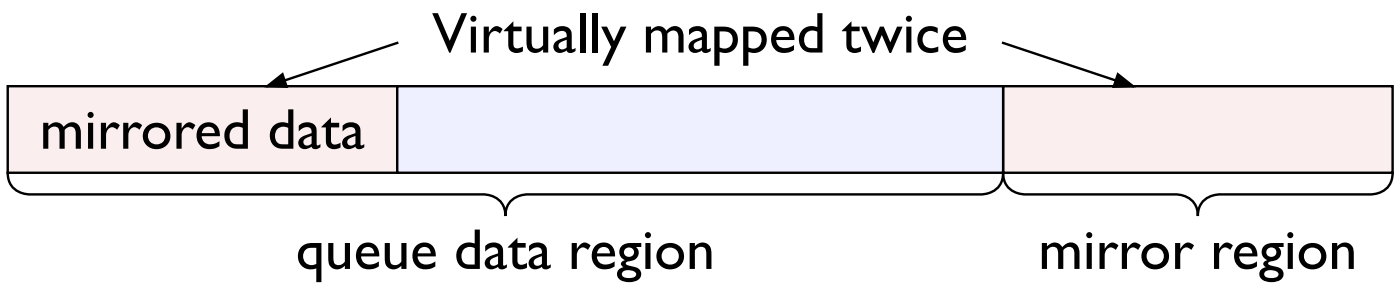
# Circular Queues for NSP

- No modulo addressing on general purpose CPUs
- Optimized libraries are for contiguous blocks
- Copying is necessary to maintain circularity for sliding window algorithms -- undesired overhead



# Circular Queues with VMM

- Virtual Memory Manager maintains circularity
  - Map a physical page to multiple addresses
  - Queue indices wrap upon reaching end
  - Contiguous access to only mirrored length
- No data copying required -- done by VM mappings



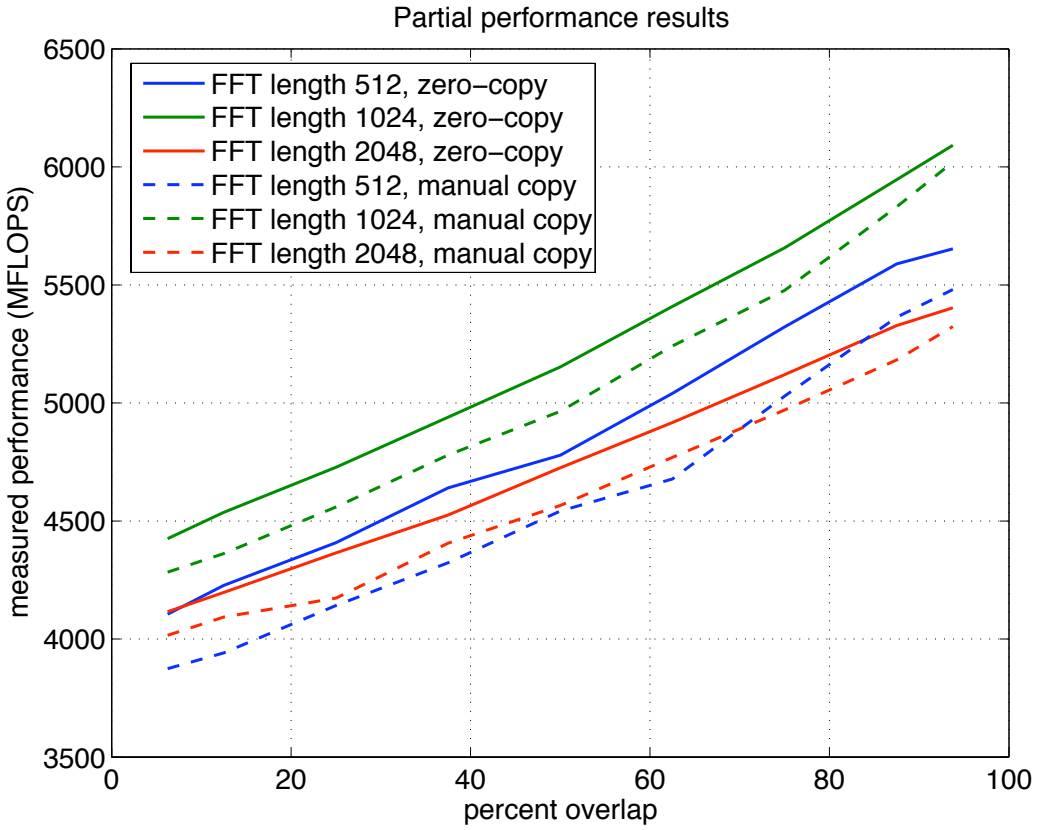
# VM Queue Implementation

- Use POSIX mmap(...) system call:
  1. Create mmap-able object (shm or tmp file)
  2. mmap object (address typically assigned by OS)
  3. mmap object again, adjacent to previous mmap
- Demonstrated to work on: Linux, LinuxPPC, MacOS X, Solaris, AIX
- Source code available at http://www.ece.utexas.edu/~allen/CPN

# Performance Study

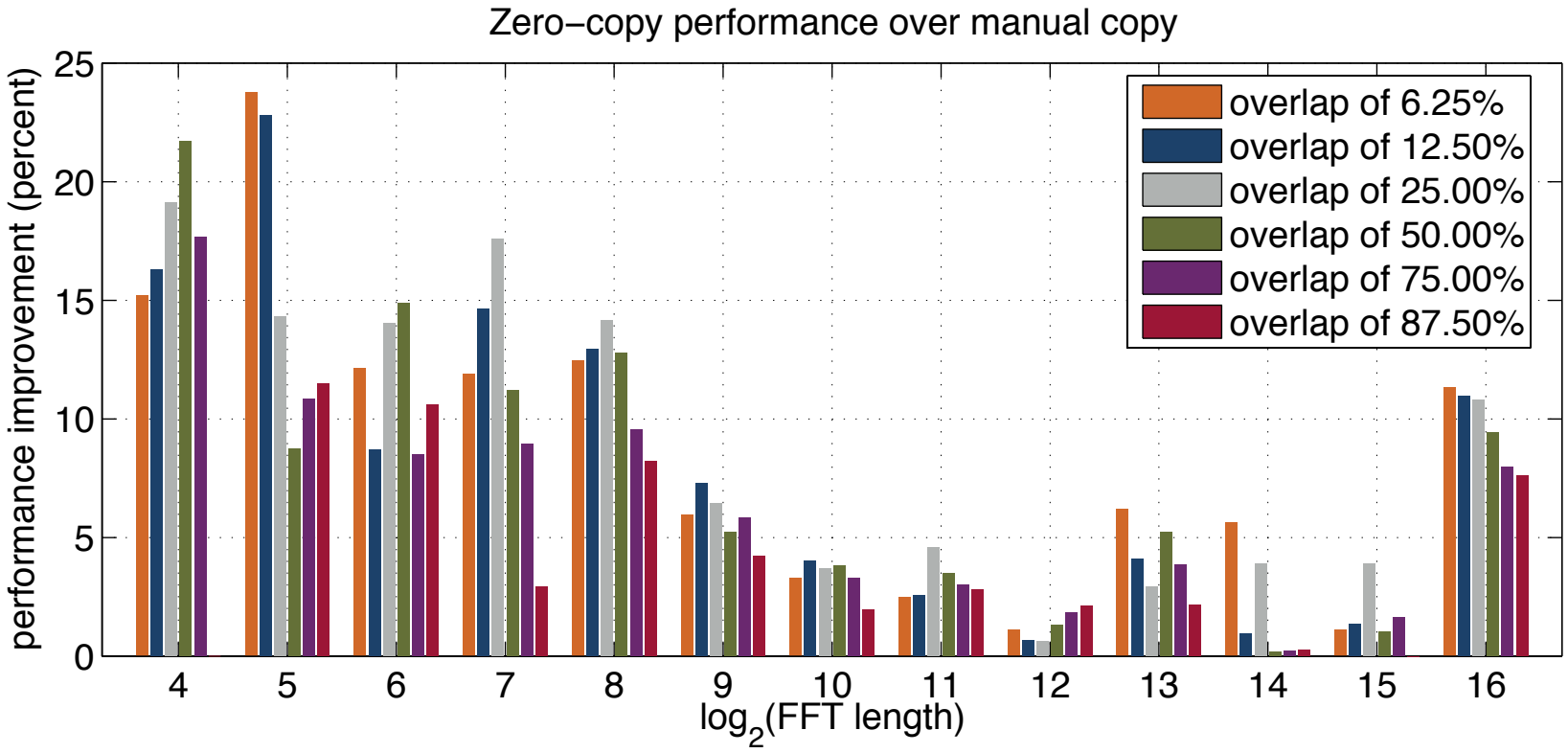
- An FIR filter implemented in the frequency domain:
  - Overlap-and-save FFT using FFTW
  - SIMD complex multiply (AltiVec/SSE2)
  - Inverse FFT using FFTW
- Compare zero-copy queue to overlap copying
  - Vary FFT length over several powers of two
  - Vary overlap size as a percentage of FFT length

# Performance Results




- FFT is 5N log2 N
- 2.5 GHz PPC G5, MacOS X 10.4
- ~2 FLOPs/cycle
- mean of 10 runs
- Zero-copy queue gives visible gain

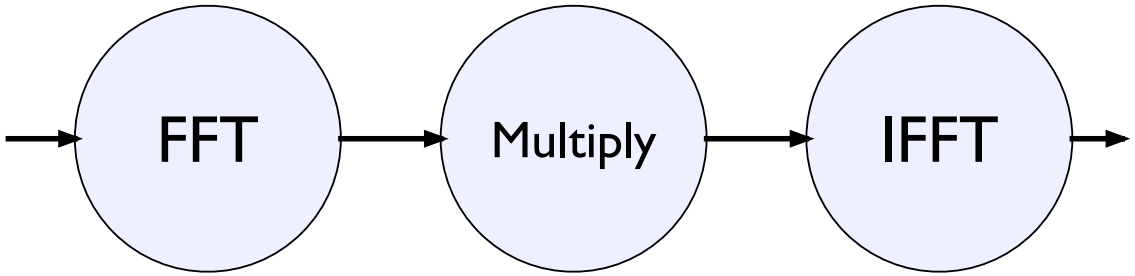
# Performance Results




- Smaller FFTs have substantial gain, tends to decrease
- Smaller workload implies a larger overhead

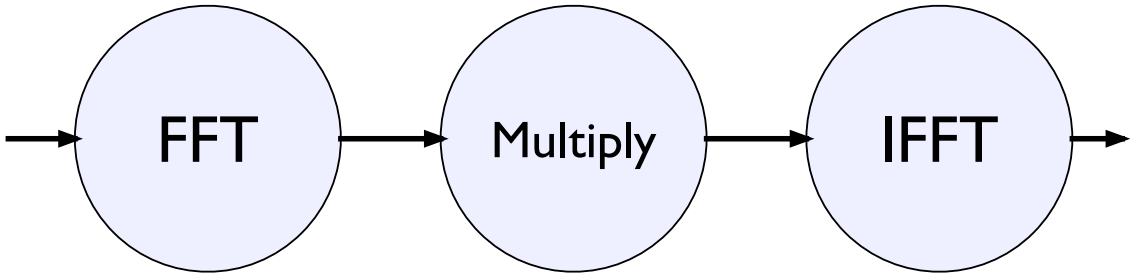
# Process Networks (PN)

- Naturally models parallelism in a system
- A formal model for concurrency [Kahn 74]
  - A dataflow model for parallel processing
  - Arcs are queues, nodes perform processing
  - Mathematically provable concurrency properties



# Computational Process Networks (CPN)

- Adds *firing thresholds* from Computation Graphs
  - Nodes need not perform manual copying
  - Nodes can operate directly from queue memory
  - Decouple computation from communication
- Implementable with zero-copy queues



# Zero-copy Queues & CPN

- Zero-copy queues can significantly reduce overhead normally required for maintaining circularity
  - This reduces system memory bandwidth usage
  - Can increase overall scalability
- Permits smaller granularity processing nodes
  - Increases potential concurrency in the system
  - Increases mapping design space to parallel HW