Real-Time High-Throughput Sonar Beamforming Kernels Using Native Signal Processing and Memory Latency Hiding Techniques

1 ARL The University of Texas at Austin

Gregory E. Allen[1]
Brian L. Evans
Lizy K. John

Department of Electrical and Computer Engineering
The University of Texas at Austin

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

Introduction

- **Sonar beamforming is computationally intensive**
  - **GFLOPS of computation**
  - **100 MB/s of data input/output**
- **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)**
  - **Low production volume (50 units), high development cost**
- **Examine performance of commodity computers**
  - **Native signal processing, multimedia instruction sets**
  - **Memory latency hiding techniques**

Native Signal Processing

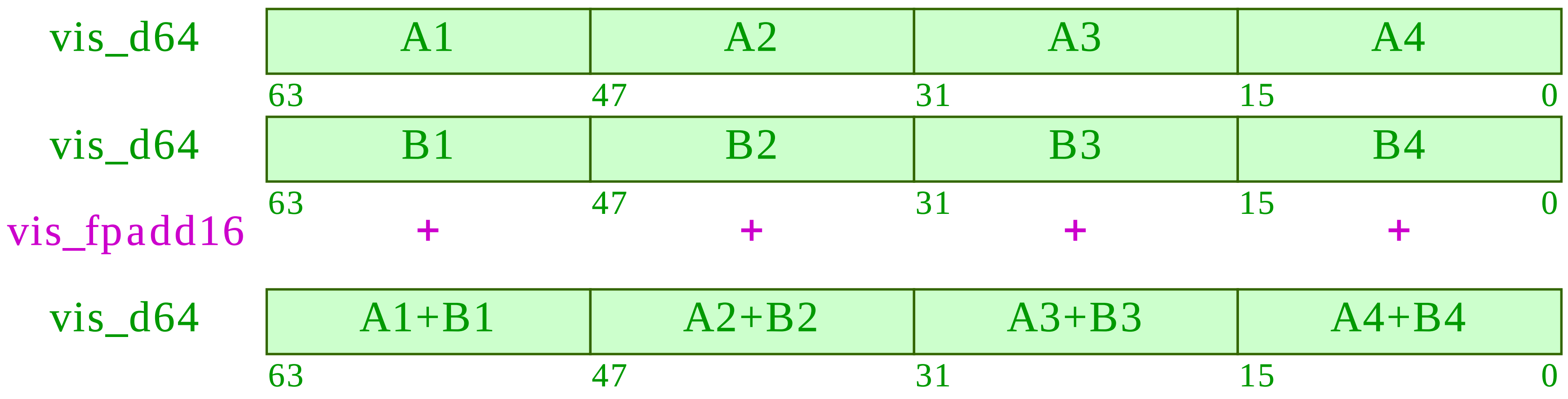
- **Single-cycle multiply-accumulate (MAC) operation**
  - **Vector dot products, digital filters, and correlation**
  - **Missing extended precision accumulation**

$$\sum_{i=1}^{N} \alpha_i x_i$$

- **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**
- **Must hand-code using intrinsics and assembly code**

Visual Instruction Set

- **50 new CPU instructions for UltraSPARC**
  - **Optimized for video and image processing**
  - **Partitioned data types in 32-bit or 64-bit FP registers**
  - **Includes arithmetic and logic, packing and unpacking, alignment and data conversion, etc.**
- **Independent operation on each data cell (SIMD)**



- **Inline function library provided for use from C/C++**

Memory Latency Hiding

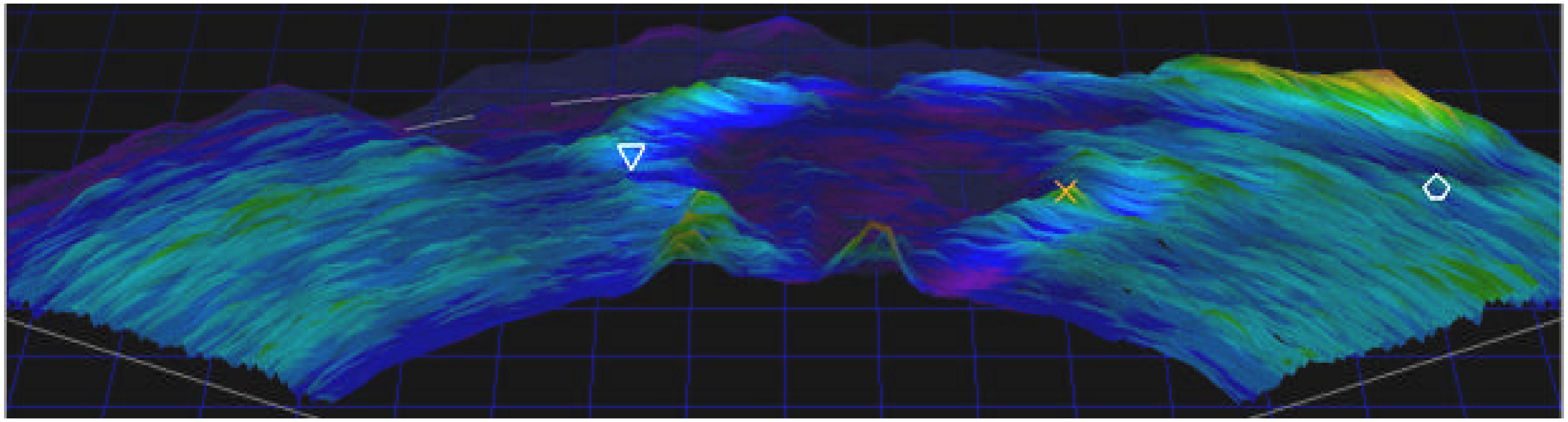
- Fast processor stalls when accessing slow memory
  - Cache memories can help to alleviate this problem
  - High-throughput streams of data amplify this problem
  - Software techniques can reduce the penalty
- Technique: Loop unrolling
  - Enlarges basic block size and reduces looping overhead
  - Can increase the time between data request and consumption
  - Low risk and no overhead, commonly used by compilers
- Technique: Software pipelining
  - Data load and usage overlaped from different loop iterations
  - Increases register usage and lifetimes, hard for compiler

Software Data Prefetching

- **Non-blocking** `prefetch` **CPU instruction**
  - **Issued at some time prior to when data is needed**
  - **Data at effective address is brought into cache**
  - **At a later** `load` **instruction, the data is already cached**
- **Problems: overhead and “prefetch distance”**
  - **Uses extra cache and issues extra instructions**
  - **Prefetch too far ahead: excessive cache usage, spillage**
  - **Not far enough ahead: stall at** `load` **instruction**
- **Can be generated by a compiler**
- **Implemented in the UltraSPARC-II CPU**

Sonar Beamforming

- We evaluate two key kernels for 3-D beamforming



- Typically the computational bottleneck in sonar
- High throughput streams of data
- Goal: best performance using any means

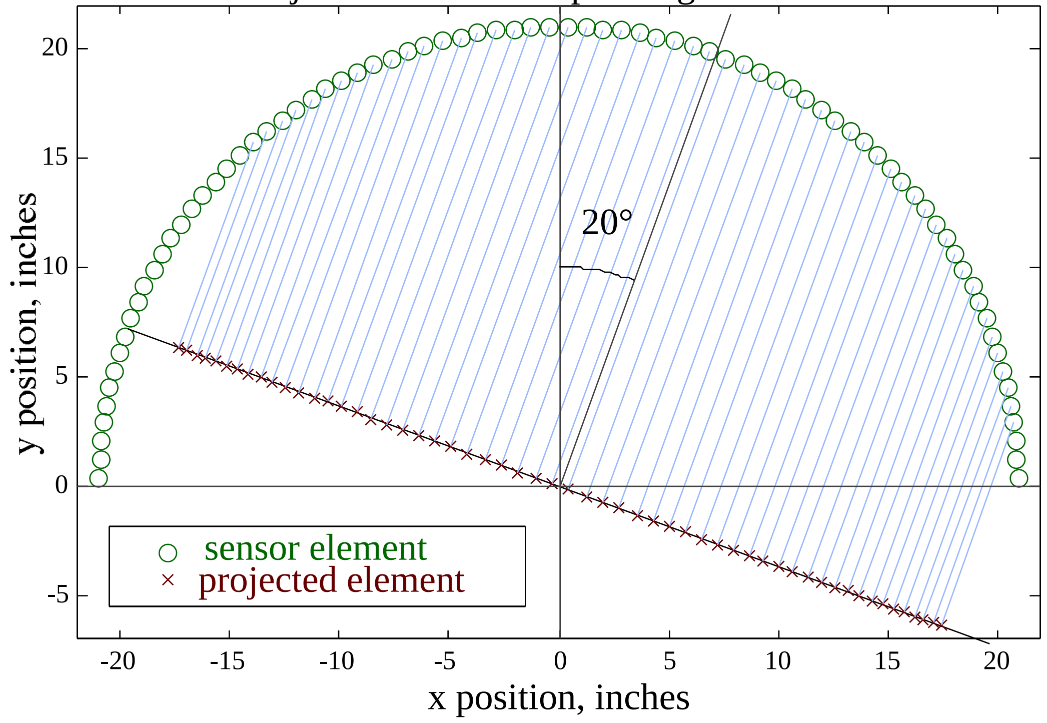
Time-Domain Beamforming

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

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

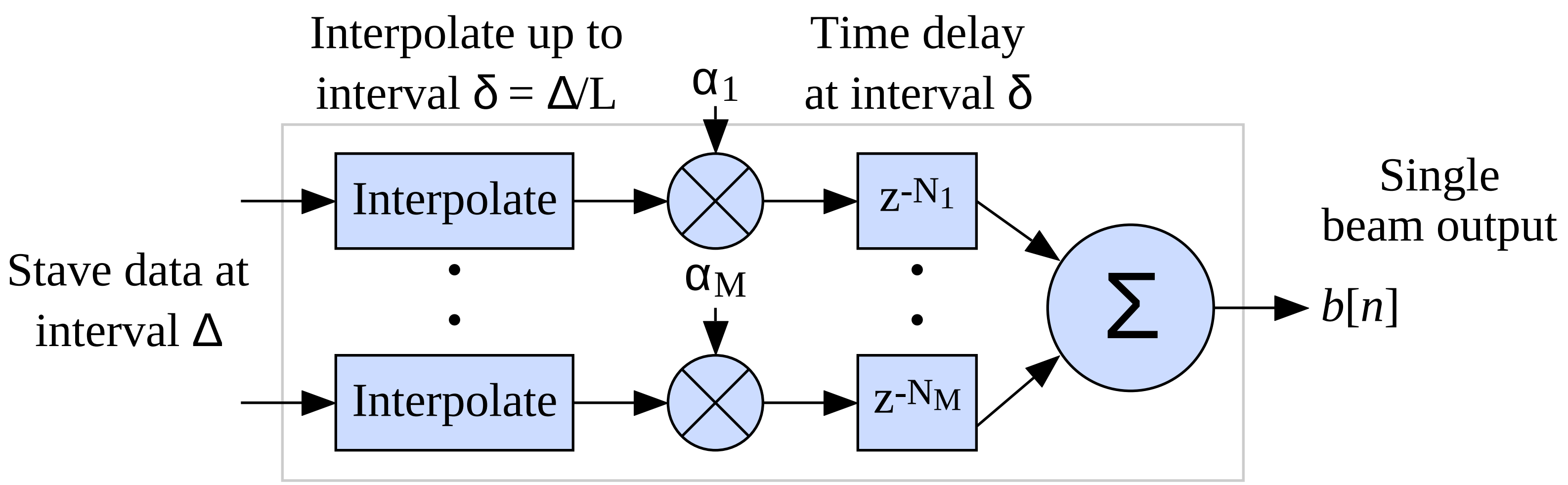
$b(t)$	beam output
$x_i(t)$	ith sensor output
τ_i	ith sensor delay
α_i	ith sensor weight

Projection for a beam pointing 20° off axis



Horizontal Beamformer

- **Sample at just above the Nyquist rate, interpolate to obtain desired time delay resolution**

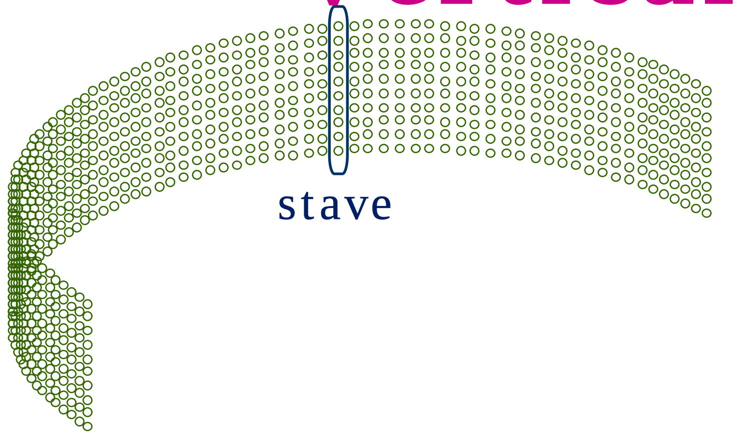


Interpolate up to interval $\delta = \Delta/L$
Time delay at interval δ
α_1
α_M
Interpolate
Interpolate
z^{-N_1}
z^{-N_M}
Σ
Stave data at interval Δ
Single beam output
$b[n]$

Digital Interpolation Beamformer

- **Modeled as a sparse FIR filter**
  - **Forming 61 beams from 80 elements with 2-point interpolation**
  - **3000 index lookup plus 6000 floating-point MACs per sample**
  - **At each sample: 12 Kbytes of data, coefficient size of 36 Kbytes**

Vertical Beamformer



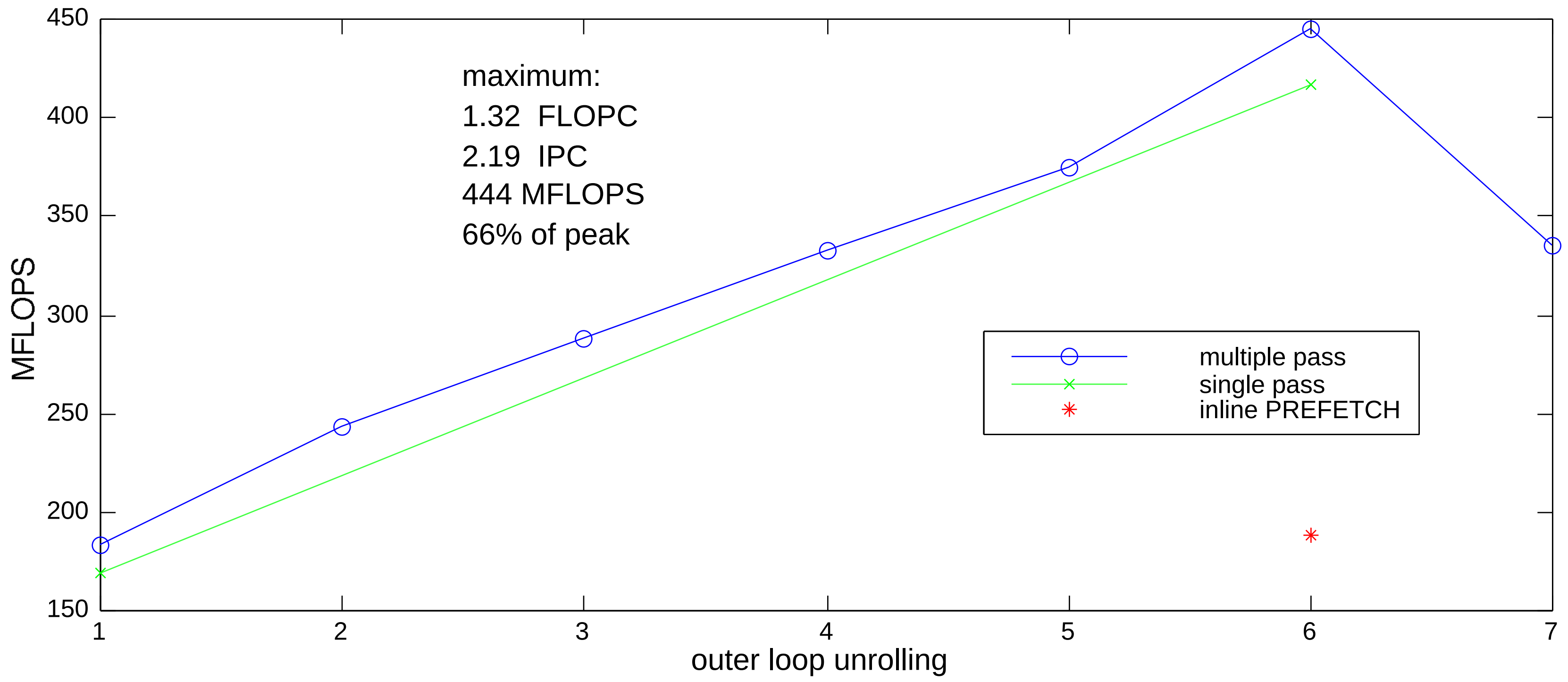
Multiple vertical transducers for every horizontal position

- **Vertical columns combined into 3 stave outputs**
  - **Multiple dot products (30 MACs per stave per sample)**
  - **Convert integer to floating-point for following stages**
- **Ideal candidate for the Visual Instruction Set (VIS)**
  - **Use integer dot products (16x16-bit multiply, 32-bit add)**
  - **Highest precision (and slowest) VIS mode**
  - **Coefficients must be scaled for best dynamic range**

Tools Utilized

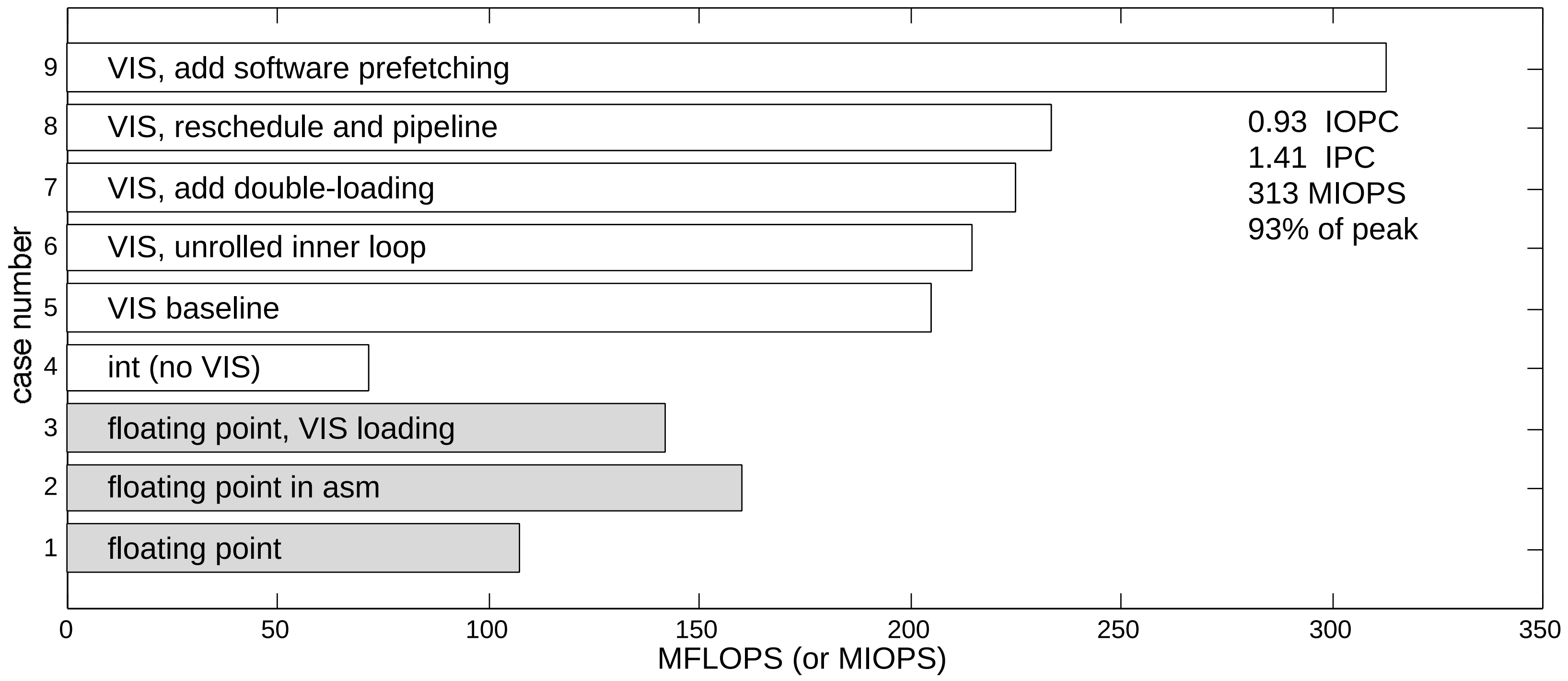
- **Sun's SPARCompiler5.0**
  - **Automated** `prefetch` **instruction generation?**
  - **Inline assembly macros for VIS instructions**
  - **Wrote assembly macros for** `prefetch` **and** `fitos` **instructions**
- **Shade:** `pficount` **(prefetch instruction counter)**
- **INCAS (It's a Nearly Cycle-Accurate Simulator)**
- **perf-monitor: hardware performance counters**
- **Benchmarks on a 336 MHz UltraSPARC-II**

Horizontal Kernel Performance



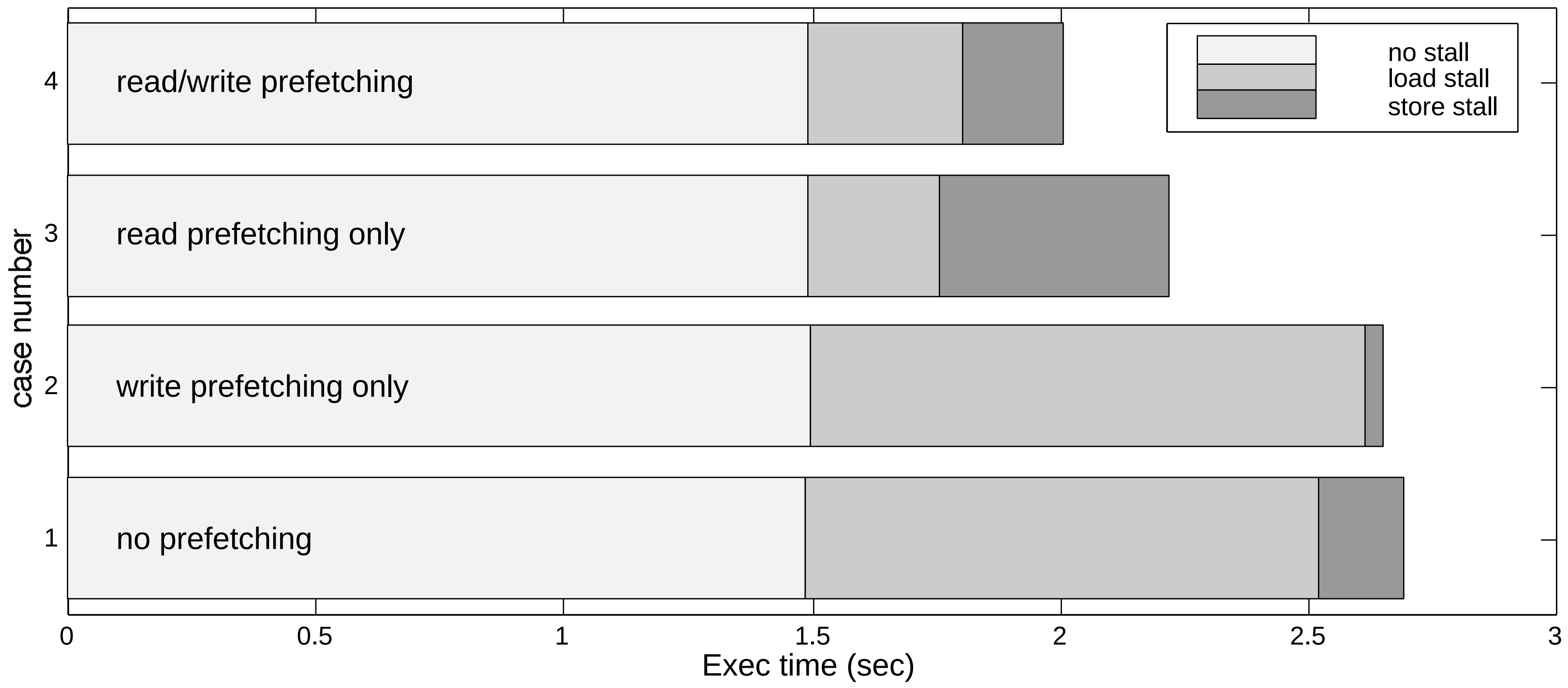
- **Hand loop unrolling gives speedup of 2.4**
- **Multiple passes improve cache usage (93% / 97%)**
- **Inline PREFETCH “breaks” compiler optimization**

Vertical Kernel Performance



- **VIS offers a 46% boost over floating-point**
- **Software prefetching gives an additional 34%**
- **104 MB/s data input, 62.7 MB/s data output**

Vertical Prefetch Statistics



- **Breakdown of execution time**
- **Execution cycles (no stall) constant across trials**
- **Internal cache statistics do not change**

Conclusion

- **Beamforming kernel results:**
  - **Horizontal beamformer kernel: 444 MFLOPS, 66% of peak**
  - **Vertical beamformer kernel: 313 MFLOPS, 93% of peak**
  - **Loop unrolling: 2.4 speedup in horizontal kernel**
  - **VIS: 1.46 speedup in vertical kernel**
  - **prefetching: 1.34 speedup in vertical kernel**
- **Near-peak performance can be achieved, but**
  - **Kernel optimization is difficult and time consuming**
  - **Compiler did not generate** `prefetch` **instructions**
- **For high-throughput real-time signal processing, general purpose CPUs can be an attractive target**