Evaluating MMX Technology Using DSP and Multimedia Applications

Ravi Bhargava *

Lizy K. John *

Brian L. Evans

Ramesh Radhakrishnan *

November 22, 1999

The University of Texas at Austin

Department of Electrical and Computer Engineering

*** Laboratory of Computer Architecture**

Evaluating MMX Technology Using DSP and Multimedia Applications

This talk is a condensed version of a presentation given at:

The 31st International Symposium on Microarchitecture (MICRO-31)

Dallas, Texas

November 30, 1998

http://www.ece.utexas.edu/~ravib/mmxdsp/

MMX

- 57 New assembly instructions
- 64-bit registers
- Aliased to FP registers
- EMMS Instruction
- No compiler support

MMX

- ❒ 8, 16, 32, 64-bit fixed-point data
- ❒ Packing, unpacking of data
- ❒ Packed moves
- ❒ 16-bit multiply-accumulate
- ❒ Saturation arithmetic

Motivation

- Independent evaluation of MMX
- How much speedup is possible?
- What tradeoffs are involved?
  - Time, complexity, performance, precision
- Characterization of MMX workloads
  - Instruction mix, memory accesses, etc.

Kernels

- ❒ Finite Impulse Response Filter
  - ❒ Speech, general filtering
- ❒ Fast Fourier Transform
  - ❒ MPEG, spectral analysis
- ❒ Matrix &Vector Multiplication
  - ❒ Image processing
- ❒ Infinite Impulse Response Filter
  - ❒ Audio, LPC

Applications

- JPEG Image Compression
  - Bitmap Image to JPEG Image
  - 2D DCT
- G.722 Speech Encoding
  - Compression, Encoding of Speech
  - ADPCM
- Image Processing
  - Uniform Color Manipulation
  - Vector Arithmetic
- Doppler Radar Processing
  - Vector Arithmetic, FFT

Methodology

- Adjust non-MMX benchmark
  - DSP environment
- Create MMX version
  - Setup like non-MMX
  - Use Intel Assembly Libraries
- Microsoft Visual C++ 5.0
- Simulate with VTune 2.5.1

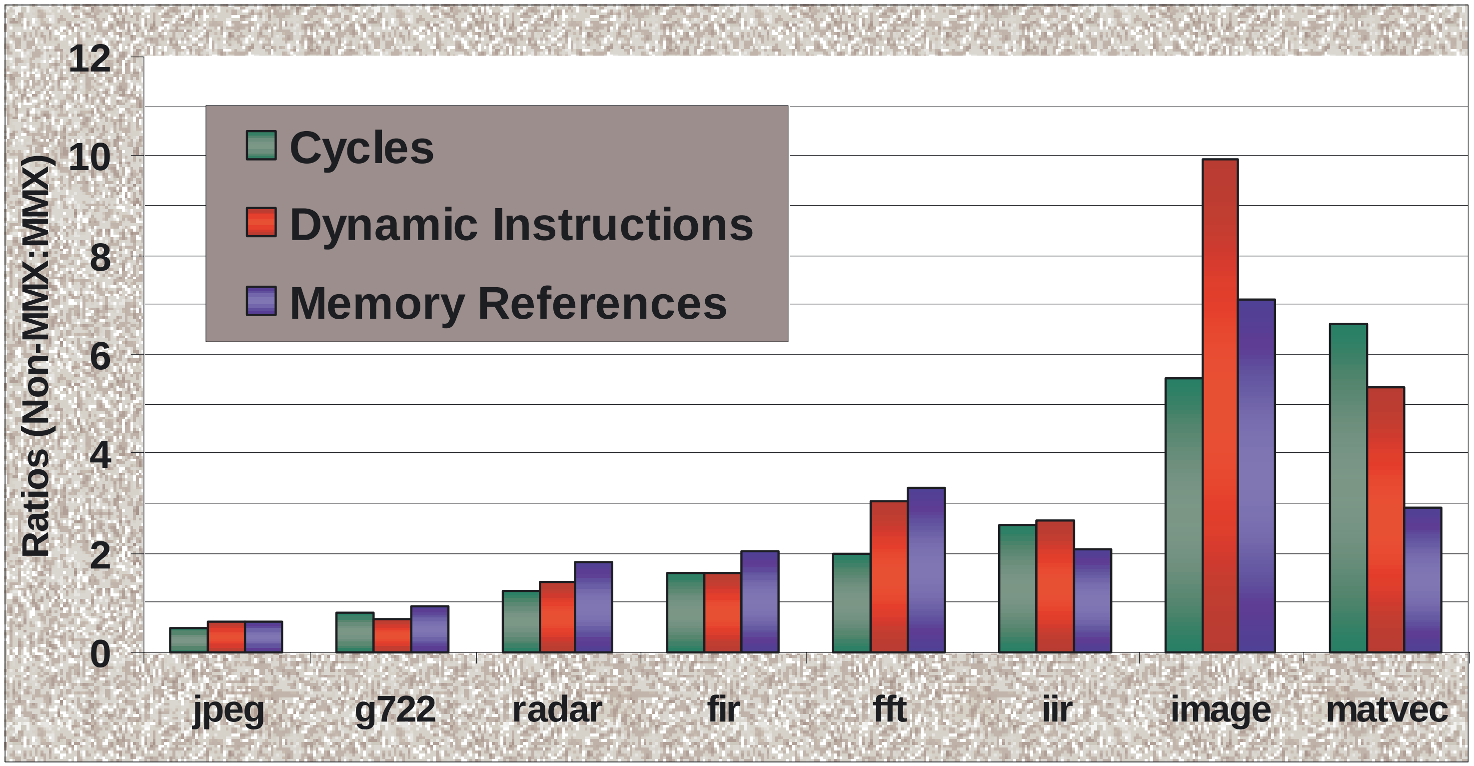
Creating MMX Benchmarks

- ***Not just function swapping***
- Different input data types
  - Fixed-point versus floating-point
  - 16-bit versus 32-bit
- Reordering of data
  - Ex: Arrangement of filter coefficients
  - Row-order versus column-order

VTune 2.5.1

- Intel performance profiling tool
  - Designed for “hot spots”
- Simulate sections of code
  - Pentium with MMX
  - CPU penalties
  - Instruction mix
  - Library calls
- Hardware performance counters

Overall Results

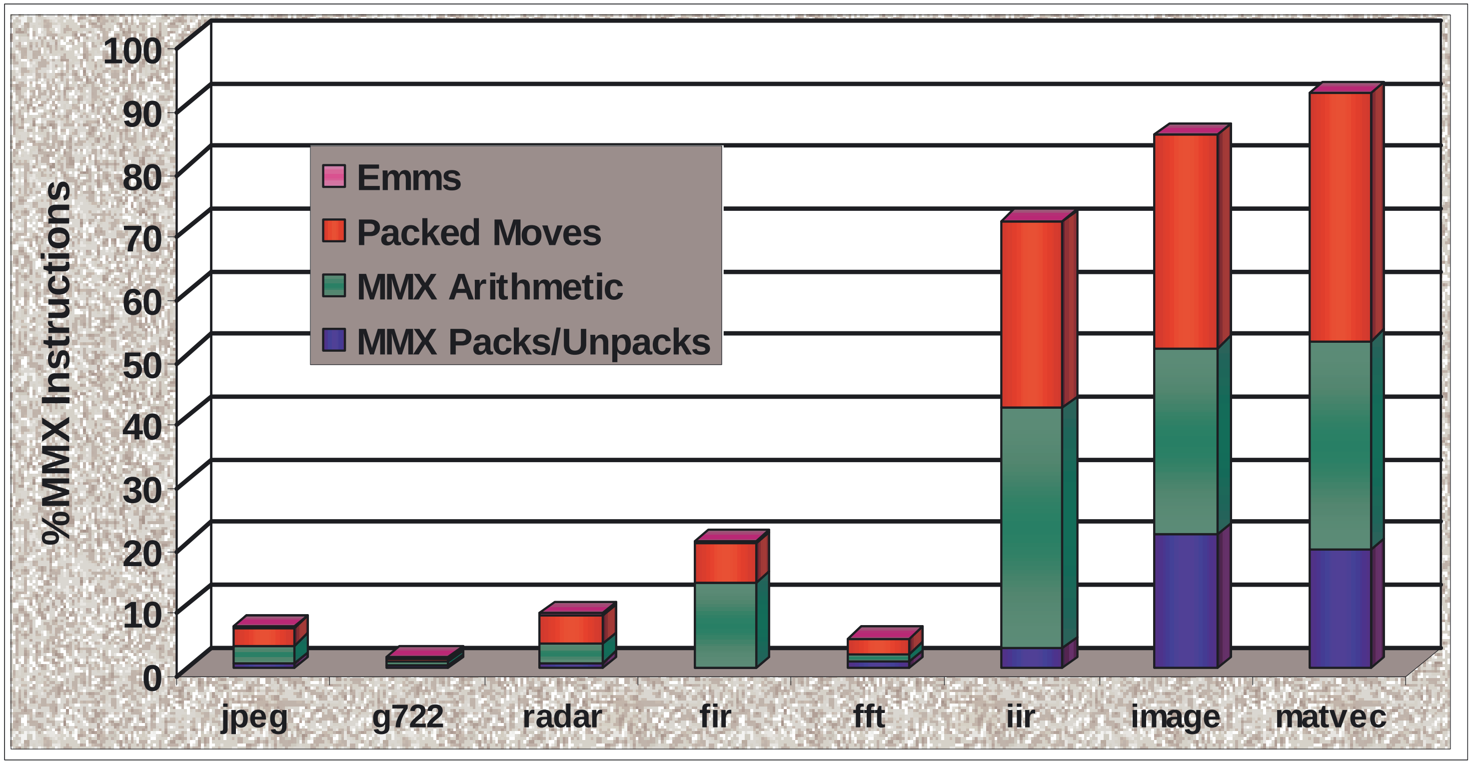


Ratio of non-MMX to MMX Programs

Overall Results

- JPEG and G722 show slowdowns
- Superlinear speedup in MatVec
  - 16-bit data, 6.6X speedup
  - Free unrolling
- MMX related overhead
  - FIR, Radar, JPEG, G722
- MMX multiplication
  - Fewer cycles
  - Requires unpacking

MMX Instruction Mix

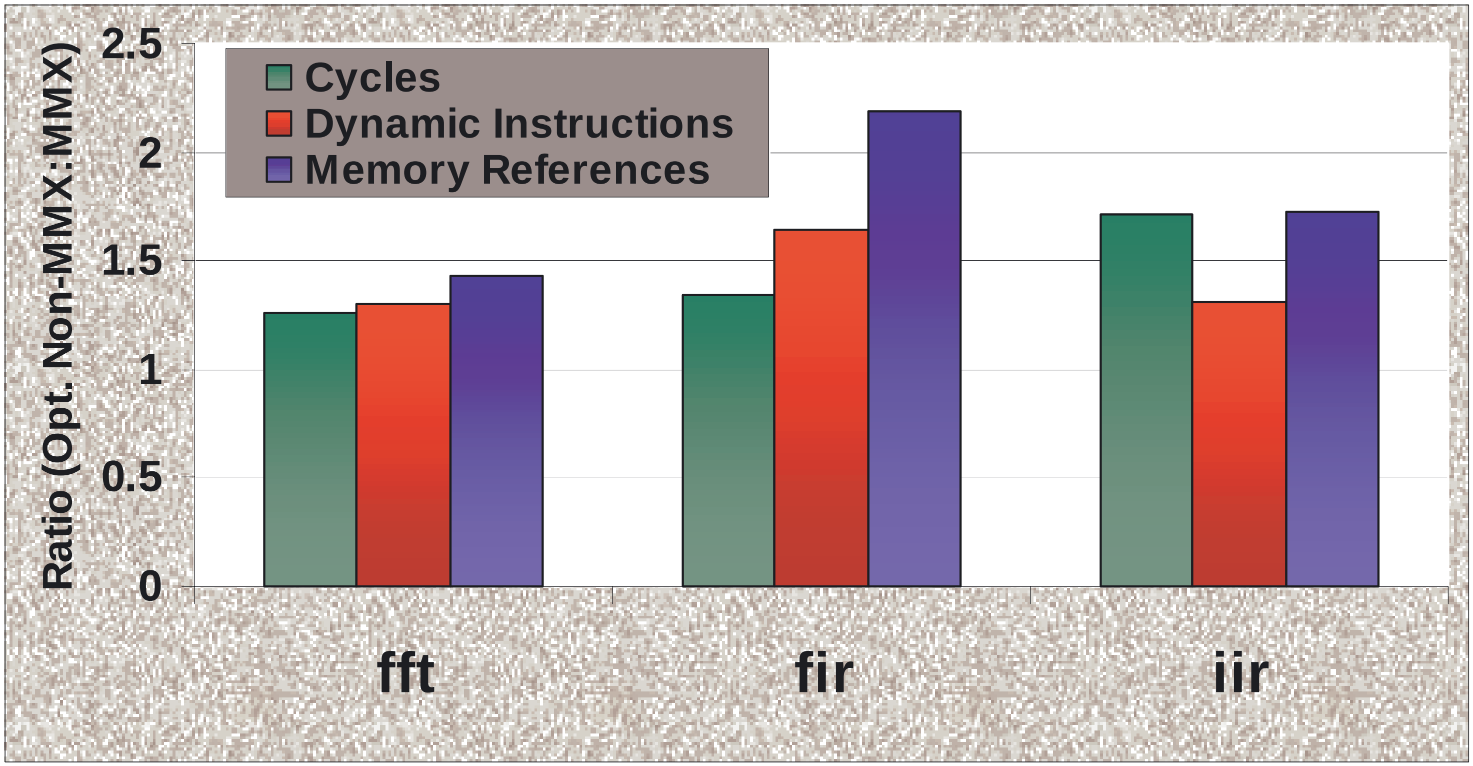


% MMX Instructions and MMX Instruction Mix.
Speedup increasing from left to right

MMX Instruction Mix

- Input set size
  - Small: FIR, Radar, G722, JPEG
  - Large: IIR, Image, MatVec, FFT
  - Affects MMX %, speedup
- "Automatic" Packing
- Less than 50% MMX arithmetic
- FFT
  - Converts to FP
  - Old version: 40% MMX, less speedup

Versus Optimized Code



Ratio of Non-MMX Assembly to MMX

Closer Look at JPEG

- Non-MMX version 1.98X faster
- ***But...*** inserted MMX code 1.6X faster
- Function call overhead
  - 8.8X more in MMX version
- MMX Maintenance Instructions
  - Accounting for precision
  - Non-sequential data accesses

Some Problems

- ***Slowdown possible***
  - JPEG and G722
- Parallel, contiguous data
  - Hard to find
- Precision
  - Obtainable at a price
- Library function call overhead
  - Hand-coded assembly, inlining

Summary of Results

- Speedup available with libraries
  - Kernels: 1.25 to 6.6
  - Applications: 1.21 to 5.5
  - Versus optimized FP: 1.25 to 1.71
- General Characteristics of MMX
  - More static instructions used
  - Fewer dynamic instructions
  - Fewer memory references
  - Less than 50% of MMX is arithmetic

This concludes this portion of the talk.

The following slides provide further information on: methodology, benchmarks, results, and additional work.

Unreal 1.0

- Doom-like game
- Command-line MMX switch
- Hardware Performance Counters
- 48% MMX Instructions
- Real-time. What is speedup?
- 1.34X more frame/second
- Same trends as benchmarks

DSP-like Environment

- Focus on “Important” Code
- Buffer Inputs and Outputs
- No OS Effects Measured
- Real-time Atmosphere

Intel Assembly Libraries

- Some functions use MMX
  - 8-bit and 16-bit data
  - Scale factors
  - Vector inputs
  - Library-specific structures
- Signal Processing Library 4.0
- Recognition Primitives Library 3.1
- Image Processing Library 2.0

Precision

- JPEG
  - Non-MMX SNR: 31.05 dB
  - MMX SNR: 31.04 dB
- Image: No Change
- G722
  - Non-MMX SNR: 5.46 dB
  - MMX SNR: 5.18 dB
- Doppler Radar
  - Less than 1%

An Example: JPEG

- Profiled Program
  - 2D DCT
  - Quantization
  - Color Conversion
  - 74% of execution time
- Small Block Size
  - 8x8 blocks of pixels

JPEG: Inserting MMX

- 2D DCT
  - Library only has 1D DCT
  - Data in different order
- Quantization
  - Not enough data parallelism
- Color conversion
  - Create and fill vectors

FIR Filter

- Finite Impulse Response Filter
- Moving averages filter
- Process one input at a time
- Non-MMX: 32-bit FP
- MMX: 16-bit fixed-point
- Filter length is 35

FFT

- Fast Fourier Transform
- Computes discrete Fourier Transform
- 4096-point
- In-place
- Whole FFT to MMX function
- Non-MMX: 32-bit FP
- MMX: 16-bit fixed-point

MatVec

- Matrix & Vector Multiplication
  - 512x512 matrix times 512-entry vector
  - Dot product of two 512-entry vectors
- Both versions: 16-bit data

IIR

- Infinite Impulse Response Filter
  - Butterworth coefficients
  - Direct form, Bandpass
  - Filter length of 8, 17 coefficients
- Requires high precision
  - Feedback
  - Our versions unstable

Doppler Radar Processing

- Subtract complex echo signals
- Removing stationary targets
- Estimates power spectrum
- Dominant frequency from peak of FFT
- 16-point, in-place FFT

G.722 Speech Encoding

- ❒ Input signal: 16-bit, 16 kHz
- ❒ Output signal: 8-bit, 8 kHz
- ❒ 6 kb speech file