Programmable VLIW and SIMD architectures for DSP and Multimedia Applications

Deepu Talla
Overview

• Introduction/Motivation
• Methodology
• Tools
• Fallacies and Pitfalls
• Benchmarks
• Results
• Future work and Conclusions
Introduction/Motivation

- VLIW processors exploit instruction parallelism while SIMD processors exploit data parallelism
- Over 90% of workloads in future expected to be multimedia and DSP oriented
- To my knowledge no quantitative work has been done in comparing commodity VLIW and SIMD processors
  - **C6x** is a **VLIW** DSP processor and **Pentium II with MMX** is a **SIMD** processor
Methodology

• Measure the execution times of benchmarks on C6x and Pentium II (MMX)

• Use execution time of Pentium II without MMX code as baseline

• Each benchmark will have three versions
  › Pentium II code without MMX
  › Pentium II code with MMX
  › C6x code
Tools

- **C6x**
  - Stand-alone simulator for execution cycle count
  - Optimizing compiler, simulator and debugger

- **Pentium II**
  - Performance counters for execution statistics
  - Intel C/C++ compiler
  - Vtune for static code analysis
Fallacies and Pitfalls

• Two completely different processors are being evaluated; so how is an equivalent playground/environment being created?
  › First of all the memory hierarchies of both processors are completely different -> Pentium II with two layers of caches and DRAM, C6x with small L1 and SRAM
  › To remove effects of memory latencies, data sets in both cases have been made to fit on chip (translates to fastest memory of each processor)
  › Each benchmark is run multiple times over the pre-loaded data set
• What is the importance of aggressive optimization of code? (particularly for DSPs)
  › Using ordinary C code is not the best step -> for a simple dot-product kernel C code is twice slower, and for the DCT it is an order of magnitude slower than optimized assembly

• Compilers can generate MMX code
  › Sure, but only for marketing people
  › It has been mentioned that compiler technology takes 5 to 10 years to catch up to an architecture
Benchmarks - Kernels

- Dot Product
  - Filtering, Matrix-Vector, Alpha Blending
- Autocorrelation
  - Filtering applications
- FIR (Finite Impulse Response) filter
Benchmarks - Applications

• Audio-effects
  › Echo effects, Signal mixing and Filtering
• G.711 standard
  › A-law to u-law and u-law to A-law
• ADPCM
  › 16-bit to 4-bit compression
Creation of Benchmarks

• Dot Product and Autocorrelation
  › Hand-coded baseline and obtained MMX and VLIW code from libraries

• FIR (Finite Impulse Response) filter
  › Hand-coded baseline and MMX code and VLIW code was obtained from libraries
  › MMX code has been tweaked to get maximum performance (needs four copies of filter coefficients !)
Creation of Benchmarks

• Audio-effects and G.711
  › Hand-coded all versions of the benchmarks

• ADPCM
  › MMX could not be used here due to the fact that computation on each data sample involved result of computation on previous data sample

All versions of benchmarks produce same results
Results - Kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>PII-NoMMX</th>
<th>PII-MMX (SIMD)</th>
<th>C6x (VLIW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot Product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Autocorrelation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The graph shows the speedup for different kernels under various conditions.
Results - Applications

- G.711
- ADPCM
- AudioEffects

Speedup

- PII-NoMMX
- PII-MMX (SIMD)
- C6x (VLIW)
Effect of MMX on Clock cycles per instruction

- Dot Product
- Autocorrelation
- FIR
- AudioEffects
- ADPCM
- G.711

![Graph showing the effect of MMX on clock cycles per instruction across different applications. The graph compares CPI (Clock Period) and the percentage of MMX instructions.]
Branch Characteristics

- Branch Characteristic
- Dot Product
- Autocorrelation
- FIR
- AudioEffects
- ADPCM
- G.711

- No-MMX
- MMX
- % MMX instructions

% of MMX instructions

Department of ECE
Laboratory for Computer Architecture
Micro-ops per instruction

![Bar chart showing UOPS per instruction for different benchmarks under No-MMX and MMX conditions. The benchmarks include Dot Product, Autocorrelation, FIR, AudioEffects, ADPCM, and G.711. The chart compares the performance of No-MMX with MMX across these benchmarks.]
Breakup of MMX instructions

- Dot Product
- Autocorrelation
- FIR
- AudioEffects
- G.711

Categories:
- Arithmetic
- Logical
- Multiply
- Shift
- Others
Future Work (by end of May)

- Create more applications
  - In fact applications like Doppler radar and ECG compression were tried, but C6x versions crashed -> need to significantly alter source code
- Evaluate the new floating-point streaming SIMD versus the C67x (waiting for a Pentium III processor to arrive in our lab!)
- Measure other statistics relevant to MMX technology (not related to this project per se, but for computer architects)
Conclusions

- Both SIMD and VLIW techniques provide significant performance improvement over baseline code.
- Compilers are very crucial for efficient code development.
  - Benefit of C6x in applications is not fully achieved unless application is hand-coded (this involves month’s of development time).
  - SIMD compilers are hardly existent, but hand-coding is comparatively easy (with intrinsics).