Resumo: *MP3 Optimization Exploiting Processor Architecture And Using Better Algorithms*

Mancia Anguita; J Manuel Martinez. MP3 Optimization Exploiting Processor Architecture And Using Better Algorithms; IEEE Micro May/June 2005 Vol 25 N 3; pp 81-91

Autor do resumo: *Ricardo Massahiro Nishihara RA 936161*

Introdução

O tempo de execução de toda aplicação é impactado por fatores como: arquitetura e freqüência de relógio do processor utilizado, complexidade computacional dos algoritmos utilizados no programa, escolha do compilador e opções de compilação, e capacidade do programador para explorar (de maneira explícita ou implicita) aspectos da arquitetura do processador utilizado. O artigo tratado neste resumo procura através de experimentos quantificar a influência de tais fatores na implementação de um decodificador de áudio MP3. O artigo inicia com uma breve descrição do processamento realizado por um decodificador MP3, descreve implementações de decodificador realizadas pelos autores com diferentes níveis e tipos de otimizações (considerando principalmente o uso pelo programador de funcionalidades da arquitetura do processador e de algoritmos mais eficientes), descreve a metodologia de teste usada para comparação das implementações (otimizações), e finalmente mostra e discute resultados destes experimentos. Este resumo segue esta mesma organização.

Descrição do decodificador MP3 (MPEG Audio Layer 3)

Uma descrição dos estágios básicos de processamento de um decodificador MP3 (descrito em detalhes pelo padrão ISO/IEC 11172-3) é apresentada no artigo para um melhor entendimento dos requisitos demandados. Conforme mostra a Figura 1 do artigo, o decodificador é composto pelos estágios:

- *Pré-processamento* - esse estágio encontra as estruturas de quadro dentro do bitstream MP3, e extrai destes quadros: dados referentes ao áudio comprimido, e informações auxiliares necessárias ao processo de decodificação, tais como as tabelas de Huffman e os fatores de escala.
- *Decodificação do código de Huffman* - o processo de codificação de Huffman é um esquema de codificação sem perdas que gera palavras-código de tamanho variável (códigos de Huffman) a partir de símbolos de entrada. Este mapeamento palavras-código versus símbolos de entrada é baseado na distribuição estátistica da seqüência de simbolos de entrada. Procura-se associar (codificar) símbolos de entrada que ocorrem mais freqüentemente a palavras-código mais curtas, e palavras-códigos mais longas a símbolos de entrada menos freqüentes, com o intuito de comprimir dados. O processo de decodificação é baseado na consulta a tabelas de Huffman que mapeiam palavras-código a símbolos. A informação auxiliar extraída no estágio de pré-processamento especifica qual tabela deve ser usada no quadro corrente. O padrão MPEG Audio Layer III define 17 tabelas, sendo que a palavra-código mais longa tem comprimento de 19 bits. A utilização de um método de look-up direto pelo decodificador envolveria tabelas muito grandes e por esta razão uma representação mais compacta traduz cada tabela de Huffman em uma estrutura de consulta em árvore. A palavra código inteira é recuperada quando uma folha da árvore é atingida, sendo que a cada folha está associada a um símbolo de saída do decodificador de Huffman (neste caso, o valor de um coeficiente em freqüência escalonado).
- *Requantização* - Este estágio reconstrói os coeficientes em freqüência originais (ou seja em sua escala original) a partir dos coeficientes escalonados recuperados pelo decodificador Huffman e dos fatores de escala recuperados no estágio de pré-processamento.
- *Reordenamento* - O codificador realiza um reordenamento de blocos curtos para aumnetar a eficiência da codificação de Huffman, o decoficador então tem de realizar o processo inverso.
- *Decodificação estéreo* – Para explorar redundância entre canais estéreo o codificador pode codificar as amostras em MS/Stereo e Intensity Stereo e decodificador tem de extrair os canais independentes.
- *IMDCT (Inverse Modified Discrete Cossine Transform)* - Este estágio é o responsável pela execução da transformada inversa do cosseno discreto modificada, operação dual à MDCT (*Modified Discrete Cossine Transform*) realizada no codificador. Esta transformada inversa é aplicada sobre blocos subbanda de 18 coeficientes de freqüência. A Figura 3 do artigo, ilustra de maneira esquemática este processamento.
- *Síntese do filtro polifásico* - Este estágio é responsável pela execução da síntese do banco de filtros polifáficos, operação dual da etapa de análise do banco de filtro polifásico realizado no codificador. A Figura 4 do artigo ilustra de maneira esquemática este processamento. Trata-se do estágio do decodificador com maior demanda computacional. As operações de MDCT/IDCT e em combinação com a análise/síntese do banco de filtros polifásicos são responsáveis pelo mapeamento freqüência versus tempo do codec MP3.

Implementações MP3 usadas nos experimentos

Os autores do artigo implementaram várias versões do decodificador MP3, cobrindo diferentes níveis e tipos de otimização, as quais são resumidas a seguir:

- *Standard:* Esta versão foi implementada seguindo a documentação do padrão MPEG, e utiliza somente as tabelas definidas por este padrão.
- *Basic:* Melhoria da versão *standard* principalmente através do uso de funções da biblioteca padrão do compilador, que se valem de funcionalidades da arquitetura do processador utilizado. Dentre exemplos de tais otimizações são citadas: troca de divisões em ponto flutuante por multiplicações, e de algumas multiplicações inteiras por deslocamentos; troca de funções de biblioteca computacionalmente intensas como cossenos e potências por tabelas; troca de código de alto nível feito pelo programador por funções de biblioteca que utilizam instruções específicas do processador; uso de *loop unrolling* para alguns loops.
- *SIMD:* Melhoria da versão *basic* através do uso de instruções SIMD do processador. Instruções SIMD executam a mesma operação sobre vários dados em paralelo, o que pode ser utilizado de maneira mais eficiente pelo programador para aumentar o desempenho de operações matriciais e vetoriais. O decodificador MP3 é fortemente baseado em operações vetoriais de modo sua implementação pode se beneficiar destas instruções SIMD. Para esta versão foram desenvolvidas implementações (rotinas inline assembly) para os seguintes estágios do decodificador MP3: requantização, decodificação estéreo, IMDCT, e síntese de filtros polifásicos. Instruções SIMD também foram utilizadas para melhorias para inicialização e transferência de blocos de memória.
- *Algorithmic:* Melhoria da versão *basic* através do uso de melhores algoritmos para os seguintes estágios do decodificador MP3: síntese de filtros polifásicos (método de Konstantinides), IMDCT (método de Marovich) e decodificação de Huffman (algoritmo *tree-clustering*).
- *Algorithmic SIMD:* Baseada na versão *SIMD,* combinada com implementações SIMD dos novos algoritmos para os estágios de síntese de filtros polifásicos e IMDCT usados na versão *algorithmic.* A decodificação de Huffman também foi baseada no algoritmo *tree-clustering.*

Comparação de desempenho

Sobre as condições de teste descritas pelos autores vale mencionar:

- *Compiladores*: As 5 versões do decodificador MP3 (*standard*, *basic*, *SIMD*, *algorithmic*, e *algorithmic-SIMD*) foram compiladas usando três compiladores: Intel C++ 7.1, Microsoft Visual C++ 6, e Visual C++ . NET 2003. A Tabela 1 do artigo resume as diferentes opções de

compilação utilizadas para ajustar o desempenho destes códigos-fonte. Resumindo estas opções pode-se dizer que:

- O2: inclui otimizações clássicas que são independentes do processador, expansão de funções inline;
- G6: otimiza código para Pentium Pro, Pentium II, e Pentium III, gerando código que é compatível com processadores anteriores;
- G7: optimiza código para o Pentium 4, gerando código que é compatível com processadores anteriores;
- Intel QxK: permite vetorização usando instruções SSE e MMX incluídas no Pentium III e Pentium 4;
- Microsoft arch:SSE: utiliza instruções SSE and cmov.

- *Processadores*: As implementações foram testadas nos processadores: AMD Athlon, Intel Pentium III e Intel Pentium 4. A Tabela 2 apresenta detalhes sobre os processadores utilizados na avaliação, tais como: família, memória cache, memória, e sistema operacional.
- *Bitstream de teste*: Foi utilizado um bitstream conhecido como Tristana, cujas características são mostradas na Tabela 3 do artigo.

Os autores, em seus experimentos, mediram o número de ciclos de clock por quadro ao invés do tempo gasto para decodificar o quadro, para os resultados obtidos fossem independentes do freqüência de clock do processador.

Vale destacar alguns dos resultados apresentados:

- As Figuras 6 (*standard* x *basic*) e 7 (*basic* x *demais versões*) do artigo mostram o desempenho em termos de ciclos de clock por quadro para as versões: *standard*, *basic*, *SIMD*, *algorithmic*, e *algorithmic-SIMD*, para três processadores e três compiladores estabelcidos pelo setup de teste. Nota-se uma diminuição considerável dos estágios mais lentos: síntese de filtros polifásicos, IMDCT, e Huffman, a medida que aumenta a exploração de funcionalidades da arquitetura e melhores algoritmos.
- A Figura 6 do artigo mostra um maior número de ciclos requerido pelo processor Pentium 4 em relação ao Pentium III e ao Athlon. Este maior número é explicado pelo maior número de estádios do pipeline do Pentium 4, 20, contra 12 do PentiumIII e 10 do Athlon. De acordo com os autores este maior número de estágios do pipeline embora tenha aumentado a freqüência de clock do Pentium 4 em relação ao Pentium III, também aumenta a penalidade para o código não-otimizado.
- A Figura 8 do artigo mostra os valores de speedup obtidos pelas versões *SIMD*, *algorithmic*, e *algorithmic-SIMD* comparadas à versão *basic* compilada com o compilador Microsoft Visual C++ 6. A figura mostra que a versão *algorithmic-SIMD* compilada com o mesmo compilador é 4 vezes mais rápida para o Pentium 4, 5 vezes mais rápida para o Pentium III, e 4,5 vezes mais rápida para o Athlon.
- Os dados da Figura 8 também mostram que as opções de compilação Intel QxK e Microsoft arch:SSE (aplicadas sobre a versão *basic* do decodificador) obtem menos speedup que qualquer outro executável obtido a partir da versão *SIMD* do decodificador.

Conclusões

O artigo termina ressaltando 3 lições extraídas destes experimentos:

- *Explorar funcionaliades da arquitetura pode ser tão importante quanto escolher os algoritmos mais eficientes*: comparando as Figuras 7 e 8, pode-se verificar que tanto o uso de instruções SIMD (versão *SIMD*) quanto o uso de algoritmos mais eficientes (versão *algorithmic*) , proporcionam melhorias consideráveis.
- *Programadores podem explorar funcionalidades da arquitetura de maneira mais eficiente que os compiladores*: conforme mostram os resultados da Figura 8 do artigo, as opções Intel QxK e Microsoft arch:SSE obtem menos speedup que qualquer outro executável obtido a partir da versão *SIMD*.
- *A escolha das opções de compilação depende da aplicação*: Em todos os resultados mostrados no artigo há pouca diferença de desempenho entre as opções G6 e G7. Por outro, pode-se notar em alguns um aumento de desempenho através do uso das opções de vetorização QxK e arch:SSE (vide Figuras 7 e 8 do artigo), o que é de se esperar dado que a decodificação MP3 é bastante intensa em operações vetoriais.