Biblioteca científica de processamento de sinais distribuída

Scientific library of distributed signal processing

Ailton Akira Shinoda[1]

Resumo

Este artigo descreve a implementação de um pacote de software livre educacional aplicado ao processamento digital de sinal inteiramente desenvolvido no sistema operacional Linux. O caráter inovador do trabalho baseia-se na distribuição do processamento dos blocos do pacote em vários nós ou máquinas da rede de computadores. Esta abordagem é recomendável para aplicações onde haja um intenso processamento, distribuindo ou dividindo as tarefas entre os diversos processadores
Palavras-chave: MPI. Processamento de sinal. FIR. IIR. Software livre.

Abstract

This paper describes the implementation of an educational open software package applied in the digital signal processing wholly developed in the Linux operational system. The innovation in this work is based on processing block package distributed in many nodes or processors in the network. This approach is recommended for application where there is a high processing; distributing or dividing the tasks among varies processors.
Key words: MPI. Signal processing. FIR. IIR. Open software.

Introdução

A utilização de software no processo educativo é cada vez mais freqüente, e, no caso específico do curso de Engenharia Elétrica, é altamente recomendável o emprego dessa ferramenta. Infelizmente existem poucos programas livres e a aquisição de software comercial torna-se inviável pelo custo significativo das licenças. Além disso, os programas comerciais não atendem as expectativas dos alunos/docentes, pois operam de forma interpretada e com tempo de processamento longo para muitas situações práticas. Diante desta necessidade, o artigo apresenta a implementação de uma biblioteca voltada ao desenvolvimento de tarefas ligadas ao processamento de sinais que possui duas características básicas: (i) portabilidade e (ii) velocidade de processamento.

A questão de velocidade de execução pode ser interpretada como sendo apenas uma característica do hardware ou eficiência do código gerado. Por outro lado, novas abordagens para esse problema vêm sendo disponibilizadas nos pacotes livres. Uma solução elegante para o problema é a execução dos programas em arquitetura de cluster de

[1] Mestre e Doutor em Engenharia Elétrica e Computação (área de comunicação móveis) pela UNICAMP. Entre 2000 a 2005 foi Professor Adjunto do Departamento de Engenharia Elétrica da Universidade Estadual de Londrina. Desde 2005 é Professor Assistente do Departamento de Engenharia Elétrica da Faculdade de Engenharia de Ilha Solteira da Universidade Estadual Paulista (UNESP). E-mail: shinoda@dee.feis.unesp.br

microcomputadores. Essa característica permite alto desempenho com máquinas convencionais.

A seção 2 descreve sucintamente os filtros com resposta ao impulso de duração finita (FIR) e com resposta ao impulso de duração infinita (IIR) amplamente empregados no processamento digital de sinal. A seção 3 apresenta os algoritmos FIR e IIR mais convenientes para serem aplicados em cluster de computadores. A seção 4 introduz os principais conceitos envolvidos em computação distribuída através da troca de mensagens MPI (Message Passing Interface) (KARNIADAKIS; KIRBY, 2003). A seção 5 apresenta os resultados parciais da biblioteca científica de processamento de sinais distribuída para os filtros FIR e IIR (OPPENHEIM; SCHAFER ,1989).

Filtros Digitais

Transformadas e filtros são as partes principais dos sistemas lineares de processamento de sinais. Embora as técnicas sejam diretamente aplicáveis ao processamento de sinais determinísticos, muitos métodos estatísticos de processamento de sinais empregam blocos componentes que são, sob certos aspectos, similares.

Filtros não recursivos são caracterizados por uma equação de diferenças na forma

$$y(n) = \sum_{i=0}^{M} b_i x(n-i) \qquad (1)$$

onde os coeficientes b_i se relacionam diretamente com a resposta ao impulso do sistema, isto é, $b_i = h(i)$. Devido ao comprimento finito de suas respostas ao impulso, filtros não recursivos são também chamados de filtros com resposta ao impulso de duração finita (FIR). Podemos expressar a equação (1) como

$$y(n) = \sum_{i=0}^{M} h(i) x(n-i) \qquad (2)$$

Aplicando a transformada z à equação (2), tem-se à seguinte relação entrada-saída

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{i=0}^{M} h(i) z^{-i} \qquad (3)$$

Na prática, a equação (3) pode ser implementada de várias formas distintas, usando como blocos básicos atrasos, multiplicadores e somadores (ANTONIOU ,1993).

A realização mais simples de um filtro FIR provém da equação (3). A estrutura resultante, que pode ser vista na Figura 1, é chamada de realização na forma direta, pois seus coeficientes multiplicadores são obtidos diretamente da função de transferência do filtro (JACKSON; KAISER ,1968).

A função de transferência de um filtro recursivo é dada por

$$H(z) = \frac{N(z)}{D(z)} = \frac{\sum_{i=0}^{M} b_i z^{-i}}{1 + \sum_{i=1}^{N} a_i z^{-i}} \qquad (4)$$

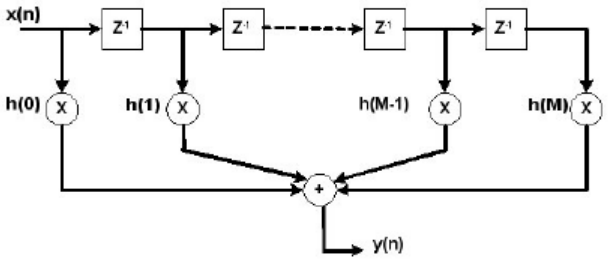

Figura 1. Forma direta para filtro FIR.

Como, na maioria dos casos, tais funções de transferência resultam em filtros cuja resposta ao impulso tem comprimento infinito, os filtros recursivos também são chamados de filtros com resposta ao impulso de duração infinita (IIR).

Os filtros digitais IIR apresentam uma ampla variedade de realizações possíveis. A realização em cascata é apresentada na Figura 2, onde os blocos

básicos representam funções de transferência simples de ordem 2 ou 1 (JACKSON ,1996).

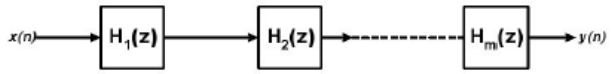

Figura 2. Forma cascata para filtro IIR.

A forma cascata baseada em blocos de segunda ordem é associada à seguinte decomposição da função de transferência:

$$H(z) = \prod_{k=1}^{m} \frac{d_{0k} + d_{1k} z^{-1} + d_{2k} z^{-2}}{1 + p_{1k} z^{-1} + p_{2k} z^{-2}} \qquad (5)$$

Essa função de transferência pode ser implementada pelo diagrama de blocos da Figura 3

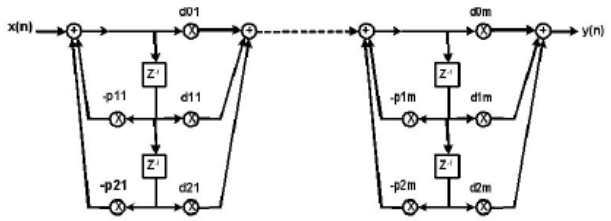

Figura 3. Realização de bloco de segunda ordem.

Algoritmo FIR e IIR

Normalmente, a implementação direta da convolução dada pela Eq. (1) gera uma saída vetorial com *M-1* amostras a mais do que o vetor de entrada. As amostras anteriores e posteriores em relação as amostras de entrada são configuradas com valores nulos quando a seqüência de entrada sobrepõe uma parte dos coeficientes do filtro. A saída do filtro apresenta um comprimento transiente igual ao comprimento do filtro até que os zeros não influenciem a saída. Um efeito similar ocorre quando o final da seqüência de entrada é atingida. No processamento digital de sinal, os blocos de entrada possuem superposição que permite o descarte dos transientes de cada bloco, característica do algoritmo *overlap and save* (EMBREE; KIMBLE,1991) usado nesse artigo. O algoritmo FIR empregado nesse trabalho gera a mesma quantidade de amostras entre a entrada e saída. A figura 4 ilustra a convolução do filtro FIR empregado pelo algoritmo seqüencial. As primeiras *M/2* amostras da convolução (*(M-1)/2* para *M* ímpar) não são calculadas. Assim, as amostras de saída se alinham com as amostras de entrada.

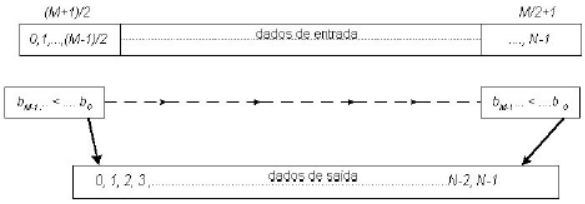

Figura 4. Sobreposição entre a sequência de entrada e os coeficientes do filtro.

O algoritmo paralelo do filtro FIR implementado nesse trabalho está descrito na Figura 5.

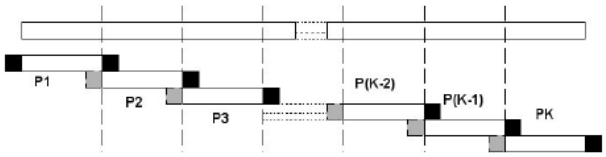

Figura 5. Algoritmo FIR paralelo.

A seqüência de entrada é dividida entre os processadores (P1,P2,...,PK) do cluster e cada nó executa a operação seqüencial descrita na Figura 4. Note que a parte hachurada mais clara são amostras das entradas e as escuras são aquelas com valores nulos descritos anteriormente no algoritmo seqüencial.

No caso do filtro IIR, a seqüência de entrada é dividida pela quantidade de seções de blocos de segunda ordem (Figura 3). Cada processador do cluster está associado a uma seção de segunda ordem, portanto na maior parte do tempo os processadores estarão trabalhando simultaneamente.

MPI

Um sistema paralelo consiste em fazer com que o sistema operacional (no caso o Linux) possa entender que um determinado software deva rodar paralelamente, e como ele deve ser distribuído. Para tanto, deve-se implementar as instruções que dizem como, quando e onde deve ser feita a divisão do software em processos menores que irão ser executados independentemente em cada um dos nós. Essa implementação é feita dentro do próprio código fonte do software (no caso desta máquina, em MPI). No cluster, empregam-se dois tipos de paralelização, a de grão grosso, que consiste na distribuição entre os diversos nós da máquina diferentes processos menores. E a de grão fino, que tenta paralelizar os laços e subrotinas dos softwares.

O sistema foi projetado visando o funcionamento em um conjunto de PCs conectados numa rede local Ethernet 100 Mbps. A implementação foi feita utilizando-se a linguagem C, o padrão de troca de mensagens MPI para computação distribuída e o sistema operacional Linux.

[* O filtro não suporta esse formato de arquivo | incorporado.TIF *]A utilização de bibliotecas de trocas de mensagens LAM/MPI, que implementam o padrão MPI, permitiu que todo o mecanismo de comunicação de dados entre os computadores fosse realizado em alto nível, através de chamadas de rotinas em C. Este mecanismo de comunicações é implementado dentro da biblioteca LAM/MPI fazendo uso da pilha de protocolos TCP/IP. Essa idéia é ilustrada na Figura 6 para o algoritmo FIR descrito na seção anterior (vide Figura 5).

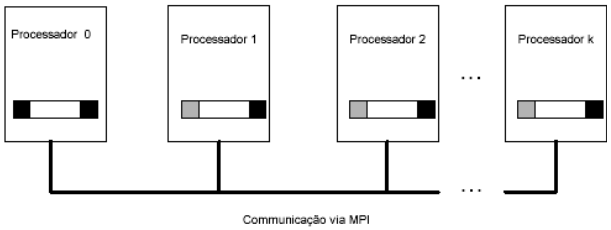

Figura 6. Processo esquemático do MPI para FIR.

Resultados

A Figura 7 mostra o sinal de entrada para os filtros FIR e IIR

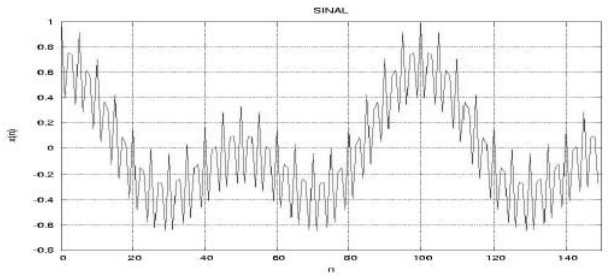

Figura 7. Sinal de entrada - x(n)

O filtro empregado no sinal de entrada da Figura 7 é um filtro passa-baixa com as seguintes características:

- faixa de passagem: *0-0.2* $f_{s;}$
- variação máxima permitida na transmissão da faixa de passagem (ripple): 0.5 dB;
- faixa de bloqueio: 0.25 – 0.5 $f_{s;}$
- atenuação mínima necessária para a faixa de bloqueio: 40 dB.

onde f_s é a freqüência de amostragem. O comprimento do filtro FIR passa-baixa para preencher essas características é M= 32 e os coeficintes b_l da Eq. (1) podem ser determinados pelo algoritmo de McClellan e Parks (1973). A quantidade de máquinas empregadas no cluster foi 6 em todas as configurações. A Figura 8 mostra o resultado do algoritmo FIR distribuído passa-baixa .

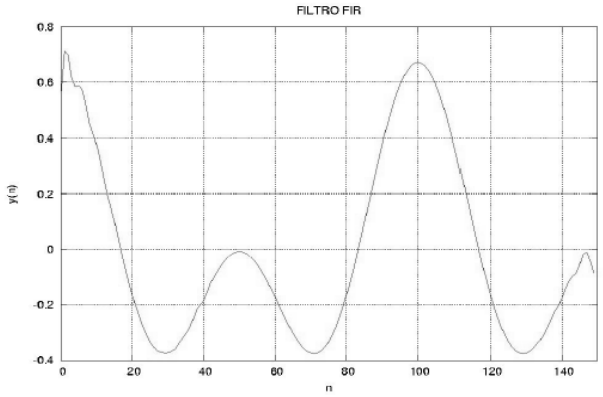

Figura 8. Filtro FIR distribuído passa-baixa.

Empregando as mesmas especificações para projetar o filtro IIR passa-baixa e selecionando o método que apresenta a menor transição da largura de banda para uma determinada ordem, chega-se ao filtro elíptico de quinta ordem. Assim, o número de seções da Eq. (5) é *m*= 3 sendo duas seções de segunda ordem e uma de primeira ordem. A Figura 9 mostra o resultado do algoritmo IIR distribuído passa-baixa.

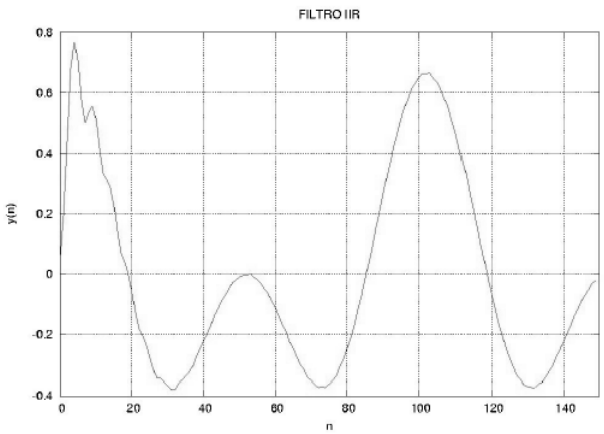

Figura 9. Filtro IIR distribuído passa-baixa.

Note a diferença entre a Figura 8 e Figura 9. Como a ordem do filtro FIR é maior do que a do filtro IIR, a curva do filtro FIR (Figura 8) é mais suave do que aquela do filtro IIR (Figura 9).

A Figura 10 mostra 6000 amostras da digitalização da voz de "chicken little" com freqüência de amostragem de 8 kHz (EMBREE; KIMBLE ,1991). Essas amostras são o sinal de entrada, *x(n)*, para o filtro FIR passa-alta.

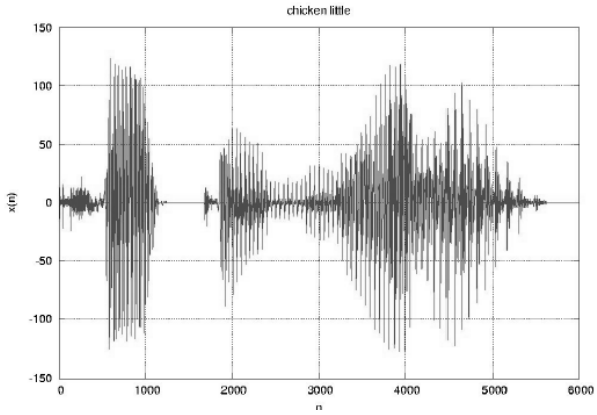

Figura 10. Digitalização de "chicken little" amostra em 8 khz.

A Figura 11 mostra o resultado do algoritmo FIR distribuído passa-alta.

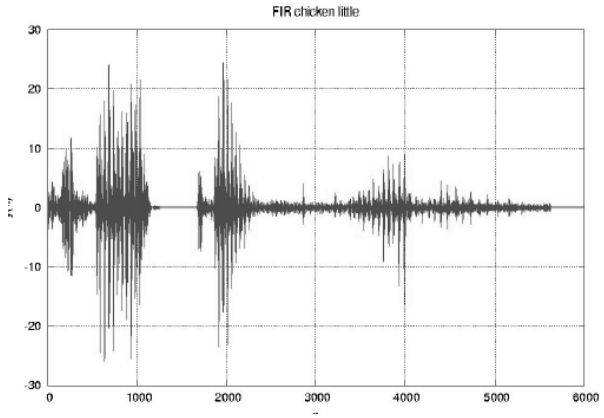

Figura 11. Filtro FIR distribuído passa-alta.

Conclusão

Este artigo apresentou a implementação do filtro FIR e IIR em uma plataforma distribuída empregando MPI na troca de mensagens. No caso do filtro FIR, a seqüência de entrada é dividida entre os processadores do cluster e cada nó executa a operação seqüencial. Em relação ao filtro IIR a

seqüência de entrada é dividida pela quantidade de seções de blocos de segunda ordem. Cada processador do cluster está associado a uma seção de segunda ordem, portanto na maior parte do tempo os processadores estarão trabalhando simultaneamente.

Para a otimização do algoritmo distribuído, houve uma preocupação no fluxo de mensagens para que a quantidade de colisões fosse a mínima possível. A biblioteca científica de processamento de sinais distribuída ainda encontra-se na fase inicial, mas as duas funções descritas no trabalho mostra a viabilidade do projeto em andamento.

Referências

ANTONIOU, A. *Digital filters:* analysis, design, and applications. New York: McGraw-Hill, 1993.

EMBREE, P. M. A; KIMBLE, B. C. *Language algorithms for digital signal processing*. Englewood Cliffs: Prentice-Hall, 1991.

JACKSON, L. B. *Digital filters and signal processing*. Boston: Kluwer Academic Publishers, 1996.

JACKSON, L. B.; KAISER, J. F. An approach to the implementation of digital filters. *IEEE Transactions on Audio and Electroacoustics,* New York, v.16, n.3, p.413-421, 1968.

KARNIADAKIS, G. E.; KIRBY II, R. M. *Parallel Scientific Computing in C++ and MPI*. Cambridge: Cambridge University Press,2003.

McCLELLAN, J. H.; PARKS, T. W. A Computer Program for Designing Optimum FIR Linear Phase Digital Filters. *IEEE Transactions on Audio and Electroacoustics*, New York, v.21, n.6, p.506-526, 1973.

OPPENHEIM, A. V.; SCHAFER, R. W. *Discrete-time signal processing.* Englewood Cliffs: Prentice-Hall, 1989.