

Algoritmo de Compressão de dados LZW em FPGA's

Heron Benincasa Nakagawa, Edward Moreno

Faculdade de Informática de Marília – Bacharelado em Ciência da Computação
Av. Hygino Muzzi Filho, 529 – Campus Universitário – CEP 17525-901 Marília – SP - Brazil

heronbenin@hotmail.com, edmoreno@fundanet.br

Resumo. *O Algoritmo LZW [1] é um algoritmo adaptativo dinâmico que comprime os dados binários em uma sequência de códigos. Ele não analisa os dados que entram e sim adiciona na tabela cada novo conjunto strings é como um Look-up-table (LUT) de strings. Dificuldades surgirão ao decorrer do projeto podendo alterar as idéias iniciais e fazendo com que sejam tomadas outras direções na respectiva proposta e no cronograma geral do projeto, sendo assim quase que impossível de detectá-las. O código que o algoritmo LZW produz pode ser de qualquer comprimento arbitrário ele deve ser expressado em mais do que um único byte (um caracter). O projeto consiste uma comparação de desempenho do Algoritmo LZW em Software, implementação em C, observando seu comportamento e desempenho e, implementá-lo em Hardware utilizando circuitos digitais programáveis (FPGA's). Após a comparação e conhecimento dos pontos fortes e fracos do Algoritmo em cada uma das implementações, poder-se-ão propor novos mecanismos adicionais de maneira a tornar a sua implementação e funcionamento mais eficiente, flexível e reconfigurável.*

Palavras Chave: *Compressão, Descompressão, LZW, FPGA.*

1- INTRODUÇÃO:

O desempenho geralmente é medido através do tempo de execução em Software de técnicas de compressão e descompressão é altamente dependente do tamanho da mensagem ou do arquivo que se deseja comprimir, tem levado nos últimos anos a comunidade científica acadêmico-industrial procurar através do Hardware a busca por velocidade e grande performance. Com o Hardware Reconfigurável, as possibilidades de uso de uma ferramenta de compressão (no caso do algoritmo LZW) com inúmeras vantagens por sua qualidade e grande taxa de rateio, leva a busca de soluções práticas a serem estudadas. O Algoritmo LZW foi e ainda é usado em aplicações tais como: compressão e descompressão de imagens, sons, programas, entre outros. Além de uma grande utilização na maioria dos tipos de transmissões de dados sejam eles para televisão, telefonia, Internet, entre outros.

O código LZW leva esse nome pelos seus criadores Lempel-Ziv em 1977 que depois em 1984 foi refinado por Terry Welch. O LZW é o mesmo que o LZ78, com algumas modificações. A saída é somente de palavras; isto quer dizer que o dicionário não pode ser vazio no começo e no início ele deve conter todos os caracteres individuais (raízes) que pode ocorrer na cadeia de caracteres. Desde todas as possibilidades de um caracter em uma string já estão no dicionário, cada passo de codificação começa com um caracter prefixado, então a primeira string encontrada para entrar no dicionário tem dois

caracteres. O caracter com o qual o novo prefixo começa é o último da próxima string. Isto é necessário para ativar o algoritmo do código para reconstruir o dicionário sem ajuda explícita do caracter na sequência do código. O código possui uma estrutura com 256 posições pré-fixadas contendo todas os caracteres (de 0 a 255, que são os caracteres ASCII), para se adicionar uma nova string ela deve não estar na definição, sempre contendo o caracter anterior e o próximo. O código LZW tem sido adotado pela maioria dos arquivos existentes tais como o ARC e o PKZIP, o algoritmo pode fazê-lo relativamente rápido e é adequado para implementação em Hardware.

O perfil do algoritmo é o seguinte: A mesa preparada pode conter milhares de itens. Inicialmente registra a posição 0 até a de 255, os usuais 256 caracteres ASCII. Ler várias palavras do arquivo a ser codificado e procurar na tabela pela mais longa coincidência. Supondo que a coincidência mais longa é dada pela string "ABC". Envia a posição de "ABC" na tabela. Lê o próximo caracter do arquivo. Se este for um "D", então registra uma nova string "ABCD" na mesa, e recomeça o processo com a letra "D". Se a mesa se enche, descarte o mais velho item ou, preferivelmente, o último usado. O conceito é que muitos arquivos, especialmente arquivos de texto, têm certas strings que se repetem várias vezes, por exemplo "the" (a palavra em inglês para "o/a"). Com os espaços, a string possui 5 bytes, ou 40 bits para codificar. Mas se nós formos adicionar a string inteira na lista de caracteres após a última, na 256, então cada vez que vier por acaso um "the", nós podemos enviar o código 256 em vez de 32,116,104,101,32 (que seria o correspondente a escrever "the"), isto tomaria 9 bits ao invés de 40 (desde que o valor 256 não passe de 8 bits).

A proposta inicial é comparar o desempenho do algoritmo LZW implementado em software e a sua posterior comparação implementado em hardware, com os resultados obtidos, há a verificação das vantagens ou não da implementação em hardware e sua utilização. O circuito em hardware se faz através de uma FPGA da família XC4000 da Xilinx, e a programação em software será feita em C usando o algoritmo mais à frente descrito.

2- TRABALHOS CORRELATOS:

Abaixo segue um resumo de uma pesquisa sobre o Algoritmo LZW implementado em linguagem C para compressão e descompressão de dados.

"LZW Compressão e descompressão Reduzindo o Fluxo de Dados na Rede" [2]. Como nós sabemos, os computadores de hoje trabalham com paralelismo e realça a riqueza de sua utilização. Paralelismo, a execução de diferentes partes de um arquivo e a visualização em cada uma das estações. Riquezas realçadas podem ser arquivadas pela indicação computacional intensiva da visualização para as mais poderosas estações de trabalho. No Explorador de Dados, o processo de seleção, consumado pela tarefa os "grupos de execuções", que são um grupo de ferramentas que podem ser transferido para as workstations . Uma vez os grupos criados, eles podem ser transferidos para as workstations onde a visualização deve ser distribuída. Além disso, os dados transferidos através da rede geralmente contém uma redundância significativa, especialmente para algumas redes sem a performance de banda larga. Este fator crítico danifica os benefícios do paralelismo pela complicada visão computacional. O propósito deste projeto , prover módulos de compressão/descompressão para reduzir a

quantia de dados através da rede usando o Algoritmo de compressão LZW. A principal razão de ter escolhido o algoritmo LZW foi pela sua grande performance e taxa de compressão.

O projeto consiste em analisar a estrutura de dados do explorador dos dados e aplicar a compressão e o descompressão de LZW criando os módulos de C para reduzir a taxa de transmissão da programação da rede. A fim saber realmente o desempenho do módulo da compressão, e dois módulos testando, o "tempo" e a "irregularidade", devem ser criados. Um outro objetivo do projeto, saber a relação da compressão da estrutura do Explorador de Dados, fluxo de dados modelo, após a aplicação do algoritmo de LZW. Algumas comparações entre a relação da compressão e o tempo que consomem na rede que aplica antes e depois o LZW foram estudadas. O modelo de dados DX é um modelo diferente do sistema de arquivos mas tem a mesma hierarquia. Em adição ao dados DX uma estrutura de dados especial, ele deve usar uma biblioteca com suporte de um Explorador de Dados quando criado seus próprios módulos. A biblioteca de Exploração de Dados é orientada a objetos, isto é, são estruturas de dados na memória global que passado por referência, e seu conteúdo é privado à implementação Furthermore, diferentes objetos podem compartilhar componentes comuns.

O módulo de "LZW Compressão" escrito para DX 2,0 , baseado na versão modificada do algoritmo de LZW como descrito no computador de IEEE, junho 1984. Esta seção descreve a característica como ele executa o algoritmo de LZW e algum detalhe sobre os interesses da estrutura de dados de DX's e da função relevante em sua biblioteca. Os principais objetos usam o "mapeamento de textura" e "equação paramétrica" geram duas terras 3-D idênticas e distribuiu estes dois objetos a duas estações de trabalho HP diferentes , " grieg " e " wagner ". Nestas estações remotas é aplicado a operação simples de translação. Depois que aqueles cálculos são feitos, recebidos e coletados os dois objetos para se tornar objeto composto em uma máquina principal (local) Neste caso, somente componentes coloridos são comprimidos pois são os únicos com um arranjo irregular.

No general, o algoritmo de LZW , é um método rápido da compressão que pode começar uma boa relação da compressão em tipos diferentes de dados. Deste projeto, pode começar o resultado previsto neste ponto. Entretanto, eles começaram notar uma melhoria satisfatória no desempenho da rede. De acordo com o estudo dele, há diversas razões para poder explicar os resultados acima. Primeiramente, não há nenhuma rotina de baixo nível suportada pela biblioteca de DX para alcançar e controlar sua estrutura de dados. O explorador dos dados, como mencionado antes, modelo orientado a objeto com a propriedade de informação escondida, adicionalmente, tem suas próprias unidades de controle de dados para controlar que tamanho do modelo de dados que pode ser aplicado. Em segundo, quando eles fazem a compressão em redes, eles devem saber o número exato dos pacotes que está sendo emitido ou recebido para testar de desempenho, ou para aplicar o mesmo algoritmo de compressão. No explorador de dados, mesmo não suportou nenhuma rotina para a programação da rede. Finalmente, é muito difícil conter dados precisos do desempenho neste projeto por causa do fluxo de dados do modelo DX. Em tal tipo do modelo, eles não podem exatamente saber a dependência e a referência entre objetos. Isto é, não podem começar a informação real do tempo sobre o tamanho total de objetos atuais. Geralmente falando, pode haver um método a resolver o dilema acima, na rede em que programa, eles podem executar um outro processo para interceptar e comprimir os pacotes antes que DX emitam para suas estações de trabalho remotas. No receptor mencionado, intercepta os dados que entram e descomprime os antes de emitirem ao módulo da execução

de DX's. O esquema acima, encaixado no Explorador de Dados e o usuário não pode especificar o módulo da compressão interativamente. Como sugerido pelo conselheiro, o módulo da compressão de LZW pode ser aplicado em muitas maneiras nos dados explorados. Um deve conservar os dados gerados pelo módulo da "exportação". Este será o trabalho futuro que pode interativamente interpretar o formato de dados de DX's e descomprime os dados do arquivo armazenado. Ele também pode conservar tamanhos de arquivos tremendos em um sistema de arquivos quando vários usuários usam o Explorador de Dados em um mesmo sistema de arquivos, pois eles sempre possuem "sistema de arquivo cheio" problemas no curso CS 418.

3- ALGORITMO LZW

O algoritmo LZW é organizado através de uma mesa de translação que mapeia strings na saída do caractere no código de comprimento fixo. A mesa de string LZW foi uma propriedade prefixa em todas as strings da mesa se ela realmente estiver na mesa. O que é se a string wK, composta por algumas strings w e alguns caracteres simples K, isto está na mesa, então w está na mesa. K é chamado de caractere de extensão no prefixo da string w. A mesa de strings LZW contém strings que devem ser encontradas antes da mensagem ser começar a ser comprimida. Ele consiste em rodar exemplos de strings na mensagem, então a avaliação das strings refletem as estatísticas da mensagem.

3.1 O Algoritmo:

Módulo de Compressão

Inicialize a mesa para conter um caractere simples numa string

Leia o primeiro caractere de entrada → string de prefixo x

Passo: Leia o próximo caractere de entrada Y

if não encontrar K(entrada esgotada):

código(x) → saída: EXIT

if xY existir na mesa de strings:

xY → x; repeat passo.

else xY não está na mesa de strings:

código(x) → saída:

xY → mesa de strings;

Y → x; repeat passo.

Módulo de Descompressão

Descompressão: Primeiro código de entrada → CÓDIGO → VELHOcódigo;

com CÓDIGO = código(Y),

Y → saída;

Y → FINchar;

Próximo Código: Próximo código de entrada → CÒDIGO → ENTRADAcódigo;

if nenhum novo código: EXIT

if CÓDIGO não definido (caso especial):

FINchar → saída;

VELHOcódigo → CÒDIGO;

código(VELHOcódigo, FINchar) → ENTRADAcódigo;

Próximo Símbolo: if CÓDIGO = código(xy)
 Y → stack,
 código(x) → CÓDIGO;
 Vá para o próximo símbolo;
 if CÓDIGO = código(Y)
 Y → saída;
 Y → FINchar;
 Faça enquanto stack não for vazio;
 stack topo → saída;
 POP stack;
 VELHOcódigo, Y → mesa de strings;
 ENTRADAcódigo → VELHOcódigo;
 Vá para o próximo código;

Exemplo:

String de Entrada = /WED/WE/WEE/WEB/WET

Caracter de Entrada	Código de Saída	Novo Valor do Código	Nova String
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

4- RESULTADOS ESPERADOS

A análise do projeto será feita com uma série de dados a serem inicialmente para testes feitos em software com vários tipos de comprimento e tamanhos, textos, sons e imagens. O mesmo será aplicado à implementação em hardware, ou seja, os mesmos códigos: os mesmos textos, as mesmas imagens, os mesmos sons. Os dados obtidos através dessa análise deverão ser relevantes, pois o mesmo processo será igual em ambos. A partir desses a conclusão será feita.

O trabalho encontra-se em sua fase inicial, ou seja, a documentação e uma visão geral e estudo da implementação em Linguagem C.

5- REFERÊNCIAS

- [1] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, Vol. IT-23, No. 3, May (1977), pp. 337-343.
- [2] Terry A. Welch " uma técnica para High-Performance dados compressão " IEEE COMPUTADOR, junho 1984. dados da IBM dos pp. 8-19 explorador 2,0 [de 2].