1 Compactação de matrizes

O processo de compactação de matrizes consiste em reduzir o número de bytes necessários para alocar uma matriz na memória. Esse processo permite que uma maior quantidade de informações seja armazenada, utilizando o mesmo hardware disponível para o método de alocação de memória tradicional. Uma das vantagens em aplicar esse método é a possibilidade de trabalhar com grandes quantidades de informações sobre dados diretamente da memória sem a necessidade de acesso a disco, leitura e escrita, o que onera o processamento.

As etapas e as implementações necessárias para utilizar o método proposto são descritas a seguir. O processo de compactação de uma matriz é realizado por linha, ou seja, o número de colunas é reduzido. Considere uma matriz $M_{l \times c}$, na qual l é o número de linhas e c o número de colunas. Cada elemento dessa matriz, denominado m_{ij}, armazena apenas um número inteiro. Para um número inteiro ser armazenado na memória de um processador de 64 bits, utilizando o maior número de bits disponíveis, demandam-se 64 bits. Logo, o número de bits (nb_T) necessários para alocar todos os elementos de uma linha da matriz M da forma tradicional pode ser definido por:

$$nb_T = c \times 64 \tag{1}$$

O nb_T calculado representa um número maior de bits do que realmente é utilizado. Considere que em uma linha da matriz, por exemplo, existam apenas elementos com valores iguais a 0 ou 1. Tais valores são representados em binário apenas por 1 bit. Portanto, quando alocados em uma variável que demande 64 bits para ser armazenada em memória, ocorre uma perda de 63 bits, que poderiam ser utilizados para armazenar outros números ou informações. Esse exemplo trata do caso mais expressivo com relação ao desperdício de memória. Calculando o número de bits nb_C realmente necessário para implementação desse exemplo, tem-se

$$nb_C = c \times 1 \tag{2}$$

A economia E_c de bits pode ser representada pela diferença entre nb_T e nb_C,

$$E_c = nb_T - nb_C \tag{3}$$

$$E_c = c \times 64 - c \times 1 \tag{4}$$

$$E_c = c \times (64 - 1) \tag{5}$$

$$E_c = c \times 63 \tag{6}$$

Com base nas equações descritas anteriormente, pode-se constatar que a economia E_c é elevada, pois c colunas demandam apenas $c \times 1$ bits ao invés de $c \times 64$ bits. O mesmo ocorre com outros valores que não necessitem de 64 bits para serem representados em memória, pois nem sempre todos os bits são utilizados.

Existem duas formas de realizar as manipulações desses bits para minimizar a perda de espaço. Cada uma dessas formas segue uma metodologia específica que deve ser aplicada à determinados casos.

A primeira metodologia consiste em determinar o valor do maior elemento que deve ser armazenado na matriz e como consequência determina o maior número de bits (B) realmente necessário para armazenar esse valor. Cada elemento m_{ij} da matriz M é representado por uma tira composta por 64 bits. No exemplo mencionado, para os valores 0 e 1, o valor de B é igual a 1, ou seja, em uma tira de 64 bits, podem-se alocar 64 valores. Cada bit dessa tira irá representar um dos valores a serem alocados. Esse fato é responsável pela economia de bits, pois 64 valores, que eram armazenados tradicionalmente em 64 colunas, são armazenados em apenas uma.

Se o maior valor a ser armazenado é 32, então $B = 5$, pois o número 32 demanda 5 bits para ser representado na memória. Assim, a tira é dividida em 12 regiões, totalizando 60 bits. Os primeiros 12 valores são armazenados nessas regiões, sendo que 13^o valor deve ser dividido, para que seja armazenada parte nos quatro bits que sobram da tira e o bit restante em outra tira. A divisão de um valor em dois segmentos faz com que o processador execute um número maior de operações para armazenar os valores, o que em determinados casos pode ser inviável. Essa metodologia deve ser aplicada em matrizes que possuem valores próximos, para que não ocorra perda de bits.

A segunda metodologia consiste em tentar minimizar as perdas quando os números possuem valores distantes. Por exemplo, quando deseja-se armazenar valores de magnitudes diferentes, como 0, 1 e 1023. Nessa faixa de valores, pode-se verificar que os valores 0 e 1 necessitam apenas de 1 bit, enquanto o valor 1023 demanda 10 bits. Seguindo a primeira metodologia, teríamos que dividir a tira em regiões de 10 bits, o que para armazenar os valores 0 e 1, faz com que ocorra a perda de 9 bits, não permitindo uma compactação mais eficaz. Para contornar esse problema é sugerida uma proposta de compactação dos dados. O primeiro passo é determinar o maior número de bits necessários B. Logo em seguida, determinar o número de bits necessários para representar B, sendo esse número definido por S. A tira não será mais representada em regiões de mesmo tamanho, mas em regiões com tamanhos distintos. Dessa maneira, um número para ser representado terá que ter o número de bits necessários para representá-lo armazenado em uma região S, e na região logo em seguida, o seu respectivo valor armazenado. Para representar um número, por exemplo 1, nessa nova metodologia, com o número de bits do maior elemento igual a 10, demanda-se o número 5, em que 4 bits é utilizando para representar o tamanho da região a ser usada e mais um bit para armazenar o valor nesse caso.

Para ilustrar essa metodologia, os valores 0, 1023 e 1 serão alocados em uma tira. Supondo que esses valores fazem parte de uma linha de uma matriz e que são armazenados em três colunas pelo método tradicional. Na tabela 1 esses valores estão representados em binário.

Tabela 1: Representação em binário.

Número	Binário	Quantidade de bits
1	1	1
1023	11111111111	10
0	0	1

Seguindo a metodologia proposta tem-se a tira

$$000011111111111101010001 \tag{7}$$

em que os valores 0, 1023 e 1 já foram adicionados. Uma representação, na qual é possível verificar as regiões em que ocorrem o armazenamento está descrita a seguir

$$\underbrace{0}_{0}\ \underbrace{0001}_{1}\underbrace{1111111111}_{1023}\underbrace{1010}_{10}\ \underbrace{1}_{1}\ \underbrace{0001}_{1} \tag{8}$$

em que as regiões para armazenar os número de bits e os valores em binários estão armazenados na tira.

O processo de compactação de uma Matriz M, com c colunas é realizado linha a linha. Dado uma linha j da matriz M e $nb(x)$ uma função que retorna o número de bits necessário para representar o valor x, o número total de bits para compactar essa linha é defindo por:

$$nb_T = \sum_{i=1}^{c} nb(m_{ij}) + nb(S) \tag{9}$$

em que B é o número de bits necessários para representar o maior valor alocado na matriz M e $S = nb(B)$, o número de bits nb_T pode ser definido como:

$$nb_T = c \times nb(S) + \sum_{i=1}^{c} nb(m_{ij}) \tag{10}$$

Como a variável m_{ij} deve assumir um valor, ela será considerada como uma variável aleatória, tal que $m_{ij} \sim U_d(0, N)$, sendo N o maior valor possível a ser armazenado. Assim, $S = nb(N)$. Logo, pode-se calcular o valor esperado para nb_T, que pode ser definido por

$$E(nb_T) = E(c \times nb(S) + \sum_{i=1}^{c} nb(m_{ij})) \tag{11}$$

$$E(nb_T) = E(c \times nb(S)) + E(\sum_{i=1}^{c} nb(m_{ij})) \tag{12}$$

$$E(nb_T) = E(\sum_{i=1}^{c} nb(m_{ij})) \tag{13}$$

$$E(nb_T) = \sum_{i=1}^{c} E(nb(m_{ij})) \tag{14}$$