



Seminário de Infraestrutura de Hardware:

Parallel patterns: Sparse
Matrix - Vector Multiplication





Equipe



Fillipe Baptistella

Ítallo Antônio

João Marcos

Juliana Serafim

Mateus Galdino



10.1 Background

10.2 Parallel SpMV Using CSR

10.3 Padding and Transposition

10.4 Using Hybrid to Control Padding

10.5 Sorting and Partitioning for Regularization



Context

Qual a relação com GPU?

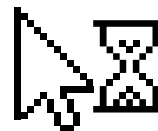
- Designer for Parallel Processing
- And to accelerate computer graphic workloads

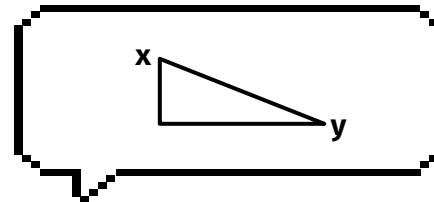
“As the 21st century began, computer scientists realized that GPUs had the potential to solve some of the world’s most difficult computing problems”.



Introduction

Capítulo 10 aborda a resolução de operações e aplicações de matrizes esparsas com foco nas soluções que permitem o melhor desempenho do sistema.





10.1

Background

Introdução a conceitos e problemáticas



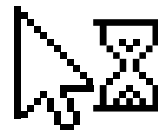
Sparse Matrix

Uma matriz esparsa é uma matriz em que **a maioria dos elementos é zero**.

Matrizes esparsas surgem em muitos problemas de modelagem em várias áreas de ciências, engenharia e finanças.

Comumente são encontradas em problemas de resolução de sistemas lineares.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$





First Problem

$$A = \begin{bmatrix} 4 & -1 & 0 & 2 & -1 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 0 \\ -1 & 4 & -1 & 2 & 0 & -1 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 0 \\ 2 & -1 & 4 & -1 & 0 & 2 & -1 & 2 & 0 & 2 & 0 & 2 & 0 & 0 \\ 2 & 0 & -1 & 4 & 0 & 2 & 0 & -1 & 0 & 2 & 0 & 2 & 0 & 0 \\ -1 & 2 & 0 & 2 & 4 & -1 & 0 & 2 & -1 & 2 & 0 & 2 & 0 & 0 \\ 2 & -1 & 0 & 2 & -1 & 4 & -1 & 2 & 0 & -1 & 0 & 2 & 0 & 0 \\ 2 & 0 & -1 & 2 & 0 & -1 & 4 & -1 & 0 & 2 & 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & -1 & 0 & 2 & -1 & 4 & 0 & 2 & 0 & -1 & 0 & 0 \\ 2 & 0 & 0 & 2 & -1 & 2 & 0 & 2 & 4 & -1 & 0 & 2 & -1 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & -1 & 0 & 2 & -1 & 4 & -1 & 2 & 0 & -1 & 0 & 0 \\ 2 & 0 & 0 & 2 & 0 & 2 & -1 & 2 & 0 & -1 & 4 & 2 & 0 & 2 & -1 & 0 \\ 2 & 0 & 0 & 2 & 0 & 2 & 0 & -1 & 0 & 2 & -1 & 4 & 0 & 2 & 0 & -1 \\ 2 & 0 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & -1 & 0 & 2 & -1 & 4 & -1 & 0 \\ 2 & 0 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & -1 & 2 & 0 & -1 & 4 & -1 \\ 2 & 0 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & -1 & 2 & 0 & -1 & 4 \end{bmatrix}$$

Pontos de atenção:

- maioria dos elementos é zero;
- os zeros não são expressivos nas operações;

Consequências:

- armazenamento de informações desnecessárias;
- impacto no desempenho;

“Storing and processing these zero elements are wasteful in terms of memory, time and energy”.



Solution

Matrizes esparsas são armazenadas de forma a evitar o armazenamento dos elementos zero de seu corpo.

Forma de armazenamento:

- Compressed sparse row - **CSR**

Definição:

A forma de armazenamento consiste em **alocar somente os números diferentes de zero**. Assim, comprime-se todos os elementos zero.




CSR - Compressed Sparse Row

Estrutura:

1. Vetor com valores dos elementos \neq de zero;
2. Vetor com o índice da coluna desses valores;
3. Vetor que indica o início de cada linha da matriz original com base na estrutura (índices) do vetor 1.

Exemplo:

Row 0	3	0	1	0		Row 0	Row 2	Row 3	{ 3, 1, 2, 4, 1, 1 }	Nonzero values data[7]	Vetor 1			
Row 1	0	0	0	0		{ 0, 2,	1, 2, 3,	0, 3 }						
Row 2	0	2	4	1		{ 0, 2, 2, 5, 7 }								
Row 3	1	0	0	1										
									Column indices col_index[7]			Vetor 2		
									Row Pointers row_ptr[5]			Vetor 3		



Another Problem

Embora o CSR diminua o armazenamento, ele não facilita operações com sistemas lineares.

$$\begin{bmatrix} A & a & 0 & a & 0 & 0 & 0 & 0 & 0 \\ a & A & a & 0 & a & 0 & 0 & 0 & 0 \\ 0 & a & A & a & 0 & a & 0 & 0 & 0 \\ a & 0 & a & A & a & 0 & a & 0 & 0 \\ 0 & a & 0 & a & A & a & 0 & a & 0 \\ & & & & \ddots & & & & \\ 0 & 0 & 0 & a & 0 & a & A & a & 0 \\ 0 & 0 & 0 & 0 & a & 0 & a & A & a \\ 0 & 0 & 0 & 0 & 0 & a & 0 & a & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ b_n \end{bmatrix}$$



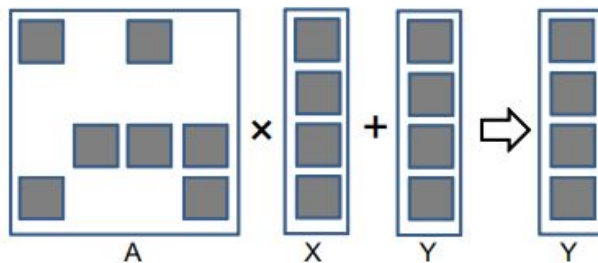


Solution

Operação: SpMV (sparse matrix-vector multiplication)

Método: adota-se uma solução X , itera $A \times X + Y$, verifica se o resultado é próximo ao vetor 0 (solução de um sistema linear).

Caso não seja, usa-se o vetor obtido anteriormente como X e se realiza uma nova iteração $A \times X + Y$.





10.2

SPMV paralelo usando BCSR



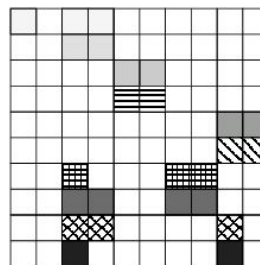


SPMV

Seria o vetor multiplicação da matriz esparsa.

É essencial em aplicações científicas e analíticas do mundo real de alto desempenho.

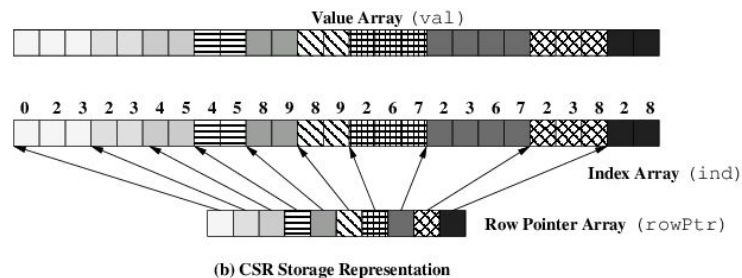
Utilizamos CSR para “compactar a matrix”.



(a) Sparse Matrix

```
for (int i=0; i <n; i++){  
    float t=0;  
    int lb = rowPtr[i];  
    int ub = rowPtr[i+1];  
    for (int j=lb; j < ub; j++){  
        int index = ind[j]  
        t += val[j]*y[index]  
    }  
    x[i] = t;  
}
```

(c) C Code for the SpMV Kernel (x=Ay)



(b) CSR Storage Representation

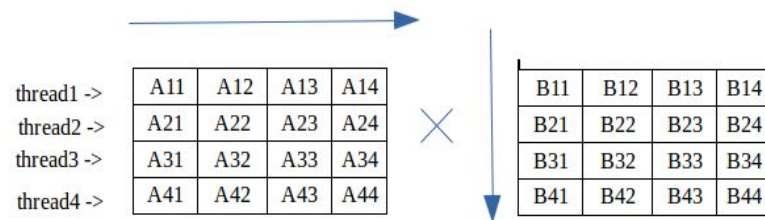


Threads ajudam na implementação

As Threads funcionam como “tarefas” que aceleram o processo de execução.

Cada Thread fica responsável por 1 linha.

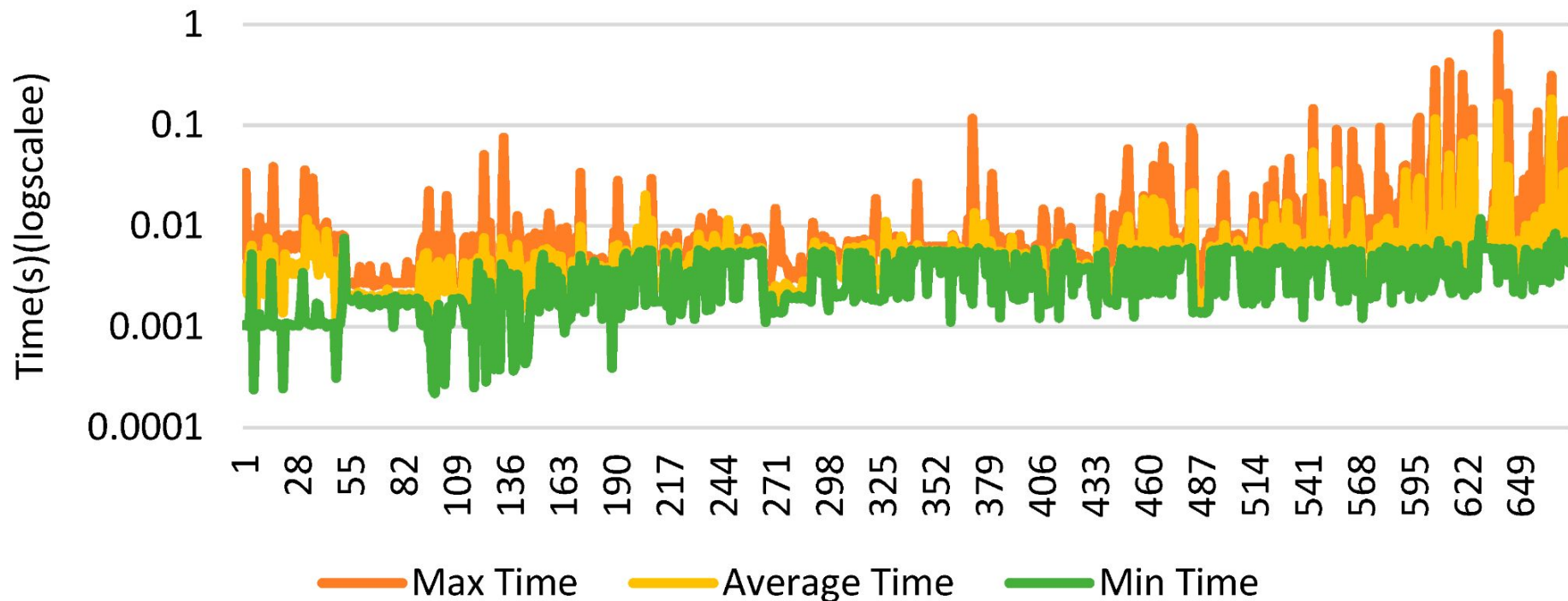
O custo computacional fica menor no final das contas.





Tempo normalmente

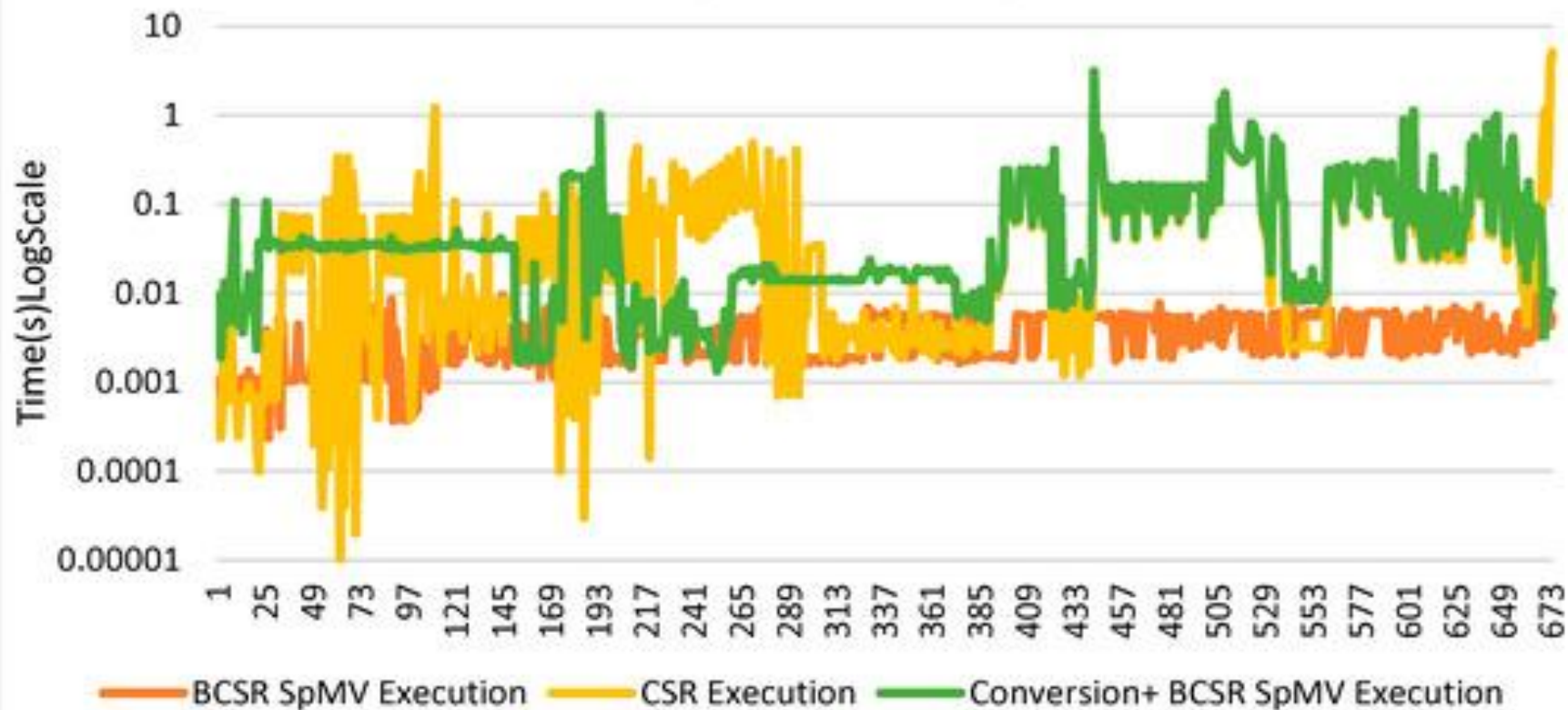
Nnz(min to max)

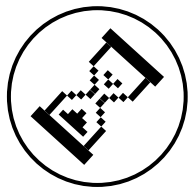




Utilizando CSR/BCSR

CSR vs(BCSR+Conversion)





10.3

Padding and Transposition

Data padding and transposition of matrix layout.

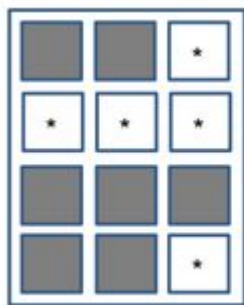


ELL Storage format

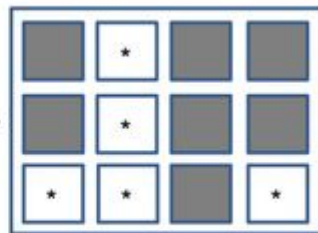
Já que é comum haver excesso de 0 nas matrizes.

Essa forma de armazenamento ajuda a poupar esses zeros no acesso à memória.

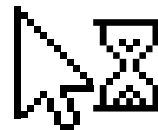
Row 0	3	0	1	0
Row 1	0	0	0	0
Row 2	0	2	4	1
Row 3	1	0	0	1



CSR with Padding



Transposed

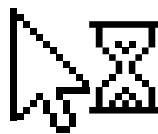




Example

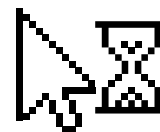
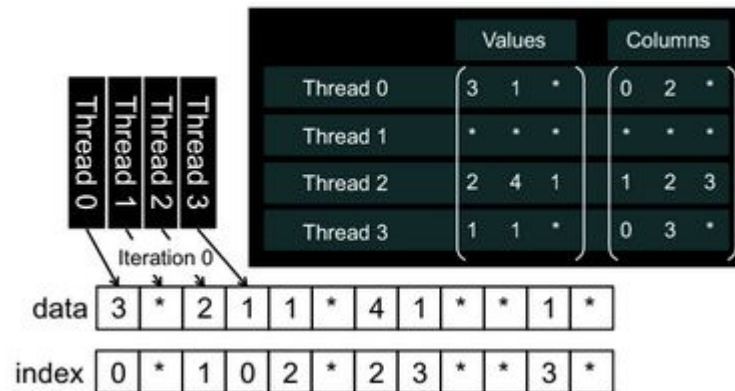
$$\text{Full Matrix} = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 & 0 \\ 0 & 4 & 5 & 0 & 6 & 0 \\ 7 & 0 & 8 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 7 & 6 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 3 & 0 \end{bmatrix}$$

$$\text{Values} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 8 & 7 & 6 \\ 5 & 0 & 0 \\ 4 & 3 & 0 \end{bmatrix}, \quad \text{Columns} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 5 \\ 1 & 3 & 5 \\ 2 & 5 & 6 \\ 3 & 0 & 0 \\ 3 & 5 & 0 \end{bmatrix}$$





ELL Storage format





SpMV/ELL

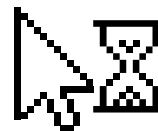
Vantagem e desvantagem em relação ao CSR:

Vantagem:

- O custo computacional e a lógica são mais simples.
- Ao permitir a coalescência de memória, SpMV/ELL deve ser executado mais rápido que SpMV/CSR.

Desvantagem:

- Quando há um número excessivo de Padded Elements, eles vão ocupar muito armazenamento e piorar os cálculos feitos pelo programa e não vão contribuir em nada para o resultado final.





10.4

Using Hybrid to Control Padding

Misturando formatos de forma colaborativa



Coordinate Format

Para lidar com Padding no ELL é utilizado o **Coordinate Format** (COO).

Cada elemento não nulo é armazenado com *col_index* e *row_index*, o que permite a sua reordenação (não altera os cálculos).

Permite fazer multiplicações entre a matriz e um vetor mais rápido

	Row 0	Row 2	Row 3
Nonzero values data[7]	{ 3, 1,	2, 4, 1,	1, 1 }
Column indices col_index[7]	{ 0, 2,	1, 2, 3,	0, 3 }
Row indices row_index[7]	{ 0, 0,	2, 2, 2,	3, 3 }

Nonzero values data[7]	{ 1 1, 2, 4, 3, 1 1 }
Column indices col_index[7]	{ 0 2, 1, 2, 0, 3, 3 }
Row indices row_index[7]	{ 3 0, 2, 2, 0, 2, 3 }



Hybrid Method

Hybrid method: criação de outro COO para redução de padding

Feito normalmente pelo host, é criada uma representação em COO para alguns elementos, que é salva em um dispositivo para ser usada após o SpMV principal.

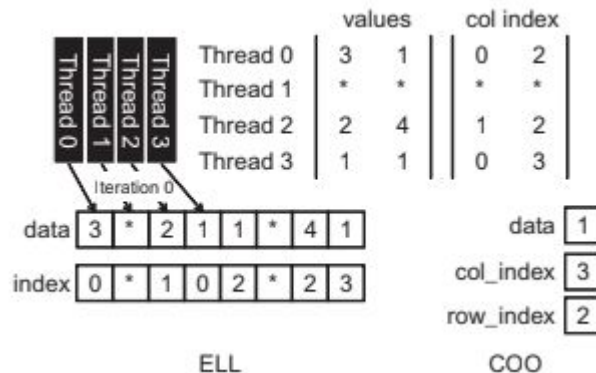
Vantagem:

Reduz a disparidade entre linhas longas e curtas e uniformiza a carga de trabalho entre threads.

Menor trabalho em aplicações que utilizam SpMV de forma iterativa

Desvantagem:

Maior trabalho quando a matriz esparsa só é usada em um único cálculo SpMV





10.5

Sorting And Partitioning For Regularization

Melhorando Ainda Mais O Desempenho



Sorting and Partitioning

- Ordenar as linhas seguindo uma classificação
- Formato JDS (jagged diagonal storage) - mantém um array que preserva o índice original
- Muito utilizado em associação com outras técnicas

Nonzero values data[7] { 2, 4, 1, 3, 1, 1, 1 }
Column indices col_index[7] { 1, 2, 3, 0, 2, 0, 3 }
JDS row indices Jds_row_index[4] { 2, 0, 3, 1 }
Section pointers Jds_section_ptr[4] { 0, 3, 7, 7 }

