Temario

- Matrices dispersas
  - Motivación e historia
  - Formatos estáticos
  - Formatos dinámicos
  - Otros formatos

Motivación

- La resolución de una gran cantidad de problemas de ingeniería involucra la utilización de matrices.

- Mejorar la precisión de la resolución de diversos problemas implica aumentar el tamaño de las matrices involucradas.

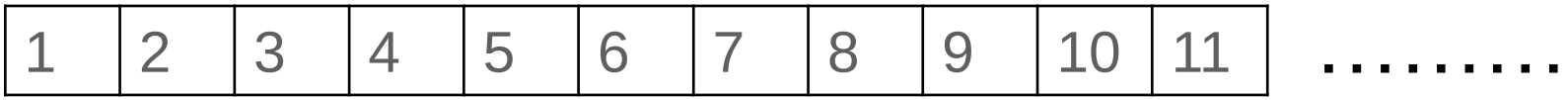
Motivación

- Situación:
  - Matrices muy grandes
  - Previsible “gran porcentaje” de valores = 0

- Se busca una forma de representar esas matrices que “cueste” menos memoria y permita acelerar los cálculos.

Representación en memoria de una matriz convencional

La memoria es una secuencia de bytes



- Ejemplo:
  - matriz de n filas y m columnas de reales de 4 Bytes
  - almacenados “por fila” (los elementos de una fila se encuentran contiguos en la memoria)
  - La posición en bytes de la celda (i, j) en la memoria es:
    inicio_matriz + **(((i – 1) * m) + j) * 4**

Matrices convencionales

Las matrices son *naturalmente* muy eficientes en el uso de memoria -para almacenar datos- y de procesador -para accederlos-.

Matrices dispersas

Definición (informal)

Es aquella que está compuesta por "muchos" elementos de valor = 0.

Matrices dispersas

Definición demasiado informal !!!! Que significa "muchos" ?!??!

- Depende …
  - Problema
  - Estructura de almacenamiento
- En general se asume un costo de almacenamiento de O(N) .. N es la cantidad de filas o columnas

Historia

- El considerado pionero en la utilización de matrices dispersas es Ralph A. Willoughby (1923-2001).
- Trabajando para resolver problemas de circuitos en laboratorios de IBM a fines de los 50’s.

Historia

- 1968, el primer congreso sobre utilización de matrices dispersas (IBM T.J. Watson Research Center)
- 1975, segundo (Argonne National Laboratory)
- 1976, Artículo de “relevamiento” I. Duff (aprox. 600 trabajos referenciados)
- ….. a la fecha crecimiento substancial del área en la computación científica

Matrices dispersas

- En esta clase nos preocupa el uso de memoria en las matrices dispersas (en el resto del curso los cálculos).
- Eventualmente se podría definir como dispersa con respecto a otro valor distinto de 0 (para almacenar).

Representación matrices dispersas

- Debemos tratar de preservar los principios de economía:
  - Mínimo consumo de memoria
  - Mínimo uso de procesador
    - “tiempo” para acceder a los datos
    - “tiempo” de cálculos
    - Localidad de datos

Matrices dispersas

- Dos grandes familias de estrategias para almacenamiento para matrices dispersas:
  - formatos estáticos: Útiles cuando se conoce la estructura de la matriz, o cuando ésta se genera en una única ocasión.
  - formatos dinámicos: Cuando la estructura de la matriz dispersa cambia en forma continua.

Formatos estáticos

- Simple o elemental (COO)
- Comprimido por fila (CRS)
- Comprimido por columna (CCS)
- Comprimido por fila a bloque (BCRS)
- Comprimido por diagonales (CDS)
- Jagged diagonal scheme and skyline

Formatos estáticos

- Formato simple o elemental (COO)
  - 3 vectores
    - Vector_f (fila)
    - Vector_c (columna)
    - Vector_d (dato)

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 3 & 4 & 0 \\ 0 & 5 & 0 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 9 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 4 & 5 & 0 \\ 0 & 0 & 6 & 0 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$d = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$
$$f = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 5 & 5 & 5 & 6 & 6 & 6 & 7 \end{pmatrix}$$
$$c = \begin{pmatrix} 1 & 2 & 5 & 6 & 2 & 4 & 2 & 3 & 4 & 1 & 4 & 2 & 5 & 6 & 3 & 5 & 6 & 7 \end{pmatrix}$$

Formatos estáticos

COO

- Se almacena la coordenada de fila y columna de cada elemento distinto de cero.
- Memoria utilizada para almacenar una matriz de dimensiones n x n y nz elementos distintos de cero ??
  - nz datos (pto. flotante doble prec.) nz*8
  - nz para col, enteros (32 bits): nz*4
  - nz para filas, enteros: nz*4

16*nz bytes

Formatos estáticos

- Ventajas
  - intuitivo
  - se utilizan estructuras del mismo tamaño
  - es igual si queremos acceder por fila que por columna
- Desventajas
  - utiliza más memoria que otras implementaciones
  - acceder por fila o columna es más difícil que en otros formatos

Formatos estáticos

- Formato CSR- (Compressed Sparse Row) o CRS o Old Yale Format
  - Vector_f (índice de cada nueva fila)
  - Vector_c (índice de columna)
  - Vector_d (dato)

Formatos estáticos

Formato CRS

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 3 & 4 & 0 \\ 0 & 5 & 0 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 9 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 4 & 5 & 0 \\ 0 & 0 & 6 & 0 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$d = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

$$f = \begin{pmatrix} 1 & 5 & 7 & 10 & 12 & 15 & 18 & 19 \end{pmatrix}$$

$$c = \begin{pmatrix} 1 & 2 & 5 & 6 & 2 & 4 & 2 & 3 & 4 & 1 & 4 & 2 & 5 & 6 & 3 & 5 & 6 & 7 \end{pmatrix}$$

Formatos estáticos

- Formato CRS
  - Un vector de punto flotante de tamaño *nz* en el que se almacenan los valores de los coeficientes distintos de cero ordenados por fila.
  - Un vector de enteros de tamaño *nz* en el que se almacenan el número de columna de los elementos distintos de cero.
  - Un vector de tamaño *n+1* siendo *n* la cantidad de filas de la matriz, en el cual se almacena el índice en el cual comienza cada fila en los dos vectores anteriores.

Formatos estáticos

- Formato CRS
  - Memoria necesaria para almacenar ??
    - *nz* * 8
    - nz * 4
    - (n+1) * 4

$$12*nz + 4*(n+1)$$

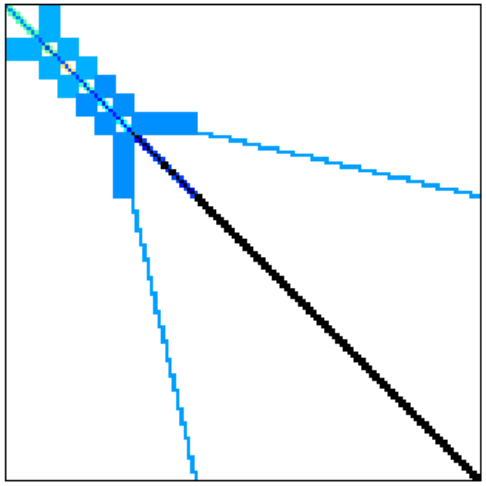
Formatos estáticos

CRS

- Ventajas
  - utiliza menos memoria que otras estrategias
  - es fácil acceder a una fila “completa”
- Desventajas
  - no se utilizan estructuras del mismo tamaño
  - es difícil acceder a una columna “completa”

Formatos estáticos

- Ejemplo: Matriz G3_circuit extraída de la colección SuiteSparse (http://sparse.tamu.edu/)
  - Utilizada en la simulación del circuito de un procesador AMD G3
  - 1.585.478 filas y columnas, 7.660.826 no ceros.
  - Almacenada como matriz densa (doble prec.) 18,28 TB
  - En formato COO: 116,89 MB
  - En CSR: 93,71 MB



Formatos estáticos

- Formato CSC (Compressed Sparse Column) o CCS
  - Igual al CSR pero almacena por columna
  - Es el formato que emplea MatLab !!!

Formatos estáticos

- Cada vez que se agrega un elemento se tiene que insertar ordenado en la matriz
  - Es necesario correr los demás elementos y rearmar la matriz
- Prueba en Octave:
  - Lleno una matriz CSC por columnas en distinto orden...

```
>> n = 512;
>> m = n*n;
>> A = spalloc( n, n, m );
>> for j = 1:n
       for i = 1:n
           A(i, j)=1+i+j;
       end
   end
```

0,88s

```
>> n = 512;
>> m = n*n;
>> A = spalloc( n, n, m );
>> for j = n:-1:1
       for i = n:-1:1
           A(i, j)=1+i+j;
       end
   end
```

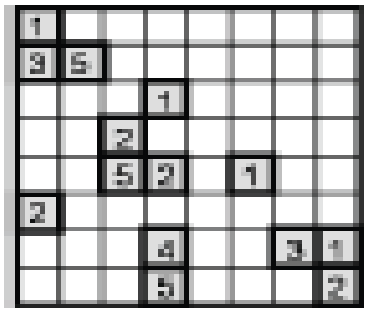
14,18s !!!

Formatos estáticos

- Comprimido por bloque de filas (BCRS)
  - Si la matriz se puede almacenar como sub-bloques regulares
  - Se utilizan los almacenamientos CRS o CCS pero accediendo a los valores según el sub-bloque

Formatos estáticos

- Comprimido por bloques de filas (BCRS)
- Matriz de nxn, con nnzb bloques de dimensión nb
- Tres vectores:
  - punto flotante val(nnzb x nb x nb)
  - enteros col (nnzb)
  - enteros bloque_fila (n/nb+1)

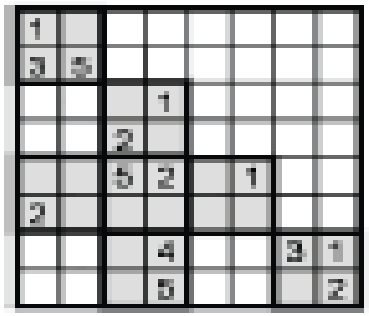


Non-zero Values	1	3	5	1	2	5	2	1	2	4	3	1	5	2
Column index	0	0	1	3	2	2	3	5	0	3	6	7	3	7

Row pointer	0	1	3	4	5	6	9	12	14

# of stored elements: 14 values
# of fetches for a SpMV: 59 fetches
Filling Ratio: 100 %

CRS

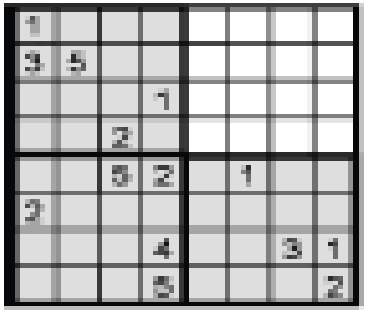


Non-zero Values	1			1			5	2		1		4	3	1
	3	5	2		2							5		2
Column index	0		1		0		1		2		1		3	

Row pointer	0	1	2	5	7

# of stored elements: 28 values
# of fetches for a SpMV: 30 fetches
Filling Ratio: 50 %

BCRS (2x2)



Non-zero Values	1						5	2		1		
	3	5			2							
				1				4			3	1
			2									2
Column index	0				1				2			

Row pointer	0	1	3

# of stored elements: 48 values
# of fetches for a SpMV: 23 fetches
Filling Ratio: 29 %

BCRS (4x4)

Formatos estáticos

Comprimido por bloque de columnas (BCRS)

- Ventajas:
  - Mejora el tiempo de acceso a los bloques
  - Puede ser necesario almacenar 0s para respetar los tamaños de bloque
- Desventajas:
  - Sirve “solo” si la matriz posee bloques regulares

Formatos estáticos

- Comprimido por diagonales (CDS)

- Se almacena una matriz rectangular con las diagonales.

$$A = \begin{pmatrix} 10 & -3 & 0 & 0 & 0 & 0 \\ 3 & 9 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 0 & 0 & 8 & 7 & 5 & 0 \\ 0 & 0 & 0 & 9 & 9 & 13 \\ 0 & 0 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val(:,-1)	0	3	7	8	9	2
val(:, 0)	10	9	8	7	9	-1
val(:,+1)	-3	6	7	5	13	0

Formatos estáticos

- Jagged Diagonal Storage (JDS) (ver figura)

  1) Los elementos nz de la matriz se corren a la izquierda (se almacena un vector con los valores distintos de 0 y uno con el número de columna original de cada elemento).

  2) Se ordenan las filas de acuerdo a la cantidad de nz (se almacena un vector de permutación).

  3) Se almacena la matriz resultante por columna (se almacena en un vector con un puntero al comienzo de cada columna en los vectores anteriores).

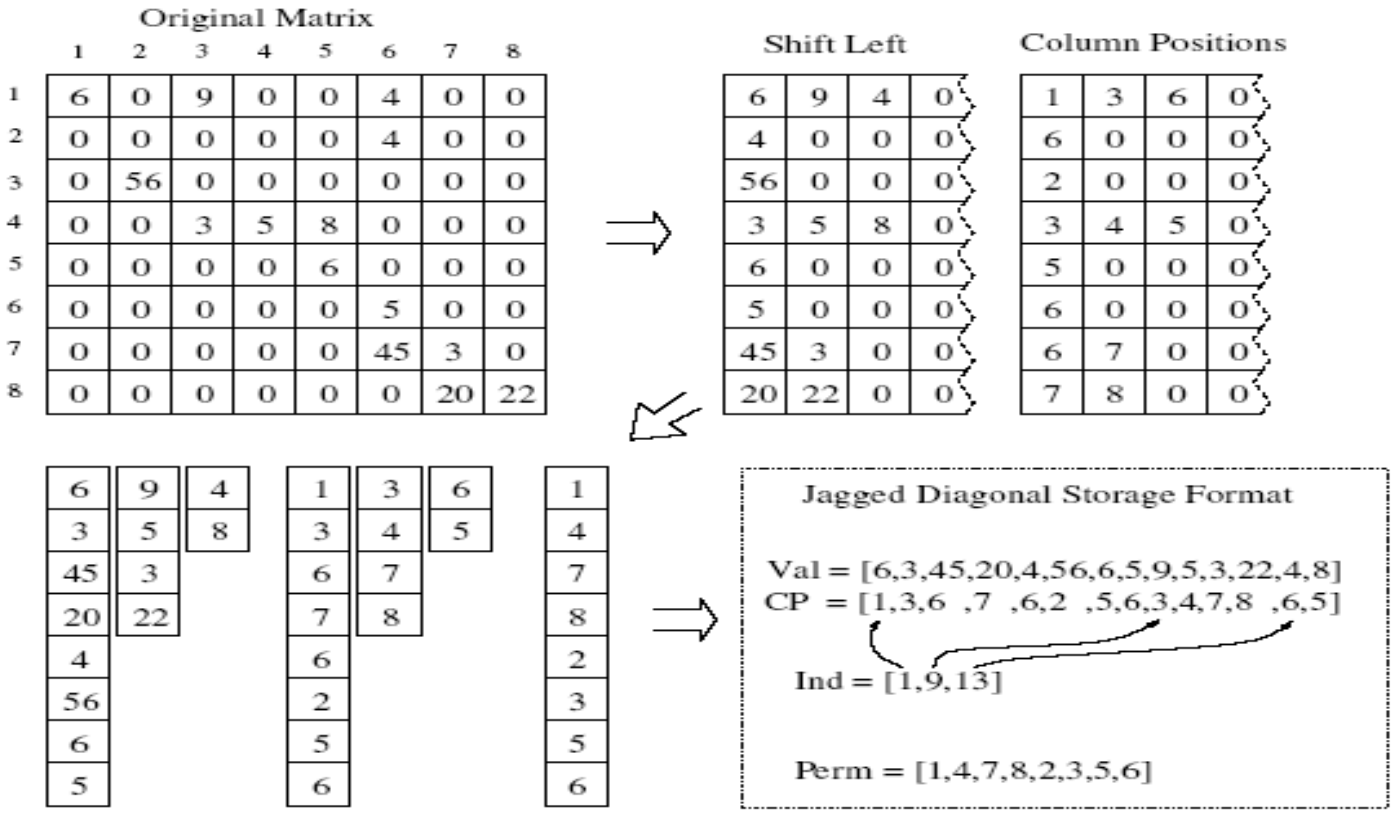


Figure 2.2: Jagged Diagonal Storage Format

Formatos estáticos

- Bit Map
  - Formato pionero caido en desuso por mucho tiempo
  - Algunos trabajos recientes utilizan estos formatos para regularizar el acceso a memoria en dispositivos como GPUs
  - Las matrices dispersas pueden representarse usando bloques de 8x8
  - Se almacena un bitmap para cada bloque no nulo (entero de 64 bits)

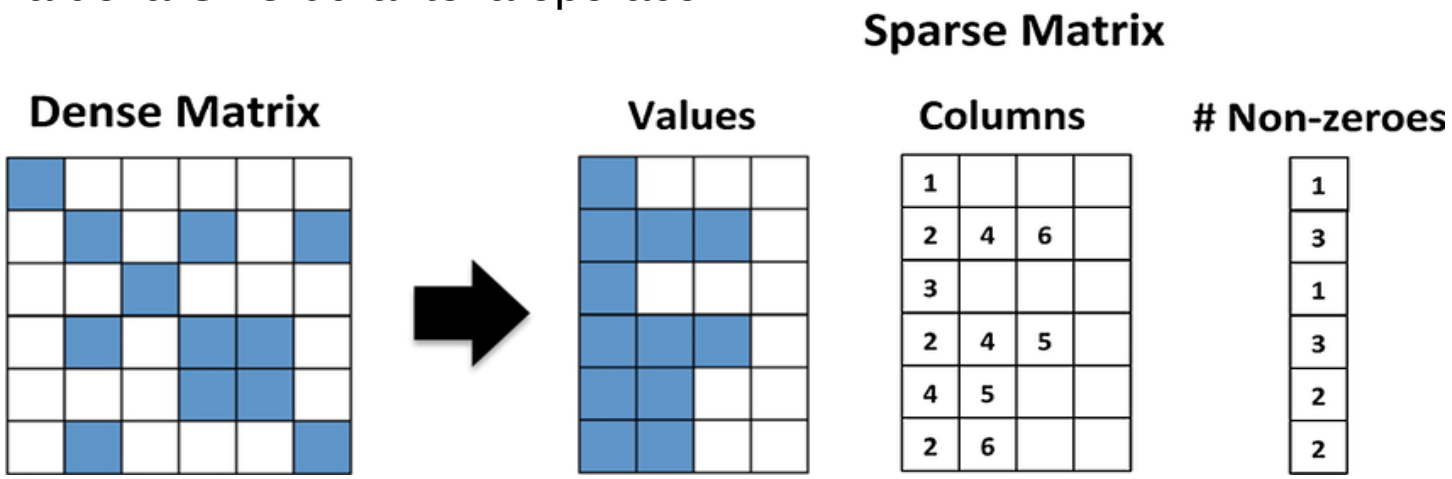
$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 3 & 4 & 0 \\ 0 & 5 & 0 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 9 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 4 & 5 & 0 \\ 0 & 0 & 6 & 0 & 7 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\begin{matrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

$$d = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

Formatos estáticos

- ELLPACK (ELL)
  - Almacena algunos 0s para optimizar el producto matriz-vector en GPUs.
  - Dos matrices de NxK (una con los valores y otra con los índices de columna)
    - N es la cantidad de filas
    - K es el máximo de nz en una fila
  - Las 2 matrices se almacenan por columnas para favorecer el acceso a memoria de la GPU durante la operacón



Formatos dinámicos

- Resuelven el problema de los formatos estáticos al insertar elementos
- No tan eficientes en almacenamiento/acceso como los formatos estáticos
- Utilizan herramientas de almacenamiento dinámico (punteros)
  - Listas enlazadas
  - Vectores de listas

Formatos dinámicos

- LLRCS (Linked List Row-Column Storage)

- Se utiliza una multi-estructura bidimensional.
- Dos vectores de tamaño filas y columnas, cada entrada:
  - un puntero para recorrer la fila correspondiente
  - un puntero para recorrer la columna correspondiente
- Cada elemento guarda:
  - Valor, fila, columna, al siguiente en la fila y en la columna.
- Memoria para nz coeficientes no nulos:
  - nz * (1 flotante + 2 punteros + 2 enteros) + 2*n punteros

Formatos dinámicos

- LLRS (Linked List Row Storage)

- Estructura unidimensional de tamaño igual a la cantidad de filas, de cada posición se puede obtener la lista de entradas de esa fila.
- Memoria para nz coeficientes no nulos:
  - nz punto flotantes
  - nz enteros
  - 2*nz punteros (o nz)
  - el vector de entrada (n punteros).

Formatos dinámicos

- LLCS (Linked List Column Storage)
- Estructura unidimensional de tamaño igual a la cantidad de columnas, de cada posición del vector se puede obtener la lista de coeficientes no nulos de esa columna.
- Posee las mismas necesidades de memoria que LLRS.

Otros formatos

- Formatos compuestos:
  - guardar la diagonal por un lado y el resto por separado
- Formatos multi-nivel:
  - estructuras tipo árbol
    - UB-tree, BUB-tree, R-tree, R*-tree, etc
  - trasladando ideas de índices, imágenes y otros.