Matrices Disperas

Pablo Ezzatti y

Octubre 2007

Índice general

Capítulo 1

Introducción

En el rea de la computacin cientfica la utilizacin de matrices est sumamaente difundida. Ejemplos de lo mencionado se pueden encontrar en la resolucin de ecuaciones diferenciales, en probramacin lineal, etc.

A medida que la computacin evoluciona la ambicin de los cientficos busca resolver problemas cada ves ms grandes y complejos. Esto implic la busqueda por parte de los investigadores de formas de aumentar la capacidad de almacenamiento y clculo por un lado, pero por otro la busqueda de estrategias que necesitaran almacenar menor cantid de datos y realizar menos clculos.

Una de las extrategias ms extendidas en la actualidad para subsanar los problemas mencionados anteriormente es la utilizacin de matrices dispersas. Es decir, al trabajar con matrices con muchos valores nulos utilizar estrategias que no necesiten almacenar los valores 0.

Lo que resta de este documento se estructura de la forma que se describe a continuación.

El Capítulo 2 introduce los conceptos generales de la las matrices dispersas.

El Capítulo 3 presenta un pequeo relevamiento histrico de la evolucin de las matrices dispersas. En particular, se profundiza en los trabajos pioneros en el rea y sus autores.

El Capítulo 4 presenta los principales formatos de almacenamiento de matrices disperasas.

El Capítulo 5 presenta en forma resumida las distintas conlusiones a las que se arribo durante este trabajo.

Capítulo 2

Conceptos Generales

2.1. Matrices dispersas

La primera aproximacin a los conceptos de matrices dispersas son aquellas matrices que disponen de muchos valores nulos que permitan almacenar dicha matriz utilizando menos memoria que en la forma convencional. En la forma convencional o matrices completas para almacenar una matriz de dimensiones mxn de elementos de puntos flotantes de doble precisin (8 bytes) se necesitan $8 * m * n$ bytes.

Si bien el concepto general es muy sencillo, no hay una definicin nica de matriz dispersa en la comunidad cintfica. Sin embargo en lo que si exite un concenso es que pensar en las matrices dispersas como forma de ahorro de almacenamiento es incompleto y que el ahorro tiene que incluir la cantidad de clculos.

En base a lo expuesto en los parrafos anteriores podemos definir que una matriz dispersa es aquella que posee una cantidad de ceros tal que no almacenarlos le permita sacar ventaja en almacenamieto y cantidad de clculo.

Algunas definiciones de autores son ...

Existen autores que mencionan la posibilidad de trabajar con matrices dispersa con respecto a otro valor, no necesariamente el 0.

Un punto importante que an resta por especificar es el porcentajes de valores nulos necesarios para tratar una matriz como dispersa o completa. Este punto no posee una respuesta concreta y va a depender del contexto, del tratamiento que se desee realizar con la matriz, el formato de almacenamiento utilizado, etc.

Capítulo 3

Evolucin Histrica

Nos pareci importante separar la evolucin del manejo de las matrices dispersas dependiendo de detrminados mojones histricos.

En particular dividimos en una primera etapa que abarca desde los comienzos hasta la primera conferencia dedicada dedicada al tema. Una segunda etapa que incluye los trabjos desde la primera conferencia hasta la segunda. Por ultimo, la etapa que abarca desde hasta nuestros das.

3.1. Primera etapa

Existe muy poca literatura disponible de esta primera etapa de la evolucin de la utilizacin de las matrices disperas. Varios autores [**?**, **?**] mencionan a XXXX como el 'padre'de las matrices dispersas

3.2. Segunda etapa

3.3. Tercera etapa

Capítulo 4

Estrategias de Almacenamiento

Las necesidades de memoria asociadas al almacenamiento de grandes matrices generalmente son una gran limitante informática. Sin embargo, con frecuencia las matrices que se obtienen al modelar problemas numéricos poseen gran cantidad de valores nulos, teniendo niveles de densidad (cantidad de valores no nulos / cantidad total de valores) sumamente bajos. Situación que se observa por ejemplo al utilizar técnicas de diferencias finitas y elementos fintos.

Una alternativa muy utilizada cuando se está trabajando con matrices de dimensiones importantes y bajo nivel de densidad[1] es la utilización de estrategias de almacenamiento disperso o ralos, es decir almacenar solamente aquellos coeficientes cuyo valor no sea cero.

Existen diversas estrategias para almacenar matrices dispersas, la elección por alguna de ellas se realiza sopesando varios factores como por ejemplo si la matriz es simétrica o no, si solo se requiere consultar los valores o se pretende modificarlos, el algoritmo con el que se desea trabajar, la mayor distancia de un elemento no nulo a la diagonal, el lenguaje de programación a utilizar, etc.

En lo que resta del Anexo se exponen algunas de las estrategias de almacenamiento disperso que normalmente se utilizan o que por su simplicidad u originalidad pueden resultar interesantes. El trabajo no intenta ser un estudio de los distintos métodos ni un relevamiento del estado del arte del tema. Si el lector está interesado en profundizar sobre el tema puede encontrar diversa literatura en el área. Algunos ejemplos son el trabajo de Pooch y Neider [3], el libro de Saad [4], el artículo de Duff [2] y la Tesis de Asenjo [1].

Las distintas estrategias de almacenamiento disperso se pueden agrupar en dos grandes clases: las estrategias estáticas y las estrategias dinámicas.

- Estrategias estáticas. Se utilizan normalmente cuando la estructura de la matriz (el patrón de coeficientes no nulos) no sufrirá grandes modificaciones. Por lo general, utilizan menos memoria y poseen velocidad de acceso a los coeficientes más rápido que las estrategias dinámicas. Entre las estrategias estáticas que se utilizan frecuentemente se destacan: el

[1] Si bien no existe un consenso sobre que es un buen nivel de densidad para trabajar con estrategias dispersas y el valor depende de muchas circunstancias, en gran parte de la literatura se especifica el valor de 0.2 de densidad como un buen nivel.

formato simple o coordenada - valor (coordinate format), el formato comprimido por fila (Compress Row Storage – CRS), el formato comprimido por columna (Compressed Column Storage – CCS), almacenamiento por diagonales (diagonal format – DIA), el formato Ellpack-itpack (ELL).

- Estrategias dinámicas. Se eligen cuando la frecuencia de modificación de la estructura de la matriz es alta ya que permiten de forma fácil agregar nuevos coeficientes. Entre las distintas estrategias dinámicas se pueden destacar: lista bidimensional doblemente enlazada (Linked List Row-Column Storage – LLRCS), listas enlazadas por filas (Linked List Row Storage – LLRS) y listas enlazadas por columna (Linked List Column Storage – LLCS).

Existe otra gran cantidad de estrategias generales de almacenamiento tanto estáticas como dinámicas, así como propuestas de almacenamiento disperso especialmente dise nadas para determinados tipos de matrices y/o para utilizar con determinados algoritmos.

A continuación se describen las distintas estrategias de almacenamiento disperso listadas anteriormente. Para facilitar la comprensión de las estrategias en algunos casos se muestra la aplicación de las estrategias descritas a la matriz A (que se puede observar en la Figura 4.1)

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

Figura 4.1: Matriz A ejemplo.

En todos los casos se supone que se almacenan matrices que poseen k coeficientes no nulos en punto flotante.

4.1. Estrategias estáticas

4.1.1. Formato simple

Este formato utiliza tres vectores para el almacenamiento de la matriz, un vector d de tipo punto flotante para los datos y dos vectores (f y c) de enteros para los índices (fila, columna). Cada valor no nulo de la matriz está asociado a 3 valores, uno por cada vector componente del formato.

En cuanto a las necesidades de memoria del método, si la matriz tiene k coeficientes no nulos entonces se utilizan k entradas de tipo punto flotante y $2 * k$ entradas de tipo entero para los índices.

En la Figura 4.1 se puede observar como quedaría la matriz A de la Figura 4.1. Si se desea acceder al valor $A(i, j)$ es necesario saber el p tal que $f(p) =$

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

Figura 4.2: Formato simple.

i y $c(p) = j$ y el valor buscado es $d(p)$. Se puede almacenar los coeficientes conservando algún orden (recorriendo por fila o columna) o en forma aleatoria.

Como ventaja de la estrategia de almacenamiento se puede destacar que es sumamente sencilla e intuitiva y que es igual de fácil el acceso por columna que por fila. En contra se destaca la necesidad de memoria, ya que si la densidad por fila/columna es mayor a uno (situación normal) utiliza más memoria que otros métodos.

4.1.2. CRS

El formato comprimido por fila (también conocida como CSR-Compressed Storage Row), utiliza al igual que el formato simple tres vector. Un vector d de tamaño k y tipo punto flotante en el que se almacenan los valores de los coeficientes. Otro vector c de tamaño k en el que se almacenan los números de columna de los elementos distintos de cero, por último, un vector f de tamaño $n+1$ siendo n la cantidad de filas de la matriz, en el cual se almacena la posición de la primera ocurrencia de cada fila.

$$
\begin{aligned}
d &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} \\
f &= \begin{pmatrix} 1 & 5 & 7 & 10 & 12 & 15 & 18 & 19 \end{pmatrix} \\
c &= \begin{pmatrix} 1 & 2 & 5 & 6 & 2 & 4 & 2 & 3 & 4 & 1 & 4 & 2 & 5 & 6 & 3 & 5 & 6 & 7 \end{pmatrix}
\end{aligned}
$$

Figura 4.3: Formato CRS.

En la Figura 4.1 se muestra como se representa la matriz A utilizando la estrategia CRS. Para acceder al valor $A(i, j)$ de la matriz A, se obtienen primero los índice $p_1 = f(i)$ y $p_2 = f(i + 1)$, posteriormente se busca el índice p tal que $p_1 \leq p < p_2$ y $f(p) = j$, luego se puede obtener el valor buscado accediendo a $d(p)$.

Si la matriz tiene más coeficientes no nulos que cantidad de filas, la estrategia CRS utiliza menos cantidad de memoria que el formato simple. Es necesario conocer todos las posiciones de los coeficientes para generar la estructura en forma eficiente, o dicho de otra forma es muy difícil agregar un nuevo valor a la estructura. Es fácil acceder a los elementos de una fila pero no a los de una columna.

4.1.3. CCS

El formato comprimido por columna es similar al CRS pero utilizando el vector comprimido para las columnas. En la Figura 4.1 se muestra como se aplica esta estrategia a la matriz A.

$$
\begin{array}{rl}
d = & \begin{pmatrix} 1 & 1 & 2 & 5 & 7 & 3 & 8 & 6 & 6 & 9 & 2 & 3 & 4 & 7 & 4 & 5 & 8 & 9 \end{pmatrix} \\
f = & \begin{pmatrix} 1 & 4 & 1 & 2 & 3 & 5 & 3 & 6 & 2 & 3 & 4 & 1 & 5 & 6 & 1 & 5 & 6 & 7 \end{pmatrix} \\
c = & \begin{pmatrix} 1 & 3 & 7 & 9 & 12 & 15 & 18 & 19 \end{pmatrix}
\end{array}
$$

Figura 4.4: Formato CCS.

Se mantienen las necesidades de memoria del formato CRS y en cuanto a las características de acceso se invierten los conceptos con respecto a filas y columnas.

4.1.4. DIA

Cuando la matriz a almacenar es de banda, se puede utilizar estrategias de almacenamiento por diagonales. Entonces se almacena la matriz original en una matriz rectangular de tama no $n * d$ siendo n la dimensión de la matriz y d la cantidad de diagonales.

Si se desea almacenar una matriz que posee d diagonales no consecutivas, una variante de formato es el Diagonal Storage Format que utiliza una matriz de tamaño $n*d$ en la que se almacenan las d diagonales y otro vector de tamaño d en el que se especifica el offset de cada diagonal con la diagonal principal.

4.1.5. ELL

En el formato Ellpack-itpack almacena una matriz con m elementos por fila en dos matrices de tama no $n * m$. En la primera se tienen los valores de los coeficientes mientras que en la segunda se puede obtener el offset de cada elemento a la diagonal.

4.2. Estrategias dinámicas

Las distintas estrategias dinámicas generalmente utilizan punteros para implementar las estructuras, por lo tanto para implementar las variantes es necesario utilizar lenguajes que soporten allocate dinámico por ejemplo C y Fortran 90.

4.2.1. LLRCS

En esta estrategia se utiliza una multiestructura bidimensional. Se dispone de dos vectores de tamaño igual a la cantidad de filas y columnas, en cada entrada se dispone de un puntero para recorrer las distintas entradas por fila en un vector o por columna en el otro.

La memoria que se necesita para implementar la estrategia LLRCS, si k es la cantidad de coeficientes no nulos, se necesitan k posiciones de punto flotante, $4 * k$ punteros, k enteros, más la memoria necesaria para almacenar los dos vectores ($2 * n$ punteros).

Para acceder al valor $A(i, j)$ se puede recorrer la lista de la entrada i por el vector de filas buscando el valor cuya columna sea j, o recorrer la lista de la entrada j del vector de columnas hasta encontrar la fila i.

4.2.2. LLRS

En esta estrategia se utiliza una estructura unidimensional, en la cual se emplea un vector de tama no igual a la cantidad de filas y de cada posición del vector se puede obtener la lista de entradas de esa fila.

En cuanto a memoria se necesitan k posiciones de punto flotante, k enteros, k punteros más el vector de entrada (n punteros).

4.2.3. LLCS

Al igual que en la propuesta anterior, en la estrategia LLCS se utiliza una estructura unidimensional, pero el vector base de la estructura es de tama no igual a la cantidad de columnas y de cada posición del vector se puede obtener la lista de coeficientes no nulos de esa columna.

La estrategia LLCS posee las mismas necesidades de memoria que la LLRS.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

1.- Item 1

2.- Item 2

2.1.- Item 2.1

2.2.- Item 2.2

2.2.1.- Item 2.2.1

2.2.2.- Item 2.2.2

2.2.3.- Item 2.2.3

3.- Item 3

4.- Item 4

5.2. Trabajo Futuro

Bibliografía

[1] R. Asenjo. Factorización lu de matrices dispersas en multiprocesadores. Ph.D. Thesis, Technical Report UMA-DAC-97/32, Universidad de Malaga – España, 1997.

[2] I. S. Duff. A survey of sparse matrix reserch. In *In Proceeding of the IEEE*, pages 500 – 535, New York, 1977. Prentice Hall.

[3] Udo W. Pooch and Al Nieder. A survey of indexing techniques for sparse matrices. *ACM Comput. Surv.*, 5(2):109–133, 1973.

[4] Youcef Saad. Sparskit: a basic tool kit for sparse matrix computations. Version 2, 1994.