Cálculo en paralelo de la homología cúbica con gridMathematica

Jónathan Heras, Luis Javier Hernández, María Teresa Rivas,
Eduardo Sáenz de Cabezón, Julio Rubio

Departamento de Matemáticas y Computación. Universidad de la Rioja.

Resumen

Presentamos un pequeño informe sobre las experiencias realizadas utilizando gridMathematica para el cálculo de la homología cúbica. Los experimentos fueron realizados sobre un cluster de 8 equipos en la Universidad de La Rioja. Los resultados obtenidos pueden considerarse alentadores, pese a que será necesaria más investigación para sacar todo el partido posible de la paralelización con gridMathematica.

1. Motivación

El objetivo de este proyecto era evaluar gridMathematica en un problema concreto de cálculo científico. Se ha elegido un método matemático que aporta nuevas formas de estudio en la minería de datos que se pueden aplicar a diversos ámbitos científicos y tecnológicos. Una de las piezas centrales de este proceso es el cálculo de la homología de complejos cúbicos. Se trata de un problema de gran interés y además su magnitud y complejidad lo hacen adecuado para abordarlo mediante el cálculo en paralelo.

El software gridMathematica fue financiado a cargo del proyecto de investigación "Programación con componentes fiables para el Cálculo Simbólico" (referencia: ANGI2005/19), concedido por el Gobierno de la Rioja. Para la realización de este trabajo hemos contado con todo el apoyo del personal de Addlink (más en concreto, de Antonio Molina y de Juan Antonio Rubio), no sólo en lo relativo a la distribución del producto, sino también en la asesoría en aspectos más tecnológicos.

2. Homología cúbica

En esta sección explicaremos los conceptos necesarios para llegar a comprender en qué consiste el cálculo de la homología cúbica.

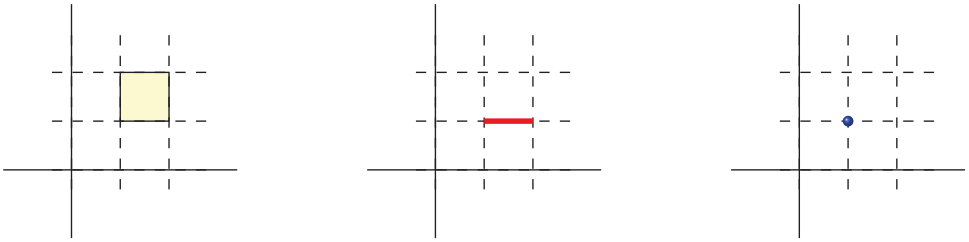

Figura 1: Ejemplos de cubos elementales en $\mathbb{R}^2$

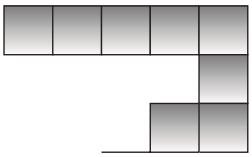

Figura 2: Complejo Cúbico $X \subset \mathbb{R}^2$

Llamaremos *intervalo elemental* a un intervalo de $\mathbb{R}$ de la forma $I = [l, l+1]$ o $I = [l, l]$ con $l \in \mathbb{Z}$. En el último caso, diremos que es *degenerado* (denota un único entero en $\mathbb{R}$). A un producto finito de intervalos elementales $Q = I_1 \times \ldots \times I_d \subset \mathbb{R}^d$ se le denomina *cubo elemental*. La figura 1 muestra gráficamente algunos cubos elementales en $\mathbb{R}^2$. Ahora, dado $X \subset \mathbb{R}^d$, diremos que X es un *complejo cúbico* si es una unión finita de cubos elementales. Un ejemplo de un complejo cúbico aparece en la figura 2.

Dentro de un complejo cúbico X, el modo en que los cubos de dimensión k se relacionan con los cubos de dimensión $k-1$ puede ser reflejado por medio de una matriz de incidencia. Se trata de una matriz con entradas 0, 1 y -1. Un cero señala que el k-cubo no tiene como cara el $(k-1)$-cubo correspondiente; si hay incidencia el valor $+1$ ó -1 depende de una orientación previa elegida sobre cada uno de los cubos de X. La figura 3 muestra un complejo cúbico muy sencillo, junto a su matriz de incidencia.

Sin definir técnicamente qué es la homología cúbica de X, baste señalar aquí que su cálculo puede realizarse a través de sus matrices de incidencia, mediante

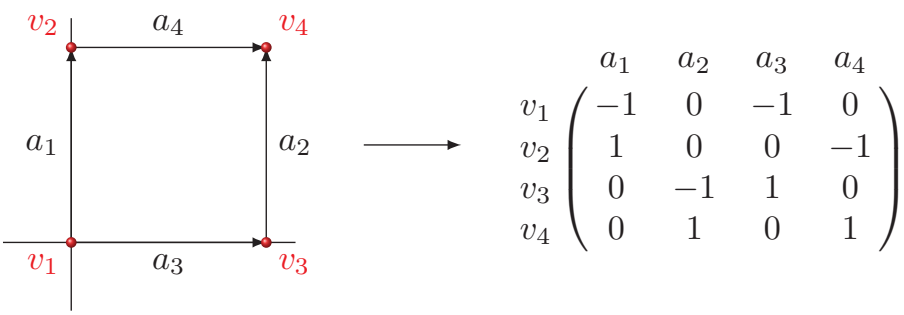

Figura 3: Complejo Cúbico y su Matriz de Incidencia

la reducción a Forma Normal de Smith de cada una de las matrices consideradas; es decir, se trata de un proceso de diagonalización de matrices de enteros (para la definición de los grupos de homología y los algoritmos básicos para el cálculo de la Forma Normal de Smith pueden consultarse, por ejemplo, [1] o [2]).

El interés de la homología cúbica frente a otras más clásicas, como la homología singular o la simplicial (véanse de nuevo [1] o [2]), reside en sus aplicaciones; entre ellas, destacan las de minería de datos. Una nube de datos suele representarse, o aproximarse, por medio de un complejo cúbico en algún espacio euclídeo $\mathbb{R}^d$. El cálculo de la homología de dicho conjunto puede permitir inferir propiedades sobre la densidad, los agrupamientos (clusters), etc. de la nube de datos inicial. Sin embargo, uno de los problemas que aparece en el uso de esta técnica de aplicaciones homológicas a la minería de datos (parcialmente desarrollada por el grupo de Topología de la Universidad de La Rioja en el proyecto ANGI2005/10) es que si el número de variables aumenta, caso frecuente en datos tecnológicos obtenidos mediante sensores, el tamaño y número de matrices es elevado y su cálculo parece adecuado para ser abordado mediante procesos de paralelización. Para más información sobre estas aplicaciones, puede consultarse la referencia [3].

3. El entorno de trabajo

Para la realización de este trabajo se utilizaron distintos programas de Wolfram:

Mathematica v6.0: Este conocido entorno de desarrollo para cálculo simbólico ha sido utilizado para la creación del paquete de uso local.

gridMathematica v2.0: Es una solución para cálculo en paralelo desarrollado sobre Mathematica.

Wolfram Workbench v1.0: Es un entorno de desarrollo para la creación de software con la tecnología Mathematica. El principal uso que le dimos fue el de crear un entorno de pruebas para comprobar el correcto funcionamiento de nuestros algoritmos.

Se desarrolló también una pequeña interfaz gráfica, de tipo calculadora (véase la figura 4), para interactuar con nuestros programas. Para ello se utilizó el paquete *GUIKit* de Mathematica.

Respecto al hardware, se utilizó un *cluster* de 8 ordenadores, numerados del 2 al 9, haciendo el computador grid6 las funciones de maestro, y también de puente entre la red de la universidad y el grid. El acceso a este grid se puede hacer tanto desde la red interna de la Universidad de la Rioja como desde el exterior utilizando una VPN (Virtual Private Network) para acceder a ella. La figura 5 muestra la estructura de nuestro grid.

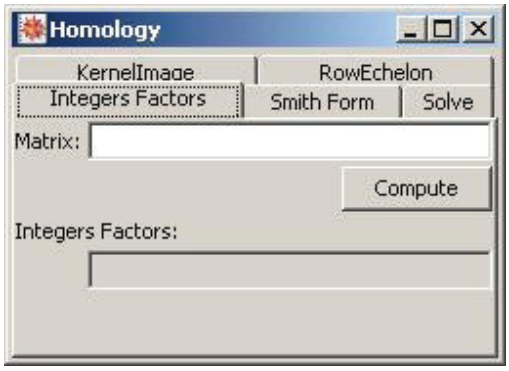

Figura 4: Interfaz de usuario

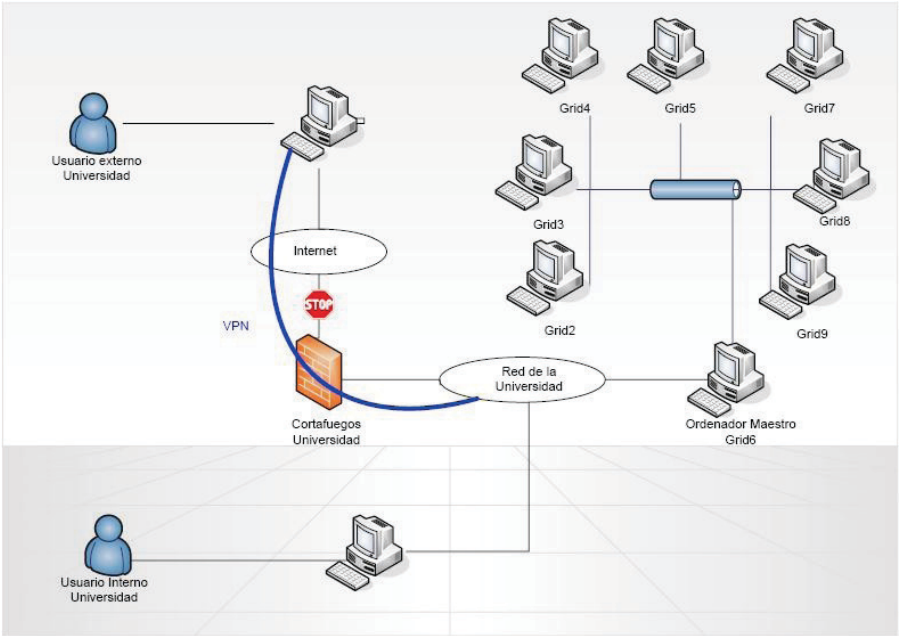

Figura 5: Estructura Hardware

4. Aproximaciones para paralelizar el algoritmo

Tanto el cálculo eficiente de la forma normal de Smith (véase, por ejemplo, [4]), como su paralelización (véanse [5] o [6]) han recibido mucha atención en el ámbito de la investigación en Cálculo Simbólico. Siendo nuestro objetivo una evaluación de las capacidades de gridMathematica más que la obtención de programas competitivos en cuanto a su eficacia respecto a otras aproximaciones, hemos preferido implementar nosotros mismos una versión simple local, como referencia, y a continuación dejarnos guiar por una aplicación directa de varios métodos adaptados al entorno de gridMathematica.

Hemos considerado 5 métodos distintos para paralelizar nuestros algoritmos, utilizando las distintas opciones que aparecen documentadas en [7]:

ParallelEvaluate: gridMathematica nos proporciona este comando genérico, que sirve para ejecutar una instrucción en paralelo utilizando los ordenadores que nosotros indiquemos.

Paralelizando a bajo nivel: Los algoritmos más básicos de nuestro paquete sirven para realizar operaciones sobre filas y columnas de una matriz. En un nivel superior tenemos otras funciones basadas en la aplicación de esos algoritmos básicos a tres matrices construidas ad-hoc. Como el trabajo sobre cada una de estas tres matrices es independiente al que se realiza en las demás, podemos distribuirlo en los ordenadores, utilizando simplemente los comandos *Send* y *Receive*.

Algoritmo MinNonZero: Este algoritmo se encarga de encontrar el menor elemento no nulo de una matriz (una tarea que se emplea con muchísima frecuencia, cuando se elige un *pivote* para hacer ceros en el proceso de diagonalización). Para ello busca primero el menor elemento no nulo de cada fila. La búsqueda en cada una de las filas es totalmente independiente de la búsqueda en las demás, por lo que es una tarea paralelizable. Para enviar las distintas filas a los ordenadores del grid creamos un vector que en cada una de sus componentes tenga un link a un kernel remoto. Este vector lo recorreremos de forma que cuando hemos utilizado todos los ordenadores volvemos al principio del vector. De este modo, experimentamos con el reparto directo de la carga en los distintos nodos.

Mejorando Smith: Dada una matriz, el algoritmo *SmithForm* utiliza el método *PartSmithForm* para hacer ceros (por medio de operaciones con filas y columnas) por debajo y a la derecha del elemento pivote de la matriz dada. Si dividimos la matriz que nos han dado en submatrices que contengan la fila del elemento pivote y una de las otras filas, y les aplicamos el algoritmo *PartSmithForm* a todas ellas, la matriz buscada se obtiene agrupando adecuadamente los resultados obtenidos. En esta ocasión utilizaremos el *scheduler* para enviar las distintas submatrices. El *scheduler*, incluido en gridMathematica, se encarga de distribuir las distintas instrucciones por los ordenadores disponibles del grid de forma transparente para nosotros. A cada instrucción que se añade al *scheduler*, mediante el comando *Queue* se le asigna un PID (Identificador de Proceso), el cual se utilizará para recoger los resultados con el comando *Wait*.

algoritmo \ matriz	5×5	10×10	15×15	30×30	50×50
SmithForm con 6	0,3568	3,4022	11,2505	146,0604	358,3147
SmithForm con 4	0,4854	3,3742	11,6476	190,4567	597,1856
SmithForm con 2	0,4170	2,9982	11,0818	195,7012	698,5674
SmithForm local	0,1715	0,8581	2,8241	29,3964	228,3113

Tabla 1: *ParallelEvaluate* aplicado a *SmithForm*

Smith en listas: La idea fundamental de este método es que tenemos una lista de matrices de las cuales queremos calcular su Forma Normal de Smith, pero de manera alterna; es decir, primero todas las impares y luego todas las pares (este algoritmo está basado en ciertas ideas algebraicas que, para no abusar de la paciencia del lector, no explicaremos aquí). Como el cálculo de la Forma Normal de Smith de cada matriz es independiente del cálculo de las demás, se pueden distribuir las matrices por los ordenadores del grid. Para hacer esto utilizamos como en el caso anterior el *scheduler*.

5. Resultados Experimentales

Los experimentos fueron realizados generando aleatoriamente matrices. Los tiempos (en segundos de CPU) de cálculo en cada columna se refieren a una misma matriz.

La tabla 1 muestra los tiempos obtenidos al aplicar el comando *ParallelEvaluate* al algoritmo *SmithForm*, utilizando distinto número de ordenadores (*SmithForm* con "n", significa que se emplean "n" ordenadores del grid) y también distintos tamaños de matrices ($m \times m$, significa m filas y m columnas). Como se puede ver, los resultados no son muy alentadores y en todos los casos, utilizando cualquier número de ordenadores, los tiempos de ejecución empeoran por encima del 50 %. Estos resultados son esperables, pues siendo *ParallelEvaluate* un comando genérico, no dispone del conocimiento experto necesario para paralelizar inteligentemente el algoritmo. Los costes (*overhead*) de distribución de tareas explicarían el deterioro de la eficiencia.

La tabla 2 hace referencia a los resultados que se obtienen sobre el algoritmo *SmithForm* al aplicar lo que hemos llamado *paralelización a bajo nivel*. En este caso vemos que se producen mejoras en los tiempos, aproximadamente entre un 15 y un 25 %.

En la tabla 3, se muestran los resultados obtenidos al paralelizar el algoritmo *MinNonZero*. Como podemos ver, dichos resultados son muy negativos, ya que los tiempos de ejecución aumentan muchísimo, sobre todo conforme el tamaño de las matrices va aumentando. La explicación que podemos dar a estos malos resultados es que la búsqueda del menor elemento no nulo de una fila consume pocos recursos, y que por tanto la transferencia de la información entre ordenadores (overhead)

Versión \ matriz	5×5	10×10	15×15	30×30	50×50
Versión Original	1,6006	5,3456	17,7105	240,4157	531,4841
Versión Paralelo	1,1600	4,3157	13,5878	180,2474	451,4861

Tabla 2: *Send-Receive* en *SmithForm*

Ejecución \ matriz	10×10	25×25	50×50	100×100	200×200
Local	0,0116	0,0525	0,1460	0,4849	1,7776
Paralelo	0,0204	0,0682	0,2586	1,4369	12,8041

Tabla 3: Resultados Paralelizando *MinNonZero*

consume más tiempo que el que se tarda en ejecutar el algoritmo en local.

Los resultados obtenidos al aplicar el método llamado *Mejorando Smith* aparecen en la tabla 4. En este caso, a medida que el tamaño de las matrices aumenta, son bastante positivos, llegando a producirse una mejora de hasta un 30 %.

Por último, las tablas 5, 6, 7, 8, hacen referencia a los resultados que se obtienen al aplicar el método de *Smith en listas* con distintos tamaños de listas y de matrices. Los "*" que aparecen en las tablas significan que Mathematica 6.0 no puede terminar el cálculo, por agotamiento de recursos. Como se puede observar, los beneficios de la paralelización en este caso son manifiestos.

6. Conclusiones y trabajo de futuro

La conclusión principal que se obtiene del trabajo es que, usado convenientemente, gridMathematica es un útil instrumento para obtener mejoras en el tiempo de ejecución del cálculo en Álgebra Homológica. Por supuesto, y como es bien conocido en la disciplina de la Programación en Paralelo, una inadecuada distribución de tareas puede deteriorar el rendimiento, como ha quedado reflejado en la anterior tabla 3.

Las líneas de trabajo futuro deberían estar basadas en la aplicación de las ideas de los artículos [5], [6] y [4] para obtener versiones ejecutables sobre gridMathematica del cálculo de la homología cúbica, que puedan competir con las documen-

modo \ matriz	5×5	10×10	15×15	20×20	30×30	50×50
Local	0,186	0,940	2,509	5,142	21,49	107,44
Paralelo	0,306	0,973	1,959	3,799	18,24	89,45

Tabla 4: *SmithParalelo* vs *Smith*

Ejecución \ matriz	5×5	10×10	15×15	30×30	50×50
Local	0,8481	4,5626	14,9649	110,3033	$*$
Paralelo	0,1743	0,7611	2,3116	35,5071	212,1252

Tabla 5: *Smith en listas* de tamaño 5

Ejecución \ matriz	5×5	10×10	15×15	30×30
Local	1,803	7,799	23,261	193,3125
Paralelo	1,4549	4,4061	8,6070	78,7754

Tabla 6: *Smith en listas* de tamaño 10

Ejecución \ matriz	5×5	10×10	15×15	30×30
Local	3,9532	18,2269	50,8457	318,8695
Paralelo	2,4304	12,8908	34,8918	247,6956

Tabla 7: *Smith en listas* de tamaño 15

Ejecución \ matriz	5×5	10×10	15×15	30×30
Local	7,2766	34,4898	235,4014	$*$
Paralelo	6,8755	32,0610	95,2244	678,4982

Tabla 8: *Smith en listas* de tamaño 25

tadas en la literatura.

Referencias

[1] M. Mrozek, T. Kacynski y K. Mischaikow. *Computational Homology*. Springer- Verlag, 2004.

[2] C.R.F. Maunder. *Algebraic Topology*. Dover, 1997.

[3] L.J. Hernández. *Orografía homológica cúbica para la minería de datos*. http://www.unirioja.es/cu/luhernan/datamining/index.html

[4] G. Villard, J. Dumas y B. D. Saunders. *On efficient sparse integer matrix Smith Normal Form computations*. Journal of Symbolic Computation 32 (2001) 71–79.

[5] G. Jägeer. *Parallel algorithms for computing the Smith Normal Form of large matrices*. Lecture Notes in Computer Science 2840 (2003) 170–179.

[6] W. Wilhelmi y I. Neumann. *A parallel algoritm for achieving the Smith Normal Form of an integer matrix*. Parallel Computing 22 (10) (1996) 1399–1412.

[7] R. Maeder. *Parallel Computing Toolkit, Unleash the power of parallel computing*. Wolfram Research, 2007.

[8] Stephen Wolfram. *The Mathematica Book 5^a Edicion*. Wolfram Media, 2003.