Instituto Tecnológico de Costa Rica

Vicerrectoría de Investigación y Extensión

Informe Final

Proyecto:

Técnicas de precondicionamiento en paralelo

Índice

Índice de cuadros

Técnicas de precondicionamiento en paralelo

Código: # 5402-1801-0480

MSc. Geovanni Figueroa Mata

Escuela de Matemática

Instituto Tecnológico de Costa Rica

email: gfigueroa@itcr.ac.cr

MSc. Luis E. Carrera Retana*

Escuela de Matemática

Instituto Tecnológico de Costa Rica

email: lecarrera@itcr.ac.cr

29 de julio de 2014

*Coordinador del proyecto

Resumen

El objetivo principal de la investigación fue diseñar e implementar en paralelo una técnica de precondionamiento con el fin de resolver sistemas de ecuaciones lineales provenientes de la solución numérica de ecuaciones diferenciales parciales; la característica común de estos sistemas lineales es su gran tamaño y el hecho de que la matriz de coeficientes asociada es rala.

Se diseño e implementó en paralelo una técnica de precondicionamiento, basada en la factorización incompleta LU (ILU), y se aplicó a problemas obtenidos de la colección de matrices ralas de la Universidad de Florida.

El precondicionador fue probado en matrices no simétricas y matrices simétricas no definidas positivas. Los mejores resultados se obtuvieron en el caso de las matrices simétricas no definidas positivas.

Abstract

The investigation's main objective was to design and implement a parallel preconditioner technique which could help solve linear equation systems required for the numerical solution of partial differential equations, systems whose common characteristics are their big size and the fact that the related coefficient matrix is sparse.

The parallel preconditioner technique designed and implemented is based in the incomplete LU factorization (ILU), and the preconditioner was applied to problems found in The University of Florida Sparse Matrix Collection.

The preconditioner was used in unsymetric matrices and in non-positive definite symetric matrices, and the best results were found in cases of non-positive definite symetric matrices.

Palabras claves: matrices ralas, técnicas de precondicionamiento, métodos iterativos para sistemas lineales, programación paralela, factorización ILU.

1. Introducción

Muchos procesos industriales y físicos son modelados por medio de ecuaciones diferenciales parciales, cuya solución al final requiere de la solución de sistemas de ecuaciones lineales. La característica común de estos sistemas lineales es su gran tamaño y el hecho de que la matriz de coeficientes asociada es rala. Por otro lado, la convergencia de la mayoría de los métodos iterativos depende de las propiedades espectrales de la matriz de coeficientes del sistema lineal . Un método que utilice una matriz de coeficientes mal condicionada va a requerir de muchas iteraciones e incluso tal vez no se alcance la convergencia, lo cual no mejoraría el inconveniente de aplicar métodos directos, pues el método iterativo podría llegar a ser más lento que un método directo. Por esta razón, se usan técnicas de preconcicionamiento que tienen por objetivo reducir el condicionamiento de la matriz de coeficientes del sistema lineal, logrando de esta forma reducir el número de iteraciones necesarioas para alcanzar la convergencia.

El problema de resolver un sistema lineal de la forma $Ax = b$ es central en la computación científica. La factorización completa LU, por ejemplo, es un método directo, pero dichos métodos son imprácticos si A es grande y rala, porque la factorización resultante puede dar una matriz densa.

Los métodos iterativos, en contraste con los métodos directos, generan una secuencia de soluciones aproximadas $\mathbf{x}_i$, y esencialmente utilizan a la matriz A solamente para productos matriz-vector, lo cual puede ser paralelizado de manera muy eficiente.

1.1. Precondicionadores

Un precondicionador M es una matriz no singular que transforma el sistema $Ax = b$ en un nuevo sistema $M^{-1}Ax = M^{-1}b$, donde el radio espectral del nuevo sistema es menor que el del sistema original. En estas condiciones, la solución del sistema transformado es equivalente a la solución del sistema original, pero sus propiedades espectrales permiten alcanzar más rápidamente la convergencia. Así, el objetivo del precondicionador es reducir el número de iteraciones requerido para lograr la convergencia, claro está sin incrementar significativamente la cantidad de cálculos por iteración.

Ciertos precondicionadores necesitan una pequeña fase de construcción y otros pueden necesitar un trabajo sustancial. Hay varias opciones para construir el nuevo sistema. Por ejemplo, tomar una matriz no singular M y formar el sistema

$$M^{-1}Ax = M^{-1}b$$

Otra forma muy usada es tomar $M = M_1M_2$, con M_1 y M_2 matrices no singulares y formar el sistema

$$M_1^{-1}AM_2^{-1}(M_2x) = M_1^{-1}b$$

A las matrices M_1 y M_2 se les llama precondicionadores izquierdo y derecho respectivamente. En este caso, el esquema para precondicionar un método iterativo es el siguiente:

- Haga $b \leftarrow M_1^{-1}b$.
- Aplique el método iterativo sin precondicionador al sistema

$$M_1^{-1}AM_2^{-1}y = b$$

- Calcule $x \leftarrow M_2^{-1}y$

Observe que si $M = A$, el algoritmo converge en una iteración, lo cual nos hace concluir que mientras mejor aproxime M a la matriz A, más rápida será la convergencia. En este trabajo M se calcula por medio de la factorización LU incompleta.

Existe una gran variedad de métodos iterativos para resolver sistemas de ecuaciones lineales los clásicos como el de Jacobi, Gauss-Seidel, sobrerelajación sucesiva (SOR) ([1]) y los llamados métodos de proyección como el de residuo mínimo generalizado (GMRES), el gradiente biconjugado y el gradiente conjugado (GC) ([11], [8], [9]). En general todos estos métodos presentan una convergencia relativamente lenta por lo que es necesario introducir mejoras en el esquema numérico con el fin de acelerar la convergencia.

Los precondicionadores clásicos utilizan una matriz M no singular que transforma el sistema $Ax = b$ en un nuevo sistema $M^{-1}Ax = M^{-1}b$ el cual tiene un radio espectral menor que el del sistema original lo cual acelera la convergencia del método iterativo. Dentro de éstos los más conocidos son las factorizaciones incompletas como la LU y la de Cholesky ([4]).

1.2. Factorización incompleta LU

Una clase importante de precondicionadores se obtiene de la factorización LU, la cual se puede calcular para matrices arbitrarias. Para las matrices ralas, se puede aplicar un procedimiento denominado factorización incompleta, es decir si A es una matriz rala, los factores L y U usualmente no tienen el mismo patrón de dispersión de la matriz A, y como la idea es conservar este patrón, entonces se descartan los elementos diferentes de cero en aquellas posiciones donde la matriz A tenía un cero (fill-elements). De esta manera se obtiene una factorización aproximada: $A \approx LU$.

La factorización incompleta LU existe si A es invertible y diagonalmente dominante o si A es una M-matriz, es decir, si A es una matriz con elementos no positivos fuera de la diagonal principal, invertible y $A^{-1} \geq 0$.

El siguiente teorema establece la existencia y unicidad de la factorización incompleta LU para M-matrices.

Teorema 1.1 *Si A es una M-matriz de tamaño $n \times n$ entonces para cada subconjunto P de $(i,j)|i \neq i, i, j = 1, 2, \cdots, n$ existe una matriz triangular inferior L con unos en la diagonal y una matriz triangular superior U tal que $A = LU - R$, con $r_{ij} = 0$ si $(i,j) \in P$. Las matrices L y U son únicas y la descomposición $A = LU - R$ es tal que LU es no singular, $(LU)^{-1} \geq 0$ y $LU \geq A$.*

A la luz del teorema anterior, si escogemos $M = LU$, donde L es triangular inferior y U es triangular superior, el sistema $Mx = y$ (instrucción del algoritmo del gradiente conjugado precondicionado) se resuelve de la forma usual, es decir, por sustitución hacia atrás y hacia adelante. La factorización incompleta que se obtiene descartando todos los fill-elements en el proceso LU se conoce como factorización incompleta de nivel cero y se denota $ILU(0)$.

En la factorización $ILU(\tau)$ la regla de eliminación usada es la siguiente:

1. Un elemento w_k, será reemplazado por cero si es menor que una tolerancia τ_i la cual se obtiene multiplicando por τ la norma del vector original de la i-ésima fila.

2. Para cada vector fila w se hace cero cualquier elemento cuya magnitud sea menor que una tolerancia τ_i. Luego se consideran sólo los p elementos mayores de la fila correspondiente, tanto en L como en U, aunque en ésta última además se incluye siempre el elemento de la diagonal.

1.3. Problema a investigar

Dada una matriz $\mathbf{A}$ de tamaño $n \times n$ y un vector $\mathbf{b} \in I\!R$ se quiere determinar el vector $x \in I\!R$ tal que $Ax = b$, aplicando un método de proyección en paralelo junto con una técnica de precondicionamiento que permita acelerar la convergencia.

Es decir, se quiere investigar sobre la mejor combinación método-precondicionador que permita resolver eficientemente y en paralelo un sistema de ecuaciones lineales proveniente de una ecuación diferencial en derivadas parciales, ya que en esta área es donde surge este tipo de matrices, debido a la discretización que se hace del dominio para su solución numérica.

En este proyecto se analizó la formulación numérica del problema, su paralelización y el uso de precondicionadores paralelos que disminuyan significativamente el tiempo de ejecución. Finalmente, se utilizaron los algoritmos desarrollados para resolver un conjunto de problemas aplicados.

2. Objetivos

1. **Objetivo general**

   Diseñar e implementar técnicas de precondicionamiento en paralelo.

2. **Objetivos específicos**

   a) Investigar sobre la existencias de las diferentes técnicas de precondicionamiento.

   b) Analizar cuáles de las técnicas de precondicionamiento estudiadas son suceptibles de ser paralelizadas y aplicadas a métodos iterativos para la resolución de sistemas lineales.

   c) Diseñar un algoritmo en paralelo para alguna de estas técnicas de precondicionamiento.

   d) Implementar utilizando programación en paralelo el algoritmo diseñado.

   e) Probar y depurar la técnica de precondicionamiento desarrollada sobre algunos sistemas de ecuaciones lineales de prueba.

3. Metodología

Se realizó una investigación bibliográfica sobre la factorización de Cholesky (la cual se aplica a matrices simétricas definidas positivas) y diferentes técnicas de factorización incompleta LU, como $ILU(0)$, $ILU(p)$, $ILUT$, $MILU$ ([11], [4], [9]).

Después de analizar los algoritmos para estas técnicas, se decidió diseñar un algoritmo en paralelo para $ILU(\tau)$, debido a que es aplicable tanto a matrices simétricas no definidas positivas (donde la factorización de Cholesky no puede ser aplicada) como a matrices no simétricas. Otra de las razones para elegir esta técnica es que la variación del parámetro τ permite ajustar el tamaño de la factorización LU.

Inicialmente se implementó el algoritmo $ILU(\tau)$ como aparece en la literatura ([11]), pero durante el proceso de prueba, se observó que este método era impráctico para matrices ralas de gran tamaño, porque primero realiza la factorización completa de la fila, antes de eliminar elementos "pequeños", lo cual en casos de más de 1 000 000 de filas, hacía que el proceso se hiciera muy lento.

Por esa razón, se tuvo que pensar en una modificación para eliminar elementos pequeños durante el proceso de factorización, sin haber finaliza la factorización completa de la fila, modificación que se explica en la Sección 4.2.

Con el objetivo de ejecutar el precondicionador sobre sistemas lineales, se diseño e implementó en paralelo el algoritmo BiCGStab, el cual, en principio, puede ser aplicado a cualquier tipo de sistema lineal.

Finalmente, de la colección de matrices ralas de la Universidad de Florida se seleccionó un conjunto de matrices simétricas no definidas positivas y otras no simétricas, a las cuales se les aplicó los algoritmos implementados.

4. Diseño e Implementación

Los algoritmos se programaron en C, con la biblioteca OpenMP para realizar el proceso de paralelización. Se compiló el código con gcc 4.9.1, en una máquina DELL con sistema operativo LINUX,

- Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor DRAM Controller
- 8Gb de memoria ram
- 4 procesadores Intel(R) Core(TM) i7-3770 3.40GHz con multi-hilos simultáneos (el sistema simula 8 procesadores)

4.1. Solución de Sistemas Iterativos

Dado que el interés en este proyecto era trabajar con matrices no-simétricas, entonces se implementó el algoritmo Gradiente Bi-Conjugado Estabilizado, conocido como *BiCGStab* por sus siglas en inglés, el cual se muestra a continuación:

Algoritmo BiCGStab

$\mathbf{r}_0 \leftarrow \mathbf{b} - A\mathbf{x}_0$; $\hat{\mathbf{r}}_0$ arbitrario
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$
Para $i \leftarrow 1, 2, 3, \ldots$ hasta converger:
    $\alpha \leftarrow (\mathbf{r}_{i-1}, \hat{\mathbf{r}}_0)/(A\mathbf{p}_{i-1}, \hat{\mathbf{r}}_0)$
    $\mathbf{s}_i \leftarrow \mathbf{r}_{i-1} - \alpha A\mathbf{p}_{i-1}$
    $\omega \leftarrow (A\mathbf{s}_i, \mathbf{s}_i)/(A\mathbf{s}_i, A\mathbf{s}_i)$
    $\mathbf{x}_i \leftarrow \mathbf{x}_{i-1} + \alpha\mathbf{p}_{i-1} + \omega\mathbf{s}_i$
    $\mathbf{r}_i \leftarrow \mathbf{s}_i - \omega A\mathbf{s}_i$
    $\beta \leftarrow (\alpha/\omega)(\mathbf{r}_i, \hat{\mathbf{r}}_0)/(\mathbf{r}_{i-1}, \hat{\mathbf{r}}_0)$
    $\mathbf{p}_i \leftarrow \mathbf{r}_i + \beta(\mathbf{p}_{i-1} - \omega A\mathbf{p}_{i-1})$

En la implementación del algoritmo se paralelizaron todos los productos matriz-vector y productos punto vector-vector.

Se implementó el algoritmo BiCGSTab en paralelo, utilizando un precondicionador de factorización LU incompleta.

Dadas las matrices L y U tales que $LU = K \approx A$, se implementa el algoritmo BiCGStab precondicionado. Para calcular el vector $\mathbf{x} = K^{-1}\mathbf{v}$, entonces se resuelven los sistemas $L\mathbf{y} = \mathbf{v}$ y $U\mathbf{x} = \mathbf{y}$.

Algoritmo BiCGStab precondicionado

$\mathbf{r}_0 \leftarrow K^{-1}(\mathbf{b} - A\mathbf{x}_0)$; $\hat{\mathbf{r}}_0$ arbitrario
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$
Para $i \leftarrow 1, 2, 3, \ldots$ hasta converger:
    $\alpha \leftarrow (\mathbf{r}_{i-1}, \hat{\mathbf{r}}_0)/(K^{-1}A\mathbf{p}_{i-1}, \hat{\mathbf{r}}_0)$
    $\mathbf{s}_i \leftarrow \mathbf{r}_{i-1} - \alpha K^{-1}A\mathbf{p}_{i-1}$
    $\omega \leftarrow (K^{-1}A\mathbf{s}_i, \mathbf{s}_i)/(K^{-1}A\mathbf{s}_i, K^{-1}A\mathbf{s}_i)$
    $\mathbf{x}_i \leftarrow \mathbf{x}_{i-1} + \alpha\mathbf{p}_{i-1} + \omega\mathbf{s}_i$
    $\mathbf{r}_i \leftarrow \mathbf{s}_i - \omega K^{-1}A\mathbf{s}_i$
    $\beta \leftarrow (\alpha/\omega)(\mathbf{r}_i, \hat{\mathbf{r}}_0)/(\mathbf{r}_{i-1}, \hat{\mathbf{r}}_0)$
    $\mathbf{p}_i \leftarrow \mathbf{r}_i + \beta(\mathbf{p}_{i-1} - \omega K^{-1}A\mathbf{p}_{i-1})$

Para la implementación del algoritmo en paralelo, se utilizó para leer la fila una lista con doble-link, para poder agregar y eliminar elementos a conveniencia.

El paradigma de paralelización OpenMP, permite que cada hilo (núcleo) trabaje de manera independiente del resto, así que no se puede asumir que cada hilo está ejecutando la misma instrucción.

Dado que la estructura de la matriz rala es por filas, lo que se hace en el algoritmo es que cada hilo trabaje con una fila i, y realiza las operaciones respectivas para eliminar un elemento a_{ij} siempre que la fila j ya haya sido terminada de factorizar.

La implementación es la siguiente:

Algoritmo ILU

```
Comienza trabajo en paralelo
    vacio ← True
    Hasta terminar filas
        #pragma omp critical
            Si (vacio)
                iFila ← siguiente++
```

```
7       Si (iFila > n)
8           Terminar
9       Si (vacio)
10          Leer siguiente fila
11          vacio ← False
12          Calcular τ
13      Si no ha llegado a la diagonal
14           Hacer cero el siguiente elemento
15      Si ya llegó a la diagonal y guardó la fila anterior
16          Eliminar los elementos "grandes" de la matriz superior
17          Guardar la matriz inferior y la matriz superior
18          vacio ← True
```

El algoritmo permite que cada línea trabaje de manera independiente, y lo único que prohibe en la Línea 4 es la posibilidad de que a dos hilos distintos se les asigne el mismo índice de una fila.

4.2. Modifiación al $ILU(\tau)$

Buena parte de la eficiencia del algoritmo, se refiere al proceso de la Línea 14. En dicho proceso, se quiere hacer cero el elemento a_{ij} de la matriz A; para ello se realiza la operación

$$F_i - \frac{a_{ij}}{a_{jj}} F_j$$

Como la matriz es rala, para un elemento $a_{jk} \neq 0$ en la fila $j-$ésima, $k > j$ dado que corresponde a la matriz superior, es bastante probable que el elemento $a_{ik} = 0$, ello obliga a incluir el elemento a_{ik} en la fila (si $a_{ik} \neq 0$, simplemente se actualiza el valor), y después tener que realizar una operación para eliminar dicho elemento con la fila k-ésima, lo cual incluirá nuevos elementos y hará que el proceso se haga muy lento.

El valor del elemento a agregar es conocido:

$$a_{ik} = -\frac{a_{ij}}{a_{jj}} a_{jk}$$

Sea a_{k*} el máximo de los valores en la fila k-ésima. Eso quiere decir, que al hacer cero el elemento a_{ik}, el máximo cambio en valor absoluto que se realizará en un valor de la fila i-ésima es:

$$\left| \frac{a_{ik}}{a_{kk}} a_{k*} \right|$$

y si dicho cambio es menor a τ, entonces se asumirá que el elemento a_{ik} no es imprescindible para el cálculo de la factorización, y no se incluirá; en otras palabras, para incluir un elemento nuevo a_{ik}, este debe cumplir que:

$$\left|\frac{a_{ik}}{a_{kk}}a_{k*}\right| \geq \tau \Rightarrow |a_{ik}| \geq \left|\frac{a_{kk}}{a_{k*}}\right|\tau$$

5. Resultados

Para probar y depurar el precondicionador, se tomaron varios sistemas de la Colección de Matrices Ralas de la Universidad de Florida[1]. Cada sistema seleccionado $A\mathbf{x} = \mathbf{b}$ proveyó no solo la matriz A sino también el vector b. En todos los casos, las matrices **no** son definidas positivas y todos los elementos de la diagonal son distintos de cero. Las principales caraterísticas de dichas matrices se resumen en el Cuadro 1.

Nombre	Simétrica	Tamaño	NNZ	Tiempo de carga (s)
add20	Sí	$2\,395 \times 2\,395$	17 319	0.016
add32	Sí	$4\,960 \times 4\,960$	23 884	0.033
memplus	No	$17\,758 \times 17\,758$	126 150	0.111
bcircuit	No	$68\,902 \times 68\,902$	375 558	0.255
atmosmodl	No	$1\,489\,752 \times 1\,489\,752$	10 319 760	3.299
gsm106857	Sí	$589\,446 \times 589\,446$	11 174 185	5.541
dielFilterV2	Sí	$1\,157\,456 \times 1\,157\,456$	24 848 204	11.66
dielFilterV3	Sí	$1\,102\,824 \times 1\,102\,824$	45 204 422	22.169

Cuadro 1: Principales características de los sistemas utilizados.

Utilizando el algoritmo BiCGStab, se procedió a resolver cada uno de los sistemas, tomando como aproximación inicial el vector $\mathbf{x}_0 = \mathbf{0}$. El error se calculó sacando la norma euclídea al vector residuo $\mathbf{r} = A\mathbf{x}^* - \mathbf{b}$, donde $\mathbf{x}^*$ es la aproximación obtenida, y en todos los casos se puso como condición de parada que el error fuera menor a 1×10^{-6}, con 300 iteraciones como máximo. El algoritmo realiza los productos matriz-vector y el producto punto entre vectores en paralelo. Los resultados se muestran en el Cuadro 2.

Luego se procedió a realizar la factorización incompleta LU con el algoritmo presentado. Como se puede observar en el Cuadro 3, el tiempo de factorización no excedió el tiempo de carga de la matriz, y el tamaño resultante de la factorización no es mucho mayor al tamaño original de la matriz (extrañamente, en las matrices grandes, 1 elemento más). En todos los casos, la función para calcular el valor de τ que se pasó como argumento al algoritmo devolvía como valor la décima parte de la norma euclídea de la fila respectiva.

[1] *http://www.cise.ufl.edu/research/sparse/matrices/*

Nombre	Número de iteraciones	Error	Tiempo (s)
add20	1	5.3×10^{-11}	0.0002
add32	1	1.1×10^{-14}	0.0003
memplus	1	1.6×10^{-11}	0.0006
bcircuit	3	5.8×10^{-4}	0.0029
atmosmodl	10	2.7×10^{-4}	0.5225
gsm106857	300	1.2×10^{-3}	16.25
dielFilterV2	300	0.22	20.37
dielFilterV3	300	6.92	31.7055

Cuadro 2: Desempeño del algoritmo (BiCGStab).

Nombre	NNZ	NNZ Matriz Inferior	NNZ Matriz Superior	Total	Tiempo (s)
add20	17 319	7 465	2 431	9 896	0.019
add32	23 884	9 436	4 989	14 425	0.017
memplus	126 150	54 197	17 767	71 964	0.043
bcircuit	375 558	153 329	68 907	222 236	0.116
atmosmodl	10 319 760	4 415 004	5 904 757	10 319 761	2.280
gsm106857	11 174 185	10 584 739	589 447	11 174 186	1.953
dielFilterV2	24 848 204	23 690 748	1 157 457	24 848 205	5.29
dielFilterV3	45 204 422	44 101 598	1 102 825	45 204 423	9.48

Cuadro 3: Factorización incompleta LU (ILU).

Como la idea de la factorización es encontrar las matrices L y U tales que $LU \approx A$, entonces sin utilizar el precondicionador en el algoritmo BiCGStab, es posible utilizar dicha aproximación para pasar un valor inicial $\mathbf{x}_0$, resolviendo el sistema $LU\mathbf{x}_0 = \mathbf{b}$. Utilizando tan solo dicha aproximación, se procedió a resolver los sistemas utilizando BiCGStab nuevamente. Los resultados se muestran en el Cuadro 4.

Es interesante observar cómo en la matrices gsm106857 y dielFilterV3, se requiere tan solo una iteración del algoritmo. Eso deja preveer que la factorización debe aproximar bastante bien el valor de A, dado que el $\mathbf{x}_0$ es muy cercano a la solución real. Un resultado anómalo es la matriz atmosmold, la cual requirió más iteraciones que utilizando el mismo algoritmo con $\mathbf{x}_0 = \mathbf{0}$.

Y finalmente se procedió a resolver cada uno de los sistemas utilizando la versión BiCGStab precondicionado cuyos resultados se muestran en el Cuadro 5. No es de sorprender que las matrices gsm106857 y dielFilterV3 necesiten tan solo una iteración. Como se comentó en el caso anterior, la aproximación $\mathbf{x}_0$ inicial promete ser bastante buena. En todo caso, el error es menor en todos los casos, lo que hace suponer que el precondicionador sí ayuda en el proceso.

Nombre	Número de iteraciones	Error	Tiempo (s)
add20	1	5.2×10^{-11}	0.0003
add32	1	9.7×10^{-15}	0.0003
memplus	1	1.1×10^{-11}	0.0008
bcircuit	1	5.5×10^{-4}	0.0025
atmosmodl	37	8.7×10^{-4}	1.8818
gsm106857	1	2.1×10^{-15}	0.0985
dielFilterV2	1	9.9×10^{-13}	0.1294
dielFilterV3	1	6.4×10^{-14}	0.2003

Cuadro 4: BiCGStab con aproximación $\mathbf{x}_0$

Esta vez, el caso anómalo de la matriz atmosmodl nos puede brindar más luz acerca de la ayuda que presta el precondicionador en BiCGStab. Observe que aquí requirió tan solo 11 iteraciones, mientras que en el caso anterior 37 iteraciones; en ambos casos la aproximación inicial $\mathbf{x}_0$ es la misma. Comparando con el caso en que se utilizó $\mathbf{x}_0 = \mathbf{0}$, simplemente se concluye que esa es una mejor aproximación, pero que el precondicionador sí ayuda a que el algoritmo converja más rápido.

Nombre	Número de iteraciones	Error	Tiempo (s)
add20	1	4.7×10^{-11}	0.0004
add32	1	3.7×10^{-15}	0.0005
memplus	1	1.1×10^{-11}	0.0014
bcircuit	1	2.7×10^{-5}	0.0055
atmosmodl	11	5.0×10^{-4}	1.1615
gsm106857	1	2.0×10^{-15}	0.1697
dielFilterV2	1	8.8×10^{-13}	0.2169
dielFilterV3	1	5.8×10^{-14}	0.3507

Cuadro 5: BiCGStab con precondicionador.

A modo de comparación, se procedió a realizar la factorización *LU* completa de un par de matrices pequeñas, cuyo tamaño se muestra en el Cuadro 6. No se realizó en el resto de matrices, porque el tiempo de factorización parecía ser demasiado. Se puede observar como en este caso la factorización completa requiere 200 veces más espacio que la matriz original, mientras que la factorización incompleta implementada requería menos espacio que la matriz original.

Nombre	Tamaño	NNZ	NNZ L	NNZ U	Tiempo (s)
add20	$2\,395 \times 2\,395$	17 319	2 003 654	2 006 063	2.26
add32	$4\,960 \times 4\,960$	23 884	2 645 953	2 650 779	9.25

Cuadro 6: Comparación entre NNZ(L), NNZ(U) y NNZ(A).

6. Discusión y conclusiones

El algoritmo del precondicionador diseñado e implementado *en paralelo*, basado en la factorización LU incompleta, dio buenos resultados con sistemas de ecuaciones lineales grandes, pues el número de iteraciones requeridas por el algoritmo BiCGStab para converger, fue de 1 iteración en casi todos los casos, tanto en el algoritmo BiCGStab precondicionado y sin precondicionar, gracias a la aproximación inicial obtenida a partir del precondicionador.

Por otro lado, este mismo algoritmo BiCGStab con una aproximación inicial de ceros, para los sistemas lineales más grandes, no se logró la convergencia con 300 iteraciones, que es el número máximo de iteraciones que se utilizó en la implementación de dicho algoritmo.

Con respecto a los tiempos de ejecución, en todos los casos el tiempo de ejecución del algoritmo es menor al tiempo requerido por el sistema para cargar en memoria la matriz. Esto evidencia que el algoritmo es susceptible de ser aplicado en la práctica a matrices realmente grandes.

El espacio en memoria que requiere la factorización incompleta LU es proporcional al espacio que requiere la matriz de coeficientes del sistema. Esto también hace que el algoritmo sea susceptible de ser aplicado en la práctica.

Dado que la aproximación inicial obtenido para el vector de incógnitas $\mathbf{x}$ a partir de la factorización LU incompleta está muy cercana a la solución del sistema, esto permite concluir que la factorización LU calculada, a pesar de el poco número de elementos utilizados, es una buena aproximación de la matriz de coeficientes A.

Referencias

[1] Burden R., Faires, D.; Análisis Numérico; International Thomson; México, 1998.

[2] Chandra, Rohit et al. Parallel Programming in OpenMP. 1 edition. Morgan Kaufmann, 2000.

[3] Chapman, Barbara et al. Using OpenMP: Portable Shared Memory Parallel Programming. 1 edition. The MIT Press, 2007.

[4] Chen, K. E.; Matrix Preconditioning Tecniques and Applications; Cambridge University Press; USA, 2005.

[5] Fedorenco, R.; A Relaxation Method for Solving Elliptic Diffrential Equations. URSS Computational Math. and Math. Phys. 1:1092-1096, 1962.

[6] Figueroa, G., Carrera, E. Métodos iterativos para sistemas lineales grandes. Informe final proyecto código: # 5402-1701-0101. Vicerrectoría de Investigación, Instituto Tecnológico de Costa Rica, 2012.

[7] Golub, Gene H.; Van Loan, Charles F. Matrix Computations. 3rd edition. The Johns Hopkins Unversity Press, 1996.

[8] Greenbaum A.; Iterative Methods for Solving Linear System; SIAM; USA, 1997.

[9] Hackbusch W.; Iterative Solution of Large Sparse System of Equations; Springer Verlag; New York, 1994.

[10] Hogben, Leslie (ed). Handbook of linear algebra. Chapman & Hall, USA, 2007.

[11] Saad, Yousef. Iterative Methods for Sparse Linear Systems. 2 edition. Society for Industrial and Applied Mathematics, 2003.

[12] Quinn, Michael. Parallel Programming in C with MPI and OpenMP. 1 edition. McGraw-Hill, 2003.

[13] Trefethen, Lloyd N.; Bau III, David. Numerical Lineal Algebra. 1 edition. SIAM: Society for Industrial and Applied Mathematics, 1997.