Estudio de Rendimiento para la Solución de Ecuaciones Lineales Usando Computación en Paralelo

AUTOR: JOSE LUIS BOLAÑO HERAZO

TESIS DE GRADO PARA OPTAR POR EL TÍTULO DE MAGISTER EN INGENIERÍA CON ÉNFASIS EN SISTEMAS

DIRECTOR: MSc. JAIRO SERRANO

INGENIERO DE SISTEMAS

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍA

MAESTRÍA EN INGENIERÍA CON ÉNFASIS EN SISTEMAS

CARTAGENA, COLOMBIA

2015

Agradecimientos

Por dar a conocer la temática planteada y el apoyo al proyecto de investigación doy en primer lugar agradecimientos al tutor Jairo Serrano. Al Profesor Jairo Useche por ilustrar la importancia en sus de desarrollar una investigación con esta temática. En el transcurso de trabajo se aplicó de alguna manera conocimientos obtenidos a lo largo de los cursos y relaciones con el personal de la institución. Por consiguiente extiendo a los agradecimientos a la planta docente de la maestría que incluiría a los profesora Luz Stella, al profesor Luis Villa, David Franco, Martín Monroy, William Caicedo, Isaac Zúñiga, Julio Seferino Hurtado, Edgardo Arrieta, Juan Carlo Martínez, entre otro.

Reconocimiento

A la Universidad Tecnológica de Bolívar por permitir que hiciera parte de su comunidad y permitirme brindar a la comunidad académica un aporte.

Tabla de Contenidos

Lista de tablas

Lista de Figuras

Lista de Abreviaturas

A continuación se listan las abreviaciones utilizadas en el documento.

Abreviatura	**Significado**
BLAS	Basic Linear Algebra Subprograms
CentOS	Community Enterprise Operating System
CSC	Compressed Sparse Column
G++	GNU C++ Compiler
GCC	GNU Compiler Collection
HPC	High-performance computing
IBM SP	IBM Scalable POWERparallel
LAPACK	Linear Algebra Package
Lis	Library of Iterative Solvers
MPI	Message Passing Interface
MUMPS	MUltifrontal Massively Parallel sparse direct Solver
Ptr.	Pointer
RUA	Real Unsymmetric Assembled
SuperLu	Supernodal LU
SWIG	Simplified Wrapper and Interface Generator

Capítulo 1

Descripción del Proyecto

Introducción

La motivación del trabajo de investigación radica en que la computación de alto desempeño son un gran aporte para la ciencia y la ingeniería para resolver problemas de un costo computacional alto. En la Universidad se resuelven matrices de ecuaciones lineales de gran tamaño que son generadas a partir del método de elementos finitos. La eficiencia para obtener resultados más precisos con el método mencionado radica en generar una matriz con una gran cantidad de incógnitas. Pero más incógnitas significan más cálculo por realizar. Con un computador de alto desempeño como el que cuenta la Universidad se espera que el tiempo de solución de las ecuaciones lineales con las que se trabaja se reduzca considerablemente con procesos en paralelo usando librerías de código abierto.

A nivel mundial se ha usado la computación de alto rendimiento para resolver problemas como curas a enfermedades, predecir terremotos, mejoramiento de motores, etc. («http://www.eafit.edu.co/», s. f.). Los computadores usados son los siguientes a nivel mundial:

1. Buscar curas a enfermedades: Watson en Inglaterra una gran base de datos de cáncer, con el fin de encontrar tratamientos más efectivos y personalizados

2. Entender cómo funciona el mundo: Fujitsu K en Japón se utiliza para entre otras cosas predecir terremotos
3. Ingeniería mecánica: Titan en Estados Unidos busca mejorar la eficiencia de los inyectores de motor.
4. Estudios de genética: Blue GenP de la Universidad de Rutger en estados Unidos.

En Latinoamérica se cuenta con muy pocos sistemas de alto desempeño en comparación con Estados Unidos que cuenta con más de 200 según www.top500.org. México tiene el laboratorio de la Universidad Autónoma de México con un computador de Alto desempeño bautizado como Miztli. Cuenta con 5,312 núcleos de procesamiento Intel E5-2670, 16 tarjetas NVIDIA m2090, una memoria RAM total de 15,000 GBytes y un sistema de almacenamiento masivo de 750 Terabytes.(«Miztli», s. f.).

Gráfica 1. Super Computador Miztli. Tomado de "http://www.super.unam.mx/index.php/content-layouts?showall=1"

La Universidad Autónoma Metropolitana, también en México, cuenta con un clúster denominado Yoltla, el cual cuenta con 216 procesadores de 10 núcleos, y con un total de 6912 GBytes de memoria RAM distribuida(«Laboratorio de Super Cómputo y Visualización en Paralelo», s. f.).

En Colombia se cuenta con el CECAD, siendo un laboratorio creado por la Universidad Universidad Distrital “Francisco José de Caldas”, su objetivo es prestar servicios de almacenamientos y cálculo a los grupos de investigación de la Universidad y a las Universidades suscritas a GridColombia. Entro los proyectos que han realizado se encuentra “Análisis de genomas y transcriptomas completos.”, “Paralelismo en Nube de Puntos.”, “Optimización de los métodos de factorización en los algoritmos asimétricos.”, entro otros(«CECAD», s. f.).

El supercomputador de Medellín es Apolo, cuenta con 96 servidores. Es utilizado por más de 22 grupos de investigación. Entre ellos se encuentra los grupos de Economía Espacial, Electromagnetismo Aplicado,Mecánica Aplicada y Gestión de Producción y Logística de Eafit(«Laboratorio Super Computo Medellín», s. f., «Super Computador Medellín», s. f.).

Teniendo en cuenta el estado actual de la infraestructura en el Latinoamérica, se hace énfasis en que esta necesidad de hacer sistemas de cómputo más eficientes se debe a que han evolucionado a través de los años para resolver problemas de mayor complejidad.

En un inicio su principal objetivo se basaba en llevar las cuentas de los bancos con una exactitud e insuperable por un empleador. Con el transcurrir del tiempo las micro tiendas agilizaron sus procesos al delegar al cálculo computarizado el trabajo de calcular el costo que un cliente generaba en una compra y las vueltas que era necesario retornar, y es a partir de allí que ningún cajero de un supermercado desperdicia su tiempo en una labor tan repetitiva. Como se ve en las labores menos exigentes el hombre busca la forma de optimizar sus resultados, en el ejemplo del cajero puede atender más rápido a los clientes. En trabajos más complejos el poder de cómputo ha ayudado a mejorar el rendimiento de los automóviles al analizar las variables que influyen en el desempeño del motor, analizar la mejor estructura de los barcos para navegar grandes distancias, y de qué forma lanzar un satélite al espacio. El poder de cómputo ha sido un factor esencial para obtener resultados óptimos y en menor tiempo. Sin embargo, como es común en la curiosidad humana, se ha llegado a pedir procesos que en apariencia parece que el poder

de cómputo llega a su límite. La solución de matrices es una de ellas, y el caso de estudio de este trabajo. Para una matriz de con poca incógnitas, cuantificada en unas cuantas decenas, es un dilema para resolverlo sin asistencia computacional. La computadora puede resolverlos sin problemas debido a la cantidad de procesos por segundos que es capaz de ejecutar sin quemar un circuito. Sin embargo en trabajos profesionales como es el caso de análisis de estructuras las incógnitas se elevan a millones lo cual dificulta encontrar la soluciones incluso para un computador de alta tecnología.

Unos de los inconvenientes que se presenta para diseñar computadores más potentes en este punto radican en que al dar más velocidad de procesos sobrecalienta los circuitos más allá de sus límites, haciendo inútil este diseño. Entonces la solución más conveniente sería no aumentar la velocidad de procesamiento y dividir las tareas. Una actividad se termina más rápido entre más procesadores estén involucrados pero también varía de acuerdo a la complejidad de la operaciones. En el caso puntual de la matrices se estimaría que para algunos factores dividir la tarea aplicando métodos particulares darían buenos resultados en unos tipos particulares de ecuaciones y en otros no. La idea principal radica en experimentar y reconocer que casos son los adecuados y que herramientas o algoritmos son adecuados a diferentes situaciones. Se planea trabajar con ecuaciones lineales de estructura NxN con sus datos dispersos en la diagonal principal. Estos estudios que yacen en la solución de este tipo de matrices es fundamental para trabajos de resistencia de materiales que son trabajados por la facultad de ingeniería. Unos resultados obtenidos a tiempo y precisos dan un mejor entendimiento en el análisis

de estructuras mecánicas en todo tipo de construcciones tales como aeronáuticas, de edificaciones, etc. Todo este caso de estudio se da a raíz de las investigaciones que se realizan en la Universidad.

En la facultad de Ingeniería de la Universidad se trabaja específicamente en analizar estructuras para áreas como por ejemplo el diseño de cascos de buques. Con las estructuras se analiza la tensión, la velocidad, o el desplazamiento de un fluido, entre otras unidades físicas en toda la superficie. Para tal objetivo se utiliza el método de elementos finitos para predecir los valores de las magnitudes. Esta técnica simplifica el problema en una ecuación lineal que entre más compleja el resultado es más preciso pero necesita más tiempo de cómputo.

Como se mencionó antes estos estudios consisten en predecir el valor de una unidad física, como puede ser la tensión para conocer la deformación de una estructura de acuerdo a una fuerza. Sin embargo para predecir la magnitud se cuentan con fórmulas que no son aplicables a formas complejas como el casco de un buque o un puente. El método de elementos finitos es la técnica que superar este problema y es la que se utiliza en la facultad. En el caso del casco del buque si se quiere conocer el grado de tensión que se puede generaren todos sus puntos se debe aplicar las ecuaciones correspondientes. La ecuación es conocida pero no puede ser aplicada para formas complicadas como el casco o un puente.

Sin embargo puede ser aplicada para formas conocidas como un triángulo o un rectángulo. El método de elementos finitos toma ventaja de este hecho. Se reemplaza la complicada forma de la estructura con una aproximación de elementos simples, la

totalidad de los elementos es conocida como la malla de elementos finitos. Ese mapa es único para cada problema; la forma puede ser de triángulos, cuadriláteros, o la combinación de distintos elementos. Los elementos se les definen las propiedades físicas del material, dependiendo del problema puede ser la elasticidad, la viscosidad, torque, entre otros. Una vez hecho esto el problema se reduce a una ecuación lineal las incógnitas representan la magnitud que se desea conocer para el estudio. Finalmente la solución de la ecuación le sirve a un software de computadora para que visualice las magnitudes físicas en una estructura en todos los puntos de la misma; por ejemplo mostrar con las zonas donde hay más tensión, mayor desplazamiento, o temperatura entre otros(Johnson & Mathematics, 2009; Logan, 2011).

El problema que radica en aplicar este método, y el cual se presenta en las investigaciones de la Universidad, se basa en que la solución obtenida es aproximada. Para obtener una solución más precisa se debe aumentar el número de elementos finitos en la malla; sin embargo a mayor número de elementos se eleva el tiempo de cómputo. La exactitud del cálculo va depender del número de elementos que se tenga en la malla. Más elementos implican que estos sean más pequeños y el resultado más preciso. Desafortunadamente implica más cálculos por hacer.

Con los modernos computadores es posible resolver estos sistemas sin embargo se hace necesario experimentar con las librerías de libre distribución y la viabilidad de usarlos con los equipos de alto desempeño de la Universidad para conseguir los resultados más exactos en un tiempo razonable.

El sistema HPC cuenta con 32 procesadores que en teoría deberían ayudar a resolver ecuaciones con un alto número de incógnitas si trabajan en paralelo. Si las herramientas de libre distribución ofrecen beneficio al utilizar esta estructura de cómputo, se reflejaría en que las investigaciones que utilizan el método de elementos finitos obtengan resultados cada vez más preciso en un tiempo razonable utilizando las herramientas y consideraciones que en este trabajo se contemple.

Objetivos

Realizar un estudio del aprovechamiento de la infraestructura HPC de la UTB en la solución ecuaciones lineales simétricas reales (Ax=b) con una demanda alta de incógnitas. Usando procesamiento distribuido bajo el uso de software libre. Específicamente se va a tratar de:

- Verificar el rendimiento en la solución de matrices de gran tamaño con software open Source que permita dividir el trabajo en varios procesadores.
- Analizar la viabilidad técnica de la infraestructura de la universidad para resolver sistemas lineales con más de 10 millones de incógnitas.
- Identificar consideraciones, con los software open Source que se utilicen, a tener en cuenta para hacer un uso adecuado de los recursos de la infraestructura HPC de la universidad.

Hipótesis

Al aumentar el número de procesadores involucrados a resolver un sistema de ecuación lineal NxN de un alto número de incógnitas se espera que el tiempo de ejecución sea reducido considerablemente. Se espera que la respuesta del sistema sea más oportuna a medida que el número de procesos aumente en matrices con más de un millón de incógnitas.

Pregunta de investigación

¿La infraestructura HPC de la Universidad puede aprovechar la disponibilidad de sus 32 procesadores para escalar los problemas de ecuaciones lineales NxN con un alto número de incógnitas?

Capítulo 2

Contexto y Motivación

El uso de herramientas computacionales para la solución de sistemas de ecuaciones lineales tiene como objetivo obtener las soluciones en el menor tiempo posible y con una certeza de confiabilidad muy alta.El mayor problema que presenta el uso de solvers es que con a mayor cantidad de datos el tiempo computacional se eleva exponencialmente.

En la última década a habido un interés en presentar diferentes soluciones para mejorar el rendimiento de obtener las soluciones a un sistema linear de ecuaciones complejo(Couturier & Jezequel, 2010; Falgout, Jones, & Yang, 2006; Guo, Gorman, Lange, Mitchell, & Weiland, 2013; Kuźnik, Paszyński, & Calo, 2013; Michailidis & Margaritis, 2011).Todos se centran principalmente en diferentes técnicas que minimicen el esfuerzo de cómputo para obtener los resultados.

Los métodos para resolver sistemas de ecuaciones lineales generalmente se califican en dos categorias principales. La primer a son los conocidos como solución directa en la cual la solución exacta se obtiene a partir de un número finito de procedimientos aritméticos, en el cual no hay lugar para un porcentaje de error. El método directo más conocido es el algoritmo de eliminación directa gaussiana(Caldwell & Zisserman, 1984; Di Felice & Clementini, 1987; Michailidis & Margaritis, 2011).

Otro método da un a solución a cada vez más aproximada en la medida que reliza el mismo procedimiento computacional en cada una de las iteraciones de la ejecución

principal, el más usado es el método de Krylov(Dongarra & Eijkhout, 2000, pp. 507–510; Knibbe, Oosterlee, & Vuik, 2011; Michailidis & Margaritis, 2011).

La solución eficiente de estos sistemas es fundamental para la resolución de diversos problemas relacionadados como la estadística, la econometría, la teoría de juegos. Muchos de esos problemas se pueden modelar en un denso sistema de ecuaciones lineales. Es por ello que la solución de grandes sistemas de lineales es de suma importancia en el campo de la computación científica(Michailidis & Margaritis, 2011).

Uno de los intentos más remotos es (Caldwell & Zisserman, 1984), en el que diseñan un programa llamado F. MATINV donde se incorporan una sección extra en la sección en que se encarga de eliminar las filas , la rutina es conocida como MATINV. Los resultados muestran que el consumo de CPU y el uso de memoria cae lo suficiente para considerar la subrutina una mejora substancial para la solución de sistemas de ecuaciones lineas de tipo A x = b. El problema de este artículos es solo comparan sus resultados con la anterior versión y no hacen comparaciones con otros mecanismos que al menos se basen también en eliminación gaussiana.

Una implementación en pascal del metodo de eliminación gaussiana, los autores defiende que pascal debe usarse más en análisis científico, con su implementación muestran las estrategias usadas para resolver este tipo de problemas numéricos. Su estudio se centra en matrices asimétricas, sus resultados muestras el tiempo que toma su implementación en solucionar distintos tamaños de matrices. Basicamente sus resultados solo le hacen ver que la limitación para resolver matricez más complejas está dada por la potencia de la CPU y la memoria dinámica que debe ser asignada al sistemas. Los

autores (Di Felice & Clementini, 1987) concluyen que van a seguir investigando en este aspecto. Igual que el anterior no se hacen comparaciones con implementaciones parecidas y a pesar que mencionan que la solución de sistemas de este tipo es de gran aplicación en la ingeniería no hacen o no mencionan que las pruebas realizadas trantan de resolver algún problema concreto.

Los autores (Stafylopatis & Drigas, 1988) hacen una explicación de un algoritmo basado en modelos estadísticos. Dicho algoritmo hace uso del método de Gauss Jordan con pivotes parciales, el cual ellos describen como un sistema de una estructura triangular. Realizan pruebas las cuales comparan con su versión secuencial y concluyen que llega a una mejora pero la implementación de la matriz para ser operada en paralelo y las políticas de accesoa la memoria son factores que pueden complicar la utilización del métodopara un problema concreto.

Otros artículos como el publicado por (Kincaid & Young, 1988)solo mencionan las características de paquetes de software desarrollado por la Universidad de Texas en Austin. El software que evaluan es ITPACK, el cual tiene varias versiones hasta la época de publicación del artículo. Hacen un recorrido de sus distintas versiones mostrando las mejoras hasta la versión 4, que llama NSPCG, en la que logran consolidar herramientas para solución de matrices simétricas y asimétricas. Este escrito no muestra resultados prácticos, tampoco es su propósito, su interes radica en mostrar la evolución de sus estudios en el desarrollo de la herramienta computacional.

A diferencia del artículo anterior el escrito de (Hayder, Keyes, & Mehrotra, 1998) se pregunta las ventajas de las librería PETSc contra una implementación de un algoritmo

para resolver sistemas de ecuaciones difereciales desarrollado en HPF (High performance Fortran). Las pruebas las realiza en un problema clásico conocido como el problema de Bratu. Sus análisis muestran que el tiempo que necesita cada implementación es considerablemente el mismo, sin embargo la implementación en HPF es más dispendiosa puesto que PETSc maneja el uso de paso de mensaje sin intervención del usuario. Los autores destacan que este tipo de herramientas es de gran importancia para los investigadores que necesitan obtener resultados sin preocuparse por desarrollar sus propias implementaciones. Hubiera sido buena idea hacer comparaciones con otros softwares o paquetes que manejaran al igual que PETSc procedimientos en paralelo y comparar los resultados.

Sin embargo no todas las librerias para solución de sistemas lineales son del todo desatentidas por el usuario.Según las observaciones de (Dongarra & Eijkhout, 2000) dependiendo del tipo de estructura del problema es preferible decirle al programa decirle cuales son las porciones a tomar para obtener los resultados con menor consumo de pasos. Esto se basa si se toma un sistema de ecuaciones denso o disperso. Se afirma que esta última es mas compleja y es necesario en muchos casos especificar que porciones de la matrix debe dedicarse cada procesador. También sustentan que las soluciones directas tienen una complejidad muy alta lo cual hace inviable solucinar problemas con gran cantidad de datos.

Los autores muestran los método mas significativos para soluciones directas(Dongarra & Eijkhout, 2000, p. 506), y muestran que los métodos de solución interactiva son más enficientes para grandes volúmenes de datos(Dongarra & Eijkhout,

2000, p. 507). Hacen una revisión de porqué la estructura de la matrix es importante, de hecho la complejidad computacional se eleva al ser no cuadrática. Este artículo trata muchos puntos importantes pero solo hacen un recuento de la metodología de solución de sistemas lineales y como algunas herramientas computacionales hacen el trabajo, pero hacen pruebas con diferentes problemas para verfiricar como son los rendimientos con diferentes aplicaciones ni como es el uso de memoria y de procesamiento.

La solución de sistemas de encuaciones lineales es usado en simulaciones. Los investigadores (Ercan, Fung, Ho, & Cheung, 2003) usan dichos sistemas para modelar la red de poder de las vías de tren. Hacen la observación que la rápida evolución de los diferentes tipos de procesadores permiten que se pueda realizar operaciones en paralelo para explotar los datos. Su simulación trata de observar cual es la media ideal de electricidad adecuada para los rieles, teniendo en cuenta variables como la cantidad de trenes, las uniones de los rieles, entre otros factores. Esto es importante porque la electrificación es crucial para el diseño, las operaciones diarias y la planificación. Todo esto lo redujeron a un sistema de ecuaciones lineales y usando la paralelización que ofrecen los modernos computadores, concluyeron que la paralelización es positiva pues no ponía en riesgo la velocidad de la simulación, lo cual era lo esperado por los autores. Este es un ejemplo práctico de la importancia del aplicaciones de solución de ecuaciones lineales, sin embargo los autores pudieron usar paquetes de software a su disposición para ver que tan mejorable era la simulación, los autores solo resolvieron el problema manipulando la paralelización con la que proporcionaba los modelos de los procesadores.

Otros modelos para resolver sistemas se ecuaciones lineales se basan en usar una grilla de computadores, de esta manera no solo se depende de los cores de un computador. En el trabajo de (Couturier & Jezequel, 2010) usan una implementación de paso de mensajes escrita en Java conocida como MPJ. Su implementación la comparan con una C usando MPI, y el rendimiento de ambas la comparan con PETSc . Concluyen que sus resultados de rendimiento y velocidad son comparables con MPI y PETSc pero no llegan a ser mejor completamente.

Los investigadores pudieron hacer una prueba del pase de mensajes a instruciones en PETSc para ver si la estructura de ingreso de la información influía al momento de crear una grilla. Se pudieron dar recomendaciones si al aumentar el número de computadores en la grilla mejoraba simpre o llega un punto de caida.

En estudios más recientes el interes por solucionar estos sistemas no ha desaparecido. El aumento de los nucleos en los procesadores ha permitido que el interes cresca. Las observaciones de (Manguoglu, 2011) muestra que si se hace una buena implementación de la paralelización se puede llegar a mejores resultados que el conocido método de Krylov. Manguoglu introdujo un nuevo solver que en sus pruebas mostró que es en la mayoría de los casos más robusto que el método de Krylov y los directos. Sin embargo enfatiza que dependiendo del la estructura de la matriz el solver debe ser ajustado. Con base en las observaciones del autor se puede decir que las herramientas como PETSc tienen ventajas aún sobre el solver propuesto. Usar la implementación que proponen requiere muchas configuaciones y adaptaciones para ponerlo a trabajar. Manguoglu comparó su implementación con el paquete PETSc pero no pudo probar otros

paquetes o incluso implementaciones de paralelizarían usando multi-hilos que ofrecen las librerías como OPENMP.

Una publicación hecha por Michailidis y Margaritis (2011), realizaron tres pruebas con OPENMP las dos primeras eran modificaciones triviales de la forma LU del método de eliminación Gaussiana y la tercera era una paralelización por segmentación que resultó tener un alto grado de paralelismo. Ellos utilizaron CPUS con 4 cores, en los cuales las pruebas se usaron matrices desde 32 × 32 a 4096 × 4096. Los resultados se centraron en la tercera implementación, la cual presentó una leve mejorar contra los otros dos que hacian sus operación por filas y por columnas respectivamente.

Su trabajo presenta muchos detalles pero solo se centraron en comparar las implementaciones desarrolladas y no probaron con algún kit para ver su rendimientos. El uso de cuatro nucleos deja muchas interrogantes relacionadas a si el rendimientos constantemente mejora con el aumentos de los hilos procesamiento, los autores consideran esto como trabajos a futuros.Debiso a este detalle los autorespueden considerar poner a prueba con herramientas que utilizan la paralelización con unidades de procesamiento gráfico, con ellos el número de nucleos es superior a los que una unidad de procesamiento central. Puesto que estos han sido un gran avance en los últimos años.

Un estudio del uso de GPU para resolver sistemas de ecuaciones lineales es presentado por (Chang, Stratton, Kim, & mei W. Hwu, 2012) usan la tecnología CUDA como base para diseñar un solver que utiliza triangulización con el fin de dividir las tareas enviar a a procesar a los diferentes hilos de procesamiento. Su implementación se

basa en el algoritmo de SPIKE. Su motivación radica en que muchos solvers están diseñador para cierto patrón de matrices, el que ellos desarrollaron es aplicable a un conjunto más amplio de problemas. En este artículos los autores comparan la implementación de SPIKE con una CPU, con GPU y con l unión de CPU y GPU. Una de las limitantes de su implementación recae en que tiene que usar una gran cantidad de caché. A pesar que la información de los autores es muy completa, ellos noenfatizan en la resolución de un problema concreto solo prueban datos establecidos de matrices. Además ellos aclaran que pueden haber software con mejores prestaciones que la actual. Lo cual lleva apensar que los algunos de los Kits mencionados en los demás escritos podría presentar un mejor rendimiento dependiendo del problema a solucionar.

Como se pudo observar a través de los diferentes escritos la solución computacional de sistemas lineales ha evolucionado, pasando desde la programación secuencial haciendo pequeñas mejoras a los algoritmos secuenciales establecidos. El uso de paralelismo se centraba en usar los núcleos de los procesadores, más adelante se centró en que diferentes porciones del trabajo se distribuyera entre varios computadores como es el caso de utilizar MPI.

La mayoría de los artículos hacían desarrollos propios de los algoritmos con pequeñas mejoras, otros comparaban su versión con paquetes como PETSc, ITPACK. O HYPRE. La gran ventaja de usar estas herramientas son los diferentes métodos de solución de ecuaciones lineales. Sin embargo en el artículo nunca pusieron a prueba los diferentes métodos que poseían.

Con base en lo anterior se propone tomar un rumbo diferente a lo estudiado. Se planea estudiar el comportamiento de algunas librerías Open Source sobre la infraestructura de HPC de la Universidad. Verificar como afecta la escalabilidad y los inconvenientes al resolver los problemas lineales. A los software mencionados anteriormente se puede sumar SuperLu, que resuelve los sistemas de ecuaciones lineales con una complejidad de $O(n^2)$ usando la matriz de la forma LU = b(Xiaoye S. Li, 2005; X. S. Li et al., 1999). Los resultados de esta investigación servirían para orientar que herramientas sería conveniente par a solucionar ciertas variantes de ecuaciones lineales dependiendo del trabajo que se esté realizando y que consideraciones tener para no subutilizar los procesadores. Adicionalmente, si el recurso está disponible se podría tener en cuenta analizar la implementación en CUDA que propuso (Chang et al., 2012), pues el código fuente se encuentra disponible para su uso.

Capítulo 3

Metodología.

En la presente investigación se trabajará con datos de sistemas lineales las cuales cuenten con una gran cantidad de incógnitas a resolver. Se utilizarán para estudiar el aprovechamiento de la infraestructura HPC de la UTB frente a sistemas lineales de variada complejidad. Se planea realizar el trabajo de la siguiente forma.

Se obtendrá matrices de repositorios que cumplan con las características necesarias (matriz dispersa NxN con elementos reales). Se buscará que tenga un tamaño para la solución de de hasta más de diez millones de incógnitas. A parte se tomará matrices provista por la facultad de ingeniería de la facultad.

Se descagará las librerias que son utilices para estos problemas. Se consultará en fuentes bibliográficas aquellas que son de libre distribución y que estén disponibles para su uso. Deben permitir solucionar las ecuaciones lineales a travésde múltiples procesadores y ser compatible con la distribuciones linux.

Se procederá a compilar las librerías de acuerdo a las intrucciones provistas. De ser necesario se descagaran también los compiladores y cualquier otra librería a la cual dependan para su funcionamiento. Dentro de la configuración se les activará la opción para procesamiento en paralelo en caso de ser opcional.

En casos de presentar errores de compilación, se recompilará una vez solucionados los inconvenientes. Generalmente los errores son por dependencias o por errores de las órdenes de compilación.

Se verificará que el software está bienn instalado ejecutando un ejemplo secillo. Una vez configurado sin problemas se procederá con otros candidatos a ser configurados.

Las librerías se configurarán para leer los archivos de matrices con los que se cuenta. El formato de matriz se adaptará al requerido por ñla librería de ser necesario.

Para cada matriz se medirá el tiempo de ejecución que toma la librería resolverlo. El tiempo de ejecución se basaría desde el instante que se ejecuta el programa hasta que obtiene las respuestas a la ecuación lineal. Contemplando desde la lectura de la matriz desde un archivo plano, la carga en la memoria, y la escritura de las solución. La herramienta “time” que viene instalada en las distribuciones de Linux es la que permitiría calcular el tiempo, de la salida que genera se toma aquel que dice “real” que hace referencia al tiempo total que tiene que esperar el usuario a que termine la aplicación su ejecución.

Los resultados obtenidos se tabularán en gráficos de dispersión donde se plasme el tiempo de ejecución que toma a cada librería resolver las ecuaciones lineales con diferente número de procesadores y tamaños de matriz.

Por último se analizarán los resultados obtenidos. Dando respuestas a los objetivos planteados y la hipótesis planteada en el trabajo.Se analizará la influencia del tamaño de la matriz con el uso procesamiento en paralelo a travez de múltiples procesadores.

Capitulo 4

Marco teórico

A continuación se presentan conceptos relacionados con la solución de ecuaciones lineales reales. Se empieza mencionando la matriz con la que se trabajaron, luego los algoritmos y propiedades para que un sistema lineal tenga solución y por último algunos conceptos computacionales.

Matriz dispersa: También llamada matriz esparza o hueca es aquella que tiene la mayoría de sus elementos ceros. Las matrices de este estilo reducen el costo de memoria computacional al hacerlo mediante el conjunto ordenado o desordenado de triples (i, j, a_{ij}) donde $a_{ij} \neq 0$(O'Connor, 1997; Tewarson, 1973, pp. 24, 203).

Gráfica 2 Matriz dispersa

Métodos Directos: Los métodos directos son aquellos que resuelven una ecuación lineal factor izando la matriz de coeficientes. Los métodos directos son la factorización LU, eliminación de Gauss, etc.

Factorización LU: es un método directo para la solución de sistemas lineales. Las condiciones iniciales de este método garantizan que se pueda factorizar la matriz de

coeficientes A = LU, donde L es una matriz triangular inferior y U es una matriz triangular superior(Davis, 2006). Si la factorización existe el problema original

$$Ax = b$$

Se escribe de la forma

$$LUx = b$$

El cual se reduce a la sucesión de sistemas triangulares

$$Ly = b \; y \; Ux = b$$

Ya que tanto L como U son triangulares cada uno de los sistemas se resuelve fácilmente por eliminación.

Eliminación de Gauss: La idea de este método directo es de reducir la solución de un sistema de ecuaciones lineales a una cuya matriz de coeficientes sea triangular. Una vez hecho esto la solución al sistema se obtiene a partir de sustitución hacia atrás o sustitución hacia adelante(Allaire & Kaber, 2008). El algoritmo es el siguiente.

1. Ir a la columna que no contenga ceros
2. En caso de tener un cero en la primera fila de la columna se intercambia con un a fila que no la tenga
3. Luego, obtener ceros debajo de este elemento delantero, sumando múltiplos adecuados de la fila superior a las filas debajo de él.
4. Se vuelve a realizar el paso 1 trabajando con la submatriz que queda al obviar la fila trabajada.

5. Con la última fila se realizar substitución hacia las filas superiores para obtener el valor de las incógnitas.

Solver frontal: Es un método directo para la solución de un sistema de ecuaciones lineales basado en la Eliminación de Gauss. Al contrario de la eliminación Gaussiana, evita una cantidad de operaciones que involucran términos cuyo valor es cero. Ajusta la matriz y elimina las ecuaciones tomando un subconjunto de elementos a la vez. Dicho subconjunto es el llamado frontal y es la parte intermedia entre la parte del sistema procesado y la que no ha sido trabajada. Solo las partes ensambladas entran al frontal. La matriz nunca es creada completamente en el proceso. El proceso de la porción frontal implica una operación sobre una matriz densa, esto es que la mayoría de los elementos no son ceros. Computacionalmente la parte frontal es llevada a la memoria principal(Amestoy, Duff, & L'Excellent, 2000)

Métodos Iterativos: Un método iterativo obtiene un conjunto de soluciones aproximadas a partir de un vector inicial de x; en cambio los métodos directos intentan resolver el sistema a partir de un conjunto de pasos finitos(Skiba, 2005).

Jacobi: Es un método iterativo basado en resolver para cada variable localmente con respecto a las otras variables; una iteración del método corresponde al resolver cada variable una vez.

Gauss-Seidel: Similar al Jacobi a excepción que usa los valores actualizados una vez son disponibles. Si Jacobi converge, el método Gauss-Seidel convergerá más rápido.

Gradiente Conjugado: Este método iterativo una secuencia de vectores ortogonales. Estos vectores son el residuo de las iteraciones. Consecuentemente usa las direcciones de búsquedas(ρ) para actualizar las por un múltiplo (α) iteraciones y los residuos. El tamaño de las secuencias puede crecer los suficiente, pero solo un pequeño número de vectores es mantenido en memoria. El pseudocódigo se muestra en la (Gráfica 3).

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
end

Gráfica 3 Algoritmo gradiente conjugado

Numero de condición: El número de condición es un valor que se multiplica al error para saber si la respuesta se aproxima a la solución. El cálculo del error en algunos casos puede ser pequeño más sin embargo la aproximación se aleja de la solución, en este caso el número de condición es alto para contra restar esa apreciación. El número de condición

no afecta considerablemente al error si está cerca a uno, por consiguiente se dice que la matriz está bien condicionada. Se considera mal condicionada si es mayor a 1(Skiba, 2005). Está representada por la fórmula

$$v(A) = \begin{cases} ||A||_{\infty}||A^{-1}||, & si\ no\ es\ singular \end{cases}$$

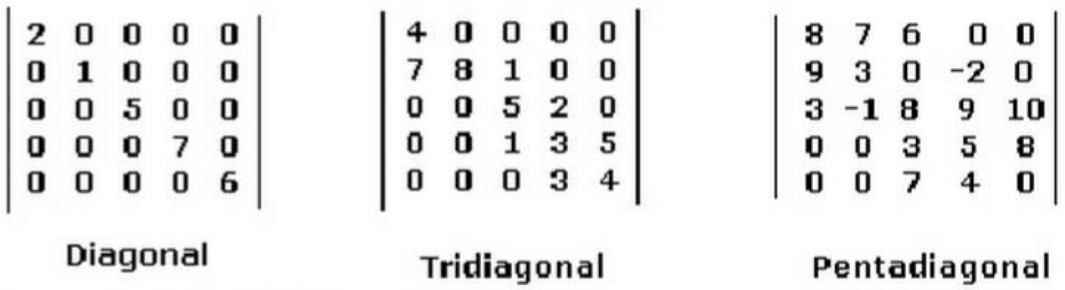
Ancho de banda: En una matriz dispersa el ancho de banda se le denomina al número de columnas distribuidas alrededor de la diagonal principal. Fuera de ella todos los elementos son nulos. Por el contrario dentro del ancho de banda puede tener elementos nulos(Carballo, Delgado, Canteli, & Rey, 2006).

Formalmente una matriz $nxn\ A = (a_{ij})$ es una matriz banda si todos los elementos son cero en una zona diagonal que está determinado por los rangos k_1 y k_2:

$$a_{i,j} = 0\ si\ j < i - k_1 \quad o \quad j > i + k_2; k_1, k_2 \geq 0$$

Los valores de k_1 y $k_2$son el ancho inferior y superior respectivamente. El ancho de banda está determinado por $k_1 + k_2 + 1$.

Una matriz de ancho de banda 1 se define como matriz diagonal. Ancho de banda tres tridiagonal. Para un ancho de banda P se le denomina P-banda (ver Gráfica 4).



Gráfica 4 Matriz banda

Valores propios: son aquellos valores λ que sastisfacen que para una matriz A sastisface la ecuación $Ax = \lambda x$(Kolman & Hill, 2006). Todo vector de x que satisfaga la ecuación anterior es un vector propio de A. De forma generar para una matriz nxn la sus valores propios están dados por

$$\boldsymbol{det(\lambda I - A) = 0}$$

Donde I es la matriz identidad.

Radio espectral: En una matriz es el mayor de los valores propios y comúnmente se denota por $\rho(.)$. Se dice que una matriz converge si su radio espectral es menor a uno(Quarteroni & Saleri, 2007).Si $\lambda_1 \ldots\ldots, \lambda_n$ son valores propios de una matriz $A \epsilon c^{nxn}$, entonces el radio espectral $\rho(A)$ es:

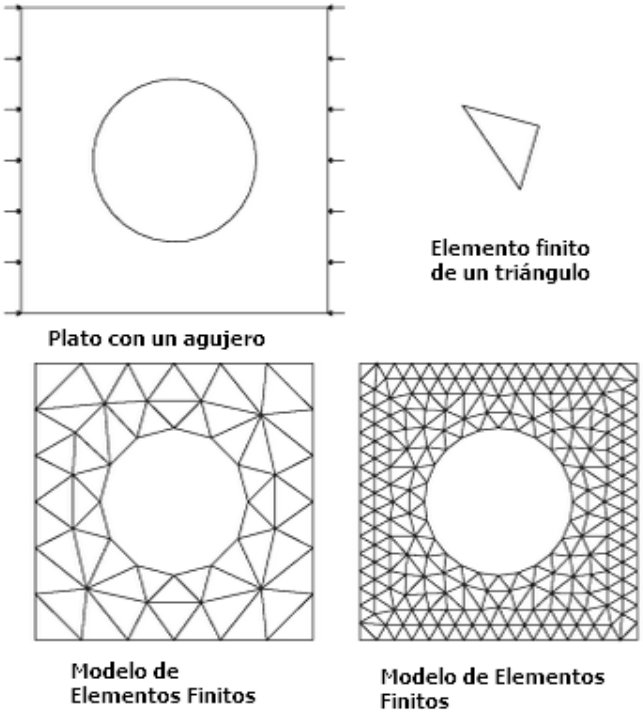
$$\rho(A) := \max_i(|\lambda_i|)$$

Computación Paralela: es una forma de cómputo en la que muchas instrucciones son ejecutadas simultáneamente, radica en el principio de que problemas grandes pueden ser resueltos dividiéndolos en problemas más pequeños(Bhujade, 2009; Pacheco, 1997).

MPI ("Message Passing Interface", Interfaz de Paso de Mensajes): es un estándar que define la sintaxis para ser usada en bibliotecas de paso de mensajes para que exploten la existencia de múltiples procesadores. El paso de mensajes es una técnica de programación concurrente usada para aportar la sincronización entre procesos de manera similar a como lo hacen los semáforos con el tráficos(Gropp, Lusk, & Skjellum, 2014; Karniadakis & II, 2003; Pacheco, 1997).

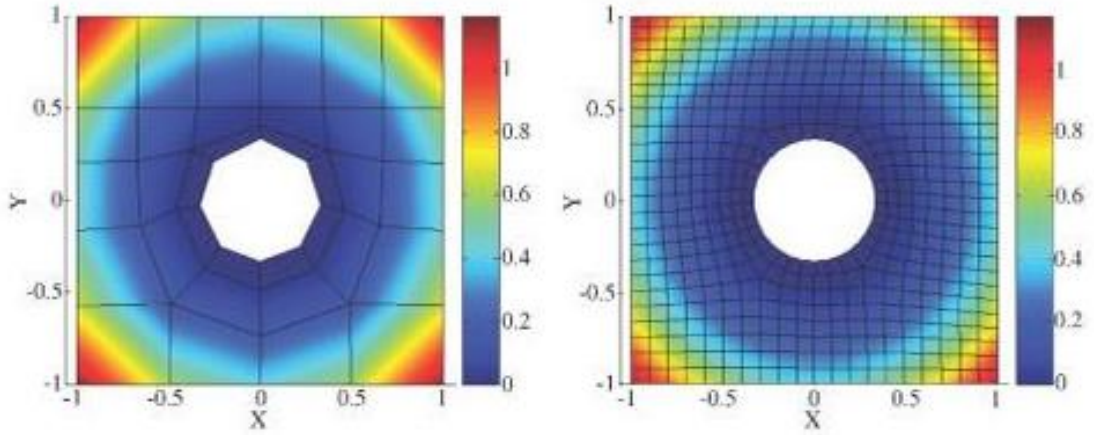
Método de Elementos Finitos: es un método utilizado en ingeniería para hallar magnitudes físicas a problemas en los que no se puede aplicar las fórmulas conocidas. Resuelve problemas como con el análisis de tensión, transferencia de calor, flujo de fluidos entre otros(Johnson & Mathematics, 2009; Logan, 2011).

No es posible aplica una fórmula en el que el problema radica en calcular la tensión de toda la superficie de un plato con un agujero en el medio debido a que la forma es irregular. Sin embargo la fórmula se puede aplicar a formas más sencilla como un triángulo. Basado en esto el problema del plato se puede representar por un número finito de triángulo conectados que representen la forma. Los elementos finitos hacen referencia a que hay una cantidad específica (no infinita) de triángulos que representan a la estructura con tal de simplificar el problema (Gráfica 5).



Gráfica 5 Plato con agujero en el medio

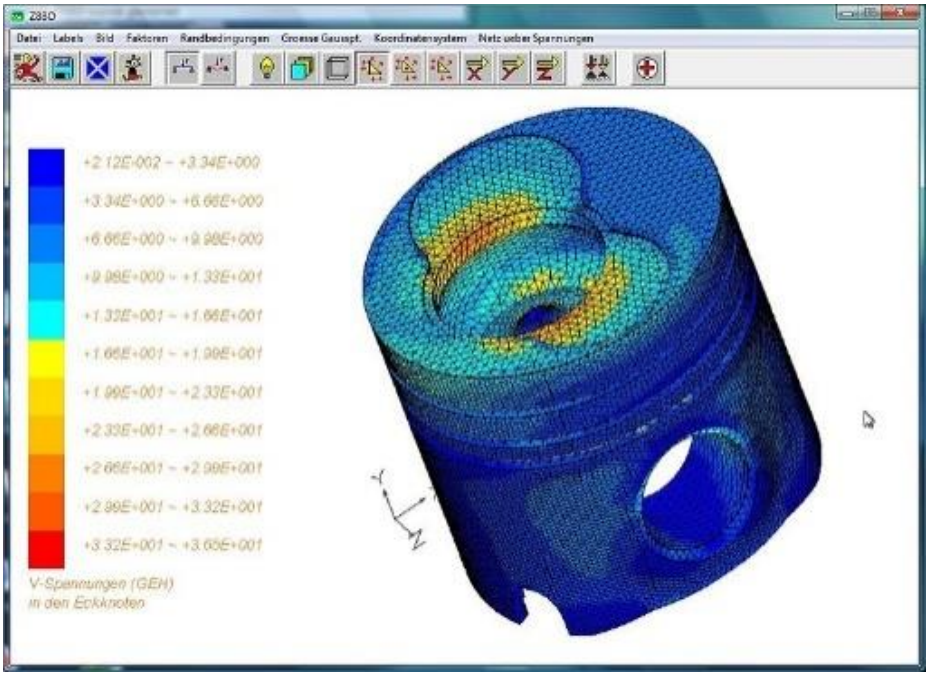
Cada elemento se le proporciona las propiedades del material que se están trabajando. Para el caso del plato sería las propiedades de la tensión del plato, las propiedades para la deformación que rige a cada elemento. Consecuentemente los nodos de cada elemento representan una solución a un sistema de ecuaciones lineales. Que finalmente resolviendo el sistema y usando un programa de cómputo de puede obtener una imagen que representa la tensión del plato donde las zonas rojas indican mayor tensión (ver Gráfica 6).



Gráfica 6 Tensión del Plato

Cuantos más elementos se utilicen la solución al problema es más aproximada. Sin embargo más elementos significan más cálculos. El tiempo requerido para resolver el sistema depende del rendimiento del equipo de cómputo. Computadores más potentes permiten realizar análisis más complejos en menos tiempo

Su aplicación es ampliamente usada para la construcción de estructuras como puentes, edificios, cascos de barcos, industria automotriz (ver Gráfica 7), etc. La precisión de los cálculos depende del número de elemento con los que se represente el objeto a analizar.



Gráfica 7 Tensión de un Pistón

El método de elementos finitos se basa en resolver problemas de ecuaciones diferenciales sobre formas irregulares convirtiéndolo en un sistema de ecuaciones lineales más sencillo de operar. A medida que el poder de cómputo incremente los cálculos serán más precisos con un tiempo prudente de procesamiento.

Capítulo 5

Algoritmos Utilizados

En esta sección se describe los algoritmos implementados por las librerías que se ejecutaron. Las librerías usadas fueron Mumps y Lis, las cuales usaban algoritmos directos e iterativos respectivamente.

El Solver frontal es utilizado por Mumps. Al contrario de la eliminación Gaussiana, evita una cantidad de operaciones que involucran términos cuyo valor es cero. Ajusta la matriz y elimina las ecuaciones tomando un subconjunto de elementos a la vez. Dicho subconjunto es el llamado frontal y es la parte intermedia entre la parte del sistema procesado y la que no ha sido trabajada. Solo las partes ensambladas entran al frontal. La matriz nunca es creada completamente en el proceso. El proceso de la porción frontal implica una operación sobre una matriz densa, esto es que la mayoría de los elementos no son ceros. Computacionalmente la parte frontal es llevada a la memoria principal(Amestoy et al., 2000)

MPI es usado para enviar mensajes asincrónicamente, de manera que un procesador puede enviar dos veces el mismo mensaje pero es recibido en orden por el procesador receptor. Una vez Mumps crea un árbol para las ecuaciones a resolver, usando sustitución hacia atrás resuelve el problema desde los nodos hijos para poder procesar el padre. Es aquí cuando se explota la paralización al resolver varios hijos simultáneamente. Los vectores del árbol a procesar por cada procesador están limitados por el buffer de almacenamiento de cada procesador. Este valor es estimado y puede no coincidir con el

Buffer restante del procesador. Mumps toma un procesador para coordinar las tareas y recibe y ensambla el árbol con todas las soluciones encontradas. (Amestoy et al., 2000)

Los estudios que se han realizado con el algoritmo frontal se han centrado en distribuir eficientemente la memoria en ambientes de paralelización, su aplicación se ha visto en el análisis de elementos finitos, y en trabajos más reciente se intenta escalar el algoritmo hacia la computación en la nube (Amestoy et al., 2000; Balis, Figiela, Malawski, & Jopek, 2015; Calo, Collier, Pardo, & Paszyński, 2011). Su estudio se ha visto aplicado a las ciencias físicas(Paszyński, Pardo, & Paszyńska, 2010a, 2010b; Paszyński, Pardo, Paszyńska, & Demkowicz, 2011), procesos químicos (Scott, 2001; Zitney, Mallya, Davis, & Stad therr, 1996; Zitney & Stadtherr, 1993) y simulaciones (Mallya, Zitney, Choudhary, & Stadtherr, 1999; Scott, 2001).

El Solver iterativo de Lis utiliza el algoritmo del gradiente conjugado. Este método iterativo usa una secuencia de vectores ortogonales. Estos vectores son el residuo de las iteraciones. Consecuentemente usa las direcciones de búsquedas(ρ) para actualizar las por un múltiplo (α) iteraciones y los residuos. El tamaño de las secuencias puede crecer los suficiente, pero solo un pequeño número de vectores es mantenido en memoria (Gráfica 8).

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)} = Ap^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence; continue if necessary
end

Gráfica 8 Algoritmo de Gradiente conjugado

Lis usa MPI para sincronizar las tareas entre los procesadores. Elige un procesador maestro que recibe y organiza los resultados de los demás procesadores al final de la solución. Las secuencias de soluciones x^0 para una solución inicial $x^{(0)}$son enviadas a cada procesador para resolver al mismo tiempo el problema.

Los últimos estudios se han enfocado en mejorar la eficiencia del algoritmo en áreas de ingeniería. Se trata de mejorar su estabilidad y la escalabilidad a múltiple procesadores mejorando el algoritmo usando otras técnicas como el método del paso descendente (Bouwmeester, Dougherty, & Knyazev, 2015; Linjun Wang, Cao, Han, Liu, & Xie, 2015; Yuan & Zhang, 2015). Se han hecho híbridos con otros métodos para

simplificar problemas de ingeniería específicos como identificación de carga en el método de elementos finitos (Linjun Wang et al., 2015; Liumei Wang, Sun, de Sampaio, & Yuan, 2015). También se han hecho estudios de cómo aprovechar el algoritmo con procesamiento en paralelo con tarjetas dedicadas de video (Gao, Liang, & Wang, 2014; Lang & Rünger, 2014; X. Li & Li, 2014)

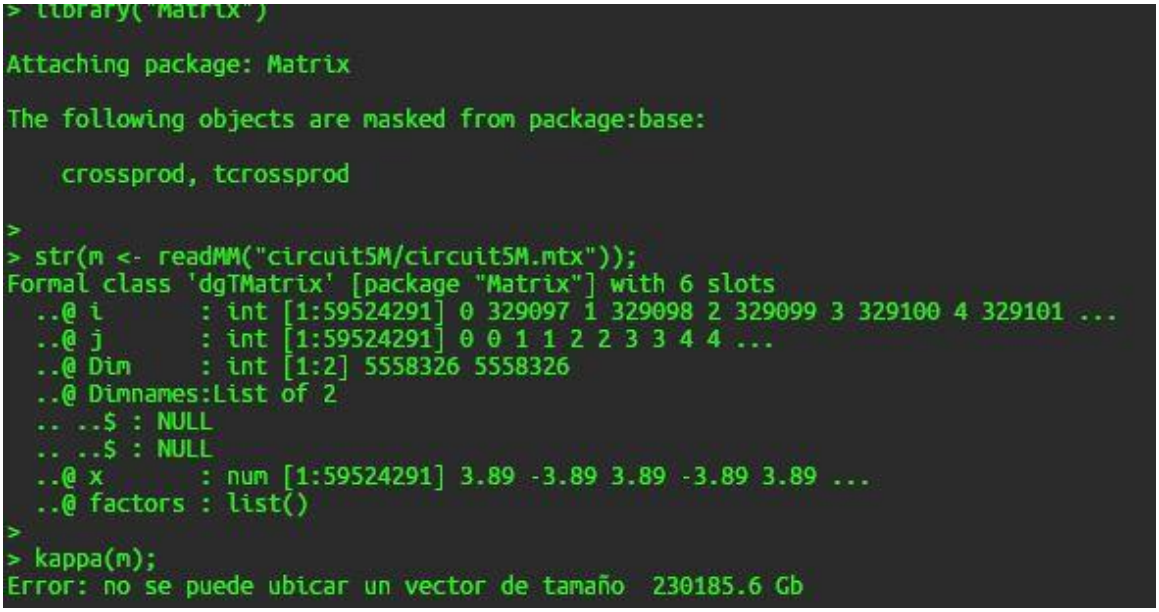
Capítulo 6

Cálculo Número de condición, banda ancha y radio espectral

Tal como se presentó en el Marco Teórico el número de condición es una medida de la matriz usada para saber si la misma está bien condicionada. En el caso de la solución de sistemas lineales reales serviría para verificar su influencia al resolver dichos sistemas lineales. Sin embargo no se pudo calcular el número de condición para las matrices porque no se podía alojar suficiente memoria. El cálculo se hizo con el programa Octave. En algunos casos era necesario más de 200GB (ver Gráfica 9) y en otros problemas más grandes era necesario contar con más de veinte mil gigabytes en memoria (ver Gráfica 10). Con Matlab presentaba el mismo resultado. Este exceso de memoria se debe a que es necesario invertir la matriz que en este caso son de gran tamaño.

```
> library("Matrix")

Attaching package: Matrix

The following objects are masked from package:base:

    crossprod, tcrossprod

>
> str(m <- readMM("scircuit/scircuit.mtx"));
Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
  ..@ i       : int [1:958936] 0 12048 18842 19270 23487 24987 27266 27956 28481 72256 ...
  ..@ j       : int [1:958936] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ Dim     : int [1:2] 170998 170998
  ..@ Dimnames:List of 2
  .. ..$ : NULL
  .. ..$ : NULL
  ..@ x       : num [1:958936] 2.32e-01 -4.90e-06 -4.89e-06 -4.90e-06 -4.90e-06 ...
  ..@ factors : list()
>
> kappa(m);
Error: no se puede ubicar un vector de tamaño  217.9 Gb
```

Gráfica 9 Cálculo de número de condición para la matriz scircuit



Gráfica 10 Cálculo del número de condición para la matriz circuit5M

El cálculo del ancho de banda se hizo también con el programa Octave. Se usó la función "bandwidth" para obtener las diagonales superiores e inferiores. El ancho de banda se obtuvo sumando las diagonales superiores, las inferiores y la diagonal principal. Para hallar el porcentaje de confinación de los elementos no ceros dentro del rango de la banda ancha se desarrolló un programa para obtener la cantidad de elementos que contiene la región limitada (ver Anexo 14), seguido se obtuvo el cociente entre la cantidad de elementos no ceros y la cantidad total dentro de la banda ancha.

Por último el radio espectral se obtuvo con "max(abs(eigs(A)))" donde "A" es la matriz leída con la función "readMM". En las matrices de gran tamaño usadas el radio espectral no convergía por tanto se pudo obtener el valor.

Capítulo7

Implementación.

La literatura tiene un gran número de librerías con rutinas y subrutinas dedicadas a los procedimientos para resolver sistemas lineales. En el proyecto después de varios intentos se utilizaron aquellos que tenían las librerías embebidas. Las herramientas que se lograron configurar fueron Mumps y Lis(X. S. Li et al., 1999).

De acuerdo al funcionamiento de estas librerías se podría clasificar con las arquitecturas de paralelización SIMD (del inglés Single Instruction, Multiple Data, en español: "una instrucción, múltiples datos") pues por cada porción de la matriz es procesada con intrucciones iguales(M. Flynn, 1966; Michael Flynn, 2007). Lis utiliza método iterativo por cada entrada de datos en el procesador(Nishida, 2010); mientras que Mumps utiliza una variante de la eliminación gausiana para hacer el trabajo(Amestoy et al., 2000) y SuperLu implementa factorización Lu y resuelve la martiz triangular por eliminación hacia adelante y ahacia atrás (Xiaoye S. Li, 2005; X. S. Li et al., 1999).

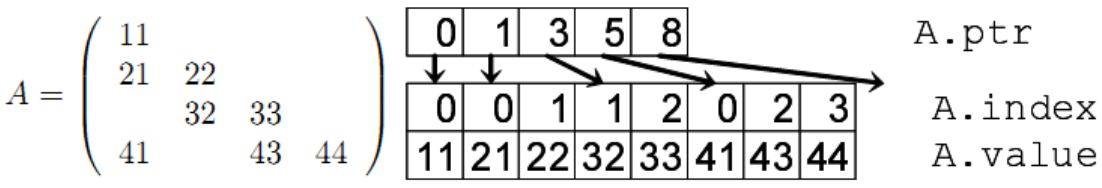
La compilación necesitaban prerrequisitos como compiladores G++ o Fortran (F77) lo cual daba complejidad para solo usar las librerías. En todos los casos se hacía necesaria una serie de pasos para tener lista la librería. Las tareas básicas que se necesitaron fueron:

- Descargar la librería
- Compilar
- Ver errores

- Bajar dependencias
- Recompilar
- Verificar que la instalación haya sido exitosa

La compilación podía durar horas debido a que necesitaba descargar las librerías necesarias y luego volver a compilar. Para realizar estas actividades era necesario tener conocimiento de Linux en la parte de línea de comandos, compilación de paquetes, descarga de paquetes, manejador de permisos y las herramientas como CMAKE. Una vez la herramienta era compilado se continuaba a programar sobre la librería para resolver la ecuación.

Se encontró que la matriz de la forma NxN debía ser representada en un formato especial, para algunas librerías. El formato Harwell-Boeing era el más usual y el más eficiente. Ninguna de las librerías manejada el formato Coordinate Text File. En el Harwell-Boeing se simplifica las filas (compressed Sparse column o CSC) usando el índice donde empieza la siguiente fila (Ptr.) y con un arreglo de índices (index) se indica a que columna pertenece cada valor.



Gráfica 11 Compressed SparseRow

Configuración Trilinos Mumps

Se usó Trilinos para configurar Mumps. Trilinos es un Framework que alberga varias librerías para ejecutarlas desde su entorno. Además se hizo necesario compilar el módulo PyTrilinos para ejecutar las instrucciones de la Librería en Python. También se tuvo que descargar Mumps para enlazarlo con el Framework.

Mumps es un software escrito en C y fortran para la solución de ecuaciones lineales dispersas usando el Métodos directos. Dos versiones de la librería están disponibles, una de ellas es la versión secuencial, la otra trabaja masivamente en paralelo usando MPI. En ambas versiones utiliza utilizan la variante de la eliminación de Gaussiana, el método frontal. El método frontal, al contrario de la eliminación Gaussiana, evita una cantidad de operaciones que involucran términos cuyo valor es cero. Ajusta la matriz y elimina las ecuaciones tomando un subconjunto de elementos a la vez. Dicho subconjunto es el llamado frontal y es la parte intermedia entre la parte del sistema procesado y la que no ha sido trabajada. Solo las partes ensambladas entran al frontal. La matriz nunca es creada completamente en el proceso. El proceso de la porción frontal implica una operación sobre una matriz densa, esto es que la mayoría de los elementos no son ceros. Computacionalmente la parte frontal es llevada a la memoria principal(Amestoy et al., 2000).

Como es de esperarse hace uso del paralelismo para procesar varias partes frontales a la vez. MPI es usado para sincronizar las tareas con cada procesador. La escalabilidad del sistema dependería de la complejidad de la matriz y en la arquitectura del hardware que se implemente. Como su implementación se basa en un método directo

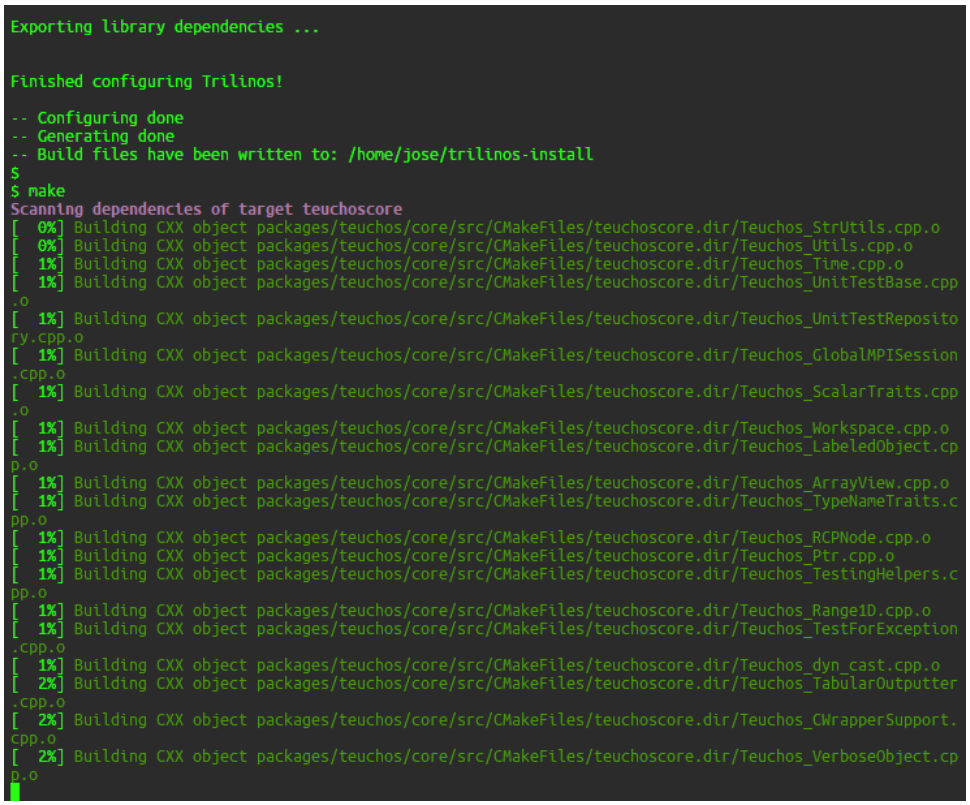
probable que ciertos sistemas no sean solucionables por la demanda en el número de incógnitas.

```
$ cmake -D Trilinos_ENABLE_TESTS:BOOL=ON -D Trilinos_ENABLE_EXAMPLES:BOOL=ON -D TPL_ENABLE_MPI:BOOL=O
N -D TPL_ENABLE_MUMPS:BOOL=ON -D Trilinos_ENABLE_Epetra:BOOL=ON -D Trilinos_ENABLE_EpetraExt:BOOL=ON
-D CMAKE_INSTALL_PREFIX:STRING="/home/jose/trilinos-install"    -D Trilinos_ENABLE_PyTrilinos:BOOL=ON
 -D BUILD_SHARED_LIBS:BOOL=ON -D Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON -D TPL_BLAS_LIBRARIES:
STRING=/usr/lib/libblas.so -D TPL_LAPACK_LIBRARIES:STRING=/usr/lib/liblapack.so  -D SWIG_EXECUTABLE:S
TRING=/home/jose/swiginstall/bin/swig  -D  Trilinos_ENABLE_AztecOO:BOOL=ON -D TPL_ENABLE_UMFPACK:BOOL
=ON -D TPL_UMFPACK_INCLUDE_DIRS="/usr/include/suitesparse/" -D Amesos_ENABLE_SCALAPACK:BOOL=ON -D TPL
_SCALAPACK_LIBRARIES:FILEPATH="/usr/lib/libscalapack-openmpi.so" ../trilinos-11.8.1-Source

Configuring Trilinos build directory

-- PROJECT_SOURCE_DIR='/home/jose/trilinos-11.8.1-Source'
-- PROJECT_BINARY_DIR='/home/jose/trilinos-install'
-- CMAKE_VERSION = 2.8.12.2
-- Found PythonInterp: /usr/bin/python2.7 (Required is at least version "2.4")
-- Python version 2.7.6
-- PYTHON_EXECUTABLE='/usr/bin/python2.7'

Setting up major user options ...

--   Setting Trilinos_ENABLE_TriKota=OFF because '/home/jose/trilinos-11.8.1-Source/./packages/TriKot
a/Dakota' does not exit!

Reading the list of packages from /home/jose/trilinos-11.8.1-Source/./PackagesList.cmake

-- Trilinos_NUM_PACKAGES='55'

Reading the list of TPLs from /home/jose/trilinos-11.8.1-Source/./TPLsList.cmake

-- Trilinos_NUM_TPLS='81'

Processing Project, Repository, and Package dependency files and building internal dependencies graph
 ...

-- Trilinos_NUM_SE_PACKAGES='115'
-- Tentatively enabling TPL 'Pthread'

Explicitly enabled packages on input (by user):  Epetra EpetraExt AztecOO PyTrilinos 4

Explicitly enabled SE packages on input (by user):  Epetra EpetraExt AztecOO PyTrilinos 4

Explicitly disabled packages on input (by user or by default):  Trios ShyLU TriKota ForTrilinos 4

Explicitly disabled SE packages on input (by user or by default):  KokkosCore KokkosCompat KokkosCont
ainers KokkosLinAlg KokkosExample KokkosMpiComm KokkosTask Trioscommsplitter Triossupport Triosnnti T
riosnssi Triosprograms Triosexamples Triostests Triosnetcdf-service Trios ShyLU TriKota ForTrilinos 1
9

Explicitly enabled TPLs on input (by user):  Pthread MPI UMFPACK MUMPS 4

Explicitly disabled TPLs on input (by user or by default):  0

Disabling all packages that have a required dependency on disabled TPLs and optional package TPL supp
ort based on TPL_ENABLE_<TPL>=OFF ...


Enabling or disabling subpackages for hard enables/disables of parent packages Trilinos_ENABLE_<PAREN
T_PACKAGE>=ON/OFF ...


Disabling forward required packages and optional intra-package support that have a dependency on dis
```

Gráfica 12 Usando Cmake para configurar Trilinos con MUMPS

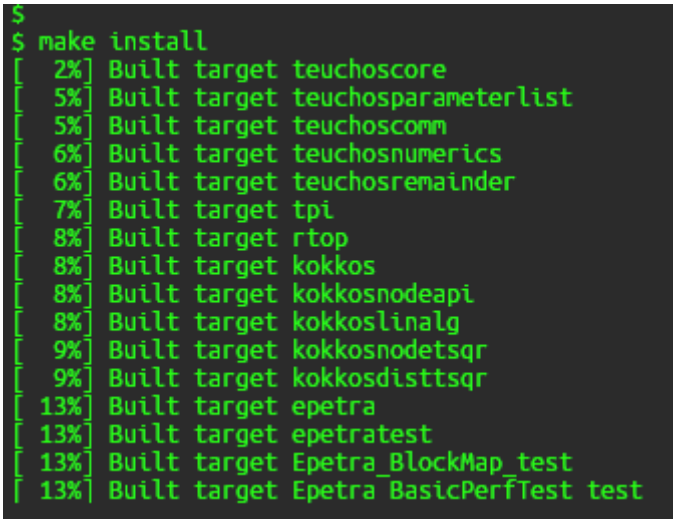
La instalación de Trilinos requiere el programa Cmake. Se necesita indicar al código fuente que paquetes debe compilar para preparar la distribución en el sistema. La configuración con Mumps requirió que se habilitaran paquetes y algunas configuraciones

requeridas. Se indicó la ubicación de las librerías dentro del sistema de las librerías BLAS, LAPACK, SWIG, entre otros; además de indicó que se compilara con MPI. MUMPS propiamente se habilitó con la instrucción "-D TPL_ENABLE_MUMPS: BOOL=ON" (ver Gráfica 12). La ejecución del comando llevaba un tiempo prudente.



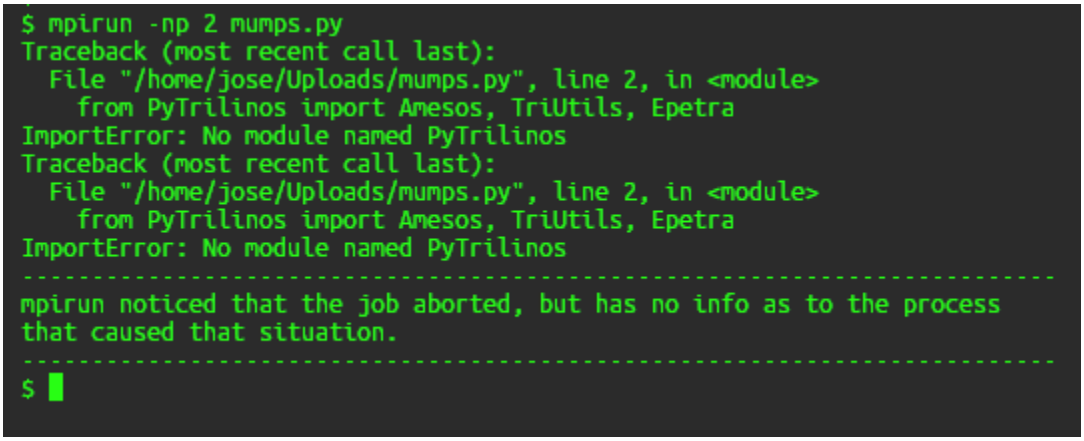
Gráfica 13 ConstruyendoTrilinos, usando el comando make

Una vez el proceso anterior se llevó a cabo se procedió a construir los objetos con el comando "MAKE" como se observa en la Gráfica 13. El proceso tomó un tiempo mucho mayor, Los errores que presentaron inicialmente se debían a que no podía encontrar las librerías como BLAS o LAPACK. En otros casos porque la versión algún programa era inferior o superior a la necesitada.



Gráfica 14 ConstruyendoTrilinos, usando el comando make install

En la Gráfica 14se muestra la instalación de Trilinos. Este paso se ejecutó una vez se hubieron realizado los pasos anteriores. Generalmente se presentaba problemas si la ubicación donde se intentaba instalar los paquetes no tenía permisos para el usuario. En ese caso se usaba el comando SUDO. En las pruebas no se hizo necesario porque se optó instalarlo en una carpeta local.



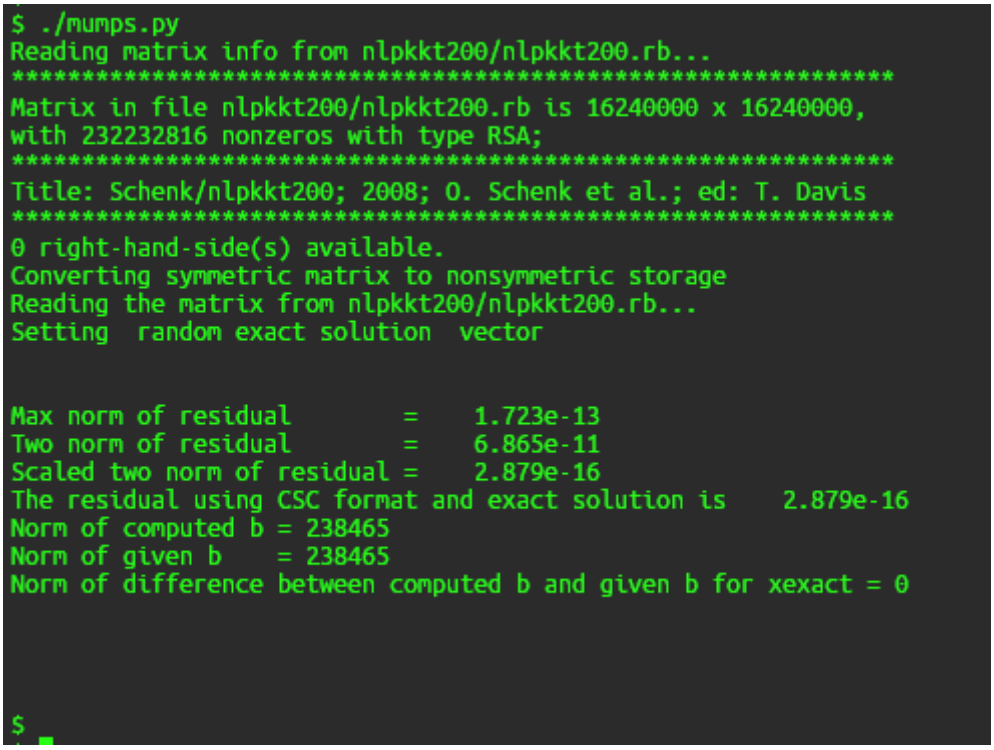
Gráfica 15 Error al intentar ejecutar un script con Mumps

Se intentó ejecutar el script que se usó en el transcurso de las pruebas con las diferentes matrices. El programa abortaba porque no podía encontrar el módulo

PyTrilinos a pesar de haber sido compilado dicho modulo, el error provocado se pude apreciar en la Gráfica 15.

Gráfica 16 Variables de entorno para Trilinos

En la Gráfica 16 se muestran dos comandos que se utilizaron para que Trilinos reconociera los módulos que se habían instalado. Se necesitaban configurar las variables de entorno para Python y de igual forma la librería de Trilinos. El primer comando le indicaba a Python de los módulos de Trilinos; el siguiente comando es para decirle al sistema donde se encontraban las librería de Trilinos. En el Anexo 1 puede verse todas las anotaciones que se tomaron para realizar toda la configuración de Trilinos con Mumps antes de llegar al resultado final.



Gráfica 17 Ejecución del script mumps.py

Con las variables de entorno debidamente configuradas el script se ejecutó correctamente. En la Gráfica 17 se observa que el script ejecutó una matriz de 16 millones de incógnitas. Esta matriz es una de las que se usó en el transcurso de este proyecto.

```
#! /usr/bin/env python
from PyTrilinos import Amesos, TriUtils, Epetra
Comm = Epetra.PyComm()
Map, A, x, b, Exact = TriUtils.ReadHB("nlpkkt200/nlpkkt200.rb", Comm)
#print b
Problem = Epetra.LinearProblem(A, x, b)
Factory = Amesos.Factory()

SolverType = "Mumps"

Solver = Factory.Create(SolverType, Problem)
AmesosList = {
"MaxProcs": 32,
"PrintStatus": True
}

Solver.SetParameters(AmesosList)
Solver.SymbolicFactorization()
Solver.NumericFactorization()
Solver.Solve()

file = open("newfile.txt", "w")
for i in x:
  file.write(str(i)+"\n")
  #print i
file.close()
#print len(x)
#print x
```

Gráfica 18 Archivo mumps.py

El archivo mumps.py contiene las instrucciones para leer la matriz de un archivo en formato Harwell Boeing y resolver el sistema de ecuaciones (el contenido se puede ver en la Gráfica 18 y se agrega al Anexo 2 una versión que contiene opciones que se omitieron). La primera línea importa del módulo PyTrilinos los objetos que se van a utilizar. Para lograr la comunicación entre procesadores usando MPI se crea la variable Comm. La lectura del archivo se hace en la tercera línea y se guarda las variables para resolver el sistema lineal, estas son la matriz, y el vector con las soluciones del sistema. La siguiente instrucción crea el objeto problema, este contiene la información de la matriz, el vector b, y el vector X (por defecto en cero).Luego se crea el Solver en el cuál

se configura la librería Mumps, se configura el máximo número de procesadores que se pueden ejecutar con MPI y las instrucciones para resolver el sistema. La última parte del script toma los valores que están en la variable X y las escribe en un archivo. Cada línea representa el índice del vector.

Configuración SuperLu

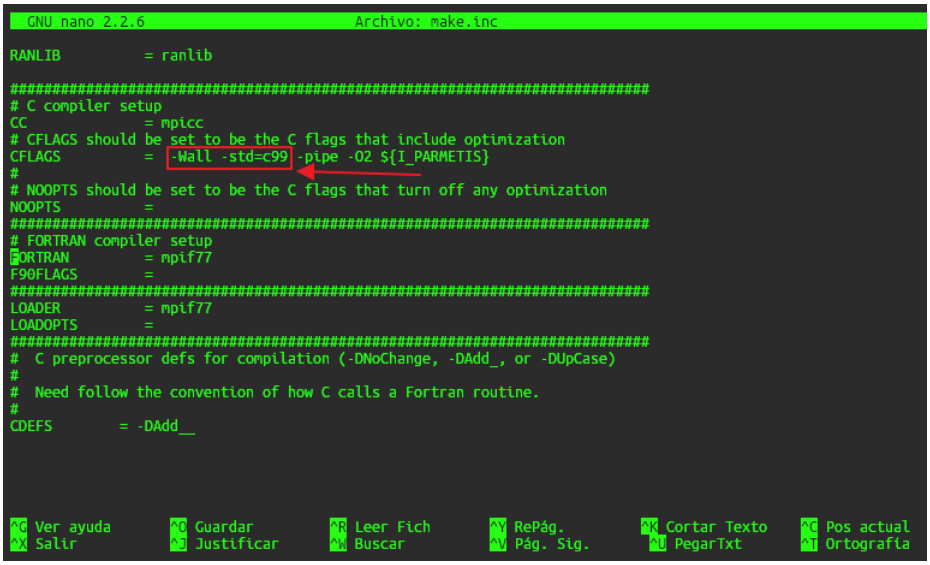
SuperLu se describe como una librería para la solución de sistemas lineales extensos, no simétricos, dispersos en sistemas de cómputo de alto rendimiento. La librería está escrita en C y es ejecutable tanto desde una aplicación en C o en Fortran. Realiza una factorización LU y resuelve el sistema triangular a través de sustitución hacia adelante y hacia atrás(Xiaoye S. Li, 2005).

Tres versiones, específica para tres arquitecturas de hardware, están disponibles para su implementación. SuperLu está se utiliza en máquinas en las que no se cuenta con recursos para paralelizar los procesos. SuperLu_MTaprovecha los sistemas de cómputo que poseen varios procesadores y comparten una sola memoria principal. Para Clústeres en donde cada procesador tiene memoria independiente de los demás la librería ofrece la versión para memoria distribuida SuperLU_DIST. La documentación está inmersa en el código fuente. Otras documentaciones están incompletas y bajo construcción en la página oficial del proyecto. Para el propósito del estudio de la librería se usó la de memoria distribuida.

```
$ wget http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/parmetis-4.0.3.tar.gz
$ tar -xvf parmetis-4.0.3.tar.gzta ci
$ cd parmetis-4.0.3
$ make
$ make config prefix=~/parmetisinstall
$ make
$ make installs
```

Gráfica 19 Instalación Parmetis

Como requisito se tuvo que instalar Parmetis. No existía una distribución así que se procedió a instalarlo desde el código fuente. Se siguió las instrucciones del archivo Install.txt. La Gráfica 19 muestra los comandos usados para instalar Parmetis. Los dos primero son usados para descargar y descomprimir. Los comandos make siguientes configuran la distribución y lo instala.



Gráfica 20 Cambios en el archivo make.inc de SuperLU_DIST

Se descargó y descomprimió SuperLU_DIST. Debido a unos errores que se presentaron y que estaban relacionados con el compilador C99 se tuvo que agregar las instrucciones que se presentan en la Gráfica 20 y en el Anexo 3 el archivo make.inc completo.

```
$ cd SuperLU_DIST_4.0/
$ cp MAKE_INC/make.i386_linux  make.inc
$
$ ls MAKE_INC/
make.altix    make.i386_linux  make.origin  make.sp.64bit  make.xe6  make.xt4.64bit      make.xt4_pgi
make.carver   make.opteron     make.sp      make.t3e       make.xt4  make.xt4_pathscale  make.xt5
$
```

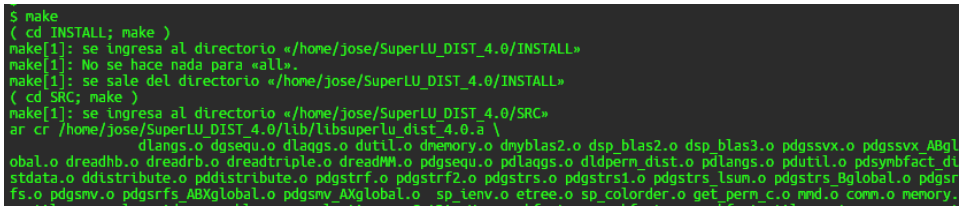
Gráfica 21 Copiando el archivo make adecuado

El archivo make.inc se copió de la ruta que especifica Gráfica 21, de acuerdo a las instrucciones del archivo README que se encuentra en la raíz del código fuente. Los otros archivos que estaban en la carpeta MAKE_INC tenían configuraciones para arquitectura de clústeres como xt4 o xt5 que no eran compatibles con el equipo.

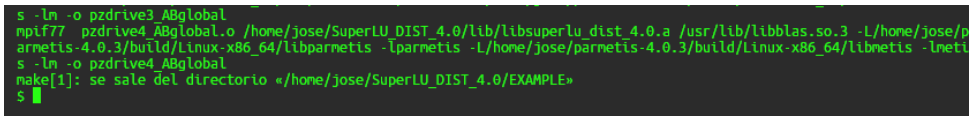
```
  GNU nano 2.2.6                        Archivo: make.inc

#                                    General Dynamics - Network Systems
#                        works for i386 Linux, with LAM-MPI 7.1.1 and GCC 4.
#
############################################################################
#
#  The machine (platform) identifier to append to the library names
#
PLAT            = _i386

#
#  The name of the libraries to be created/linked to
#
DSuperLUroot    = ${HOME}/SuperLU_DIST_4.0
DSUPERLULIB     = $(DSuperLUroot)/lib/libsuperlu_dist_4.0.a
#
BLASDEF         = -DUSE_VENDOR_BLAS
BLASLIB         = /usr/lib/libblas.so.3
METISLIB = -L/home/jose/parmetis-4.0.3/metis/include/ -lmetis

############################################################################
## parmetis 4.x.x, 32-bit integer
PARMETIS_DIR    := ${HOME}/parmetis-4.0.3
## parmetis 4.x.x, 64-bit integer
# PARMETIS_DIR  := ${HOME}/Carver/lib/parmetis-4.0.3_64

METISLIB := -L${PARMETIS_DIR}/build/Linux-x86_64/libmetis -lmetis
PARMETISLIB := -L${PARMETIS_DIR}/build/Linux-x86_64/libparmetis -lparmetis

^G Ver ayuda      ^O Guardar       ^R Leer Fich     ^Y RePág.        ^K Cortar Texto   ^C Pos actual
^X Salir          ^J Justificar    ^W Buscar        ^V Pág. Sig.     ^U PegarTxt       ^T Ortografía
```

Gráfica 22 Archivo make.inc

También se tuvo que especificar donde estaba ubicado el código fuente del SuperLu y la ubicación del Parmetis, que este caso estaba instalado localmente y el sistema no puede ayudar con su ubicación.

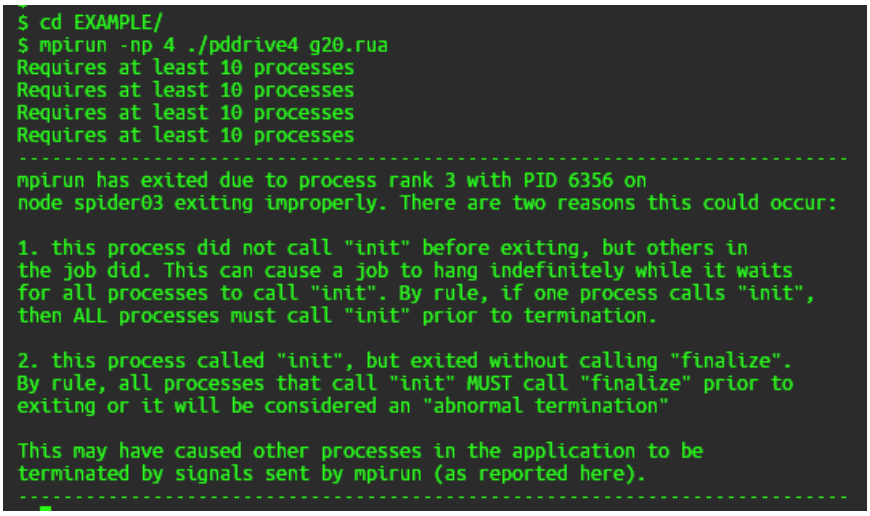


Gráfica 23 Salida inicial del comando make en SuperLU_DIST



Gráfica 24 Salida final del comando make en SuperLU_DIST

Para compilar se ejecutó el comando “make” en la capeta del código fuente. Generó un registro de acciones de las instrucciones que usaba para construir la herramienta. En la Gráfica 23 se observa las primeras instrucciones que realiza. El final del proceso se está en la Gráfica 24. Al final la compilación terminó sin errores. El Anexo 4 muestra los comandos utilizados para instalar SuperLu



Gráfica 25 Ejecutando ejemplo de SuperLU_DIST

Se probó un ejemplo que es totalmente funcional para resolver ecuaciones lineales a partir de un archivo en formato Harwell Boeing. Se usó MPI con cuatro procesadores sobre la matriz g20.rua. No se obtuvo resultados, el programa requería al menos 10 procesadores. Al usar diez procesadores el programa mostraba información que pareciera repetida. El código fuente ppddrive4.c, igual que los otros contenidos en la carpeta, tenían comentarios para realizar una implementación independiente, sin embargo los resultados de los ejemplos no fueron entendibles y la documentación no era suficientemente clara.

```
 * On an IBM SP, the program may be run by typing
 *    poe pddrive -r <proc rows> -c <proc columns> <input_file> -procs <p>
 * </pre>
```

Gráfica 26 Documentación que explica la ejecución en un sistema IBM SP

La documentación en el código fuente se centraba en computadores especiales para trabajo en paralelo como IBM SP (ver Gráfica 26). SP son las siglas en inglés de Scalable POWERparallel, es un computador de alto desempeño desarrollado IBM con un software especial para clústeres. Esto muestra que la documentación está destinado arquitecturas de clústeres diseñada con ese propósito. Si se observa en los pasos anteriores cuando se intentó copiar el archivo make.inc todas las opciones eran para sistemas de cómputo de alto desempeño con una arquitectura y software definido (XT4, XT5, etc.).Este motivo llevo a que no se siguiera usando esta librería para las pruebas, puesto que no se obtenían resultados y en cambio se generaban errores.

Configuración Lis

El Solver Lis usa métodos iterativos para resolver ecuaciones lineales. Su instalación no necesita de librerías como LAPACK o BLAS. Se compila utilizando un compilador de C. Para la ejecución en paralelo utiliza OpenMP o MPI. En el primer caso recomendable cuando el ambiente de la computación en paralelo se basa en los núcleos del procesador. Cuando se usan varios procesadores entonces se usa la librería MPI. Las librerías de paralización como los compiladores, son configuradas por el usuario para generar la configuración adecuada. Lis acepta los formatos de matrices Harwell Boeing y Matrix Market(Kotakemori et al., 2008; Nishida, 2010; Pujii, Nishida, & Oyanagi, 2005).

```
$ sudo apt-get install libcr-dev mpich2 mpich2-doc
Reading package lists... Done
Building dependency tree
Reading state information... Done
libcr-dev is already the newest version.
mpich2 is already the newest version.
mpich2-doc is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 173 not upgraded.
```

Gráfica 27 Instalando dependencias de Lis

Como se dijo anteriormente Lis no necesita librerías externas de algebra lineal. Sin embargo si tiene algunas dependencias. Se usó el administrador de paquetes para instalas directamente en el sistema (ver Gráfica 27).

```
$ cd lis-1.4.54/
$
$ ./configure --prefix=/home/jose/lis-install --enable-mpi
```

Gráfica 28 Configurando Lis con MPI

```
config.status: executing depfiles commands
config.status: executing libtool commands

  Build with OpenMP library                = no
  Build with MPI library                   = yes
  Enable FORTRAN 77 compatible interface   = no
  Enable Fortran 90 compatible interface   = no
  Enable SA-AMG preconditioner             = no
  Enable double-double precision support   = no
  Enable long double precision support     = no
  Enable 64bit integer support             = no
  Enable dynamic linking                   = no
  Enable profiling                         = no

  C compiler         = mpicc
  C flags            = -O3 -fomit-frame-pointer  -DUSE_MPI -DHAVE_CONFIG_H
  C libraries        = -lm   -L/usr//lib -L/usr/lib/openmpi/lib -L/usr/lib/gcc/x86_64-linux-gnu/4.8 -L/usr/lib/gcc/x86_
64-linux-gnu/4.8/../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../../lib -L/lib/x86_64-linux-gnu
 -L/lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/4.8/../../.. -lmpi -ldl -
lhwloc -lgcc_s -lpthread
  MPI libraries      =
  MPIRUN script      = mpirun

$
```

Gráfica 29 Configuración de Lis con la librería MPI

```
$ make
$ make check
$ make install
$ make installcheck
```

Gráfica 30 Comandos para instalar Lis

Para definir a Lis la ubicación de instalación y que agregara soporte MPI se usó el comando configure. En la Gráfica 28 se especifica la ubicación de la ruta de instalación en lis-install que debe construirse con MPI. El final de la ejecución de este comando se aprecia en la Gráfica 29en donde efectivamente MPI se integró a la librería. La instalación se realizó con los comandos que aparecen en la Gráfica 29.Después de esto se ejecutó unos scripts de ejemplo para verificar que MPI no presentaba inconvenientes.

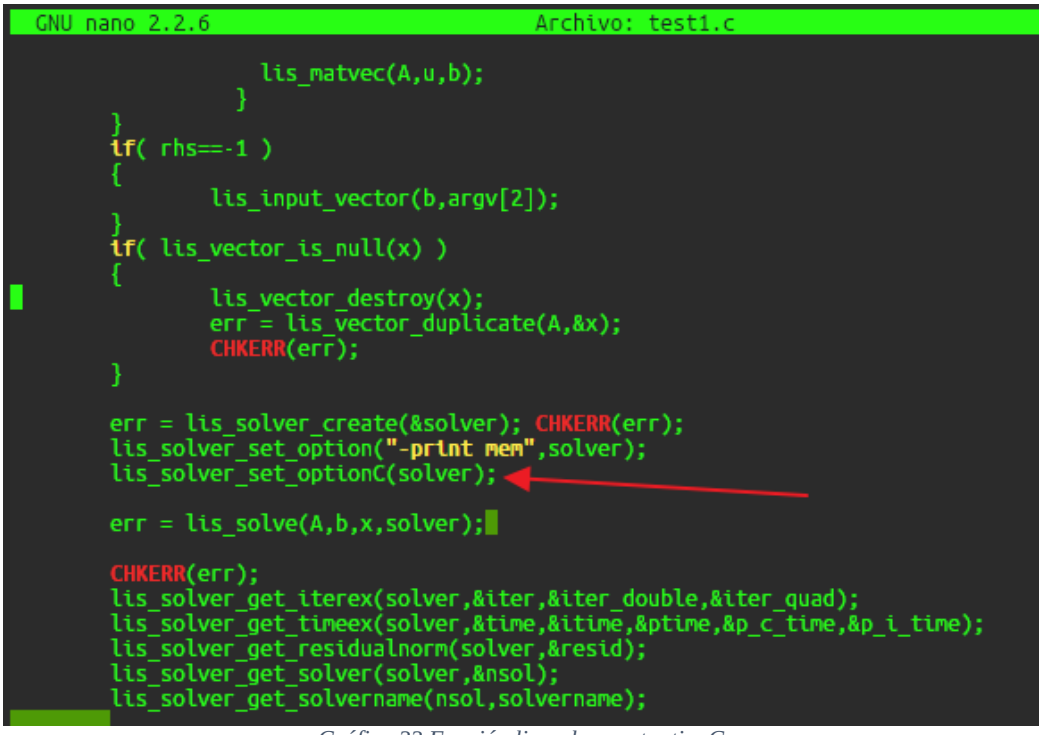
```
$ cd test
$
$ mpirun -np 4 ./test1  testmat.mtx 0 solution.txt rhistory.txt

number of processes = 4
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
conv_cond : ||b-Ax||_2 <= 1.0e-12 * ||b-Ax_0||_2
storage   : CSR
lis_solve : normal end

BiCG: number of iterations = 15 (double = 15, quad = 0)
BiCG: elapsed time         = 6.508827e-04 sec.
BiCG:   preconditioner     = 9.059906e-06 sec.
BiCG:     matrix creation  = 9.536743e-07 sec.
BiCG:   linear solver      = 6.418228e-04 sec.
BiCG: relative residual    = 1.457895e-16
```

Gráfica 31 Prueba con Lis

La prueba de laGráfica 31 muestra que Lis se instaló sin ningún problema. Los resultados los guarda en el archivo solution.txt. Este script se utiliza, también, en las pruebas realizadas con las diferentes matrices.

```
$ mpirun -np 4 ./test1  testmat.mtx 0 solution.txt rhistory.txt -maxiter 10000

number of processes = 4
matrix size = 100 x 100 (460 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
conv_cond : ||b-Ax||_2 <= 1.0e-12 * ||b-Ax_0||_2
storage   : CSR
lis_solve : normal end

BiCG: number of iterations = 15 (double = 15, quad = 0)
BiCG: elapsed time         = 5.838871e-04 sec.
BiCG:   preconditioner     = 5.006790e-06 sec.
BiCG:     matrix creation  = 0.000000e+00 sec.
BiCG:   linear solver      = 5.788803e-04 sec.
BiCG: relative residual    = 1.457895e-16
```

Gráfica 32 Aumentando el número de iteraciones



Gráfica 33 Función lis_solver_setoptionC

Los parámetros son modificables por la línea de comando. Por ejemplo se puede cambiar la tolerancia como se muestra en la Gráfica 32. Esas opciones se pasan por línea de comando porque el código fuente utiliza la función “lis_solver_setoptionC” que precisamente toma los argumentos que envíe el usuario y los envía al Solver. En la Gráfica 33 se observa el lugar donde se utiliza la función dentro del archivo test1.c. El Anexo 5 muestra en orden y con comentarios los pasos utilizados para configurar la librería.

Archivo de matriz

Las matrices se guardan en texto plano. Generalmente en el archivo se indica los índices de los coeficientes de la matriz. En el mismo archivo o en uno independiente se almacena las variables independientes. Para los problemas en donde la matriz es dispersa solo se guarda los datos cuyo coeficiente es cero.

$$\begin{bmatrix} 1 & & & & & \\ & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix} X = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Gráfica 34 Matriz de dimensión seis

La matriz que se muestra en la Gráfica 34 se va a utilizar para mostrar su representación en los formatos que se describen más adelante.

Formato Matrix Market

El formato más utilizado es Matrix Market por la simpleza para generar y guardar la matriz. Denota con índices la posición de los valor no cero del sistema. Un índice indica la fila dentro de la matriz y el siguiente la columna. El vector de variables independientes se guarda en un archivo independiente donde un índice por cada valor representa la fila a la fila a la que pertenece dentro de la matriz(Boisvert, Pozo, Remington, Barrett, & Dongarra, 1996; “Matrix Market Exchange,” n.d.).

Una versión extendida de Matrix Market permite guardar el vector de variables independientes en el mismo archivo donde se almacena los coeficientes no ceros de la matriz dispersa. Una simple deducción hace inferir que la versión extendida es siempre la mejor a considerar; pero en algunos casos resulta más conveniente crear los archivos por separado. Cuando la matriz es considerablemente grande no es sencillo abrir el archivo porque el editor no maneja la cantidad de memoria para operar. Si se necesita cambiar los resultados de la matriz es más sencillo generar solo el archivo de las variables independientes.

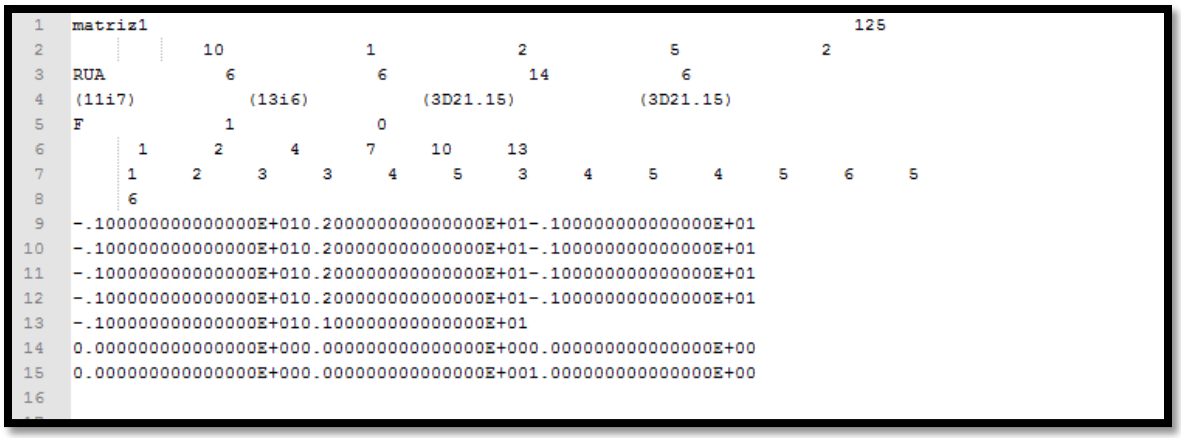
```
1  %%MatrixMarket matrix coordinate real general
2  6 6 14 0 0
3  1 1 1.0
4  2 2 2.0
5  2 3 -1.0
6  3 2 -1.0
7  3 3 2.0
8  3 4 -1.0
9  4 3 -1.0
10 4 4 2.0
11 4 5 -1.0
12 5 4 -1.0
13 5 5 2.0
14 5 6 -1.0
15 6 5 -1.0
16 6 6 1.0
```

Gráfica 35 Archivo Matriz Market

EL archivo de la Gráfica 35 es la representación en Matrix Market de la matriz de la ecuación descrita en Gráfica 34.El encabezado de la primera línea indica que se está guardando los coeficientes de la matriz. La segunda línea indica que es una matriz de seis por seis con catorce elementos no nulos. Los últimos dos números, que este caso son ceros, indican que este archivo no tiene el vector de soluciones. El Anexo 6 muestra este archivo.

Formato Harwell Boeing

El formato Harwell Boeing almacena la información de la matriz al omitir los índices de las columnas. Para ello utiliza tres vectores para almacenar la información de la matriz dispersa. Un vector tiene los datos de los coeficientes de valor no cero de la matriz dispersa. Los datos son ordenados por columnas. El primer vector indica en qué posición del vector de datos inicia la siguiente columna. El vector restante indica la fila a la que pertenece el dato en la posición respectiva. Esta forma de almacenar la matriz se le conoce como Columnas Dispersa Comprimida o por sus siglas en inglés Compressed Sparse Column (CSC)(«HB Files Harwell Boeing Sparse Matrix File Format», 2014).



```
matriz1                                                                 125
         10             1             2             5             2
RUA            6             6            14             6
(11i7)           (13i6)          (3D21.15)           (3D21.15)
F              1             0
      1      2      4      7     10     13
     1     2     3     3     4     5     3     4     5     4     5     6     5
     6
-.1000000000000000E+010.2000000000000000E+01-.1000000000000000E+01
-.1000000000000000E+010.2000000000000000E+01-.1000000000000000E+01
-.1000000000000000E+010.2000000000000000E+01-.1000000000000000E+01
-.1000000000000000E+010.2000000000000000E+01-.1000000000000000E+01
-.1000000000000000E+010.1000000000000000E+01
0.0000000000000000E+000.0000000000000000E+000.0000000000000000E+00
0.0000000000000000E+000.0000000000000000E+001.0000000000000000E+00
```

Gráfica 36 Archivo Harwell Boeing

La Gráfica 36 muestra la representación en Harwell Boeing de la matriz de la Gráfica 34. Las cinco primeras líneas son los encabezados que describen la estructura de la matriz y del mismo archivo. Los datos de la matriz están contenidos desde la línea

cinco hasta la 15. Para este caso en la línea seis se encuentran los índices que indican el inicio de la siguiente columna. Las filas seis, siete y ocho, indica la fila y la columna que pertenece el respectivo coeficiente. De la fila novena a la trece están propiamente los coeficientes. La solución al sistema está en las filas catorce y quince.

Para crear la matriz desde el archivo se tiene que observar primero los índices de inicio de cada columna. De esta manera la línea seis dice que la columna uno pertenece el primer elemento, la columna dos con el segundo elemento y el tercero, puesto que los elementos de la tercera columna inician con el índice 4. Siguiendo con la tercera columna estaría el dato correspondiente al índice 4, 5 y 6. La cuarta columna son los elementos que están en la posición 7 hasta la nueva. La quinta los de la posición 10 hasta la 12. La última columna, la sexta, se toma aquellos que están en la posición 13 y 14.

Los datos de los coeficientes en este archivo no se distinguen a simple vista, parece confundirse unos con el otro en la misma línea. Para distinguir se tiene que tomar los 21 primeros caracteres. Esto se define en el encabezado del archivo. Si se ve la línea cuatro, esta define el formato de los números. El primer formato (11i7) es para los indicadores de inicio de columna. Describe con la "i" que los números son enteros; el "17" que el número debe ocupar esa cantidad de espacios si no lo hace se rellena con espacios en blanco a la izquierda; el "11" indica que debe haber esa cantidad máxima de números por línea. El segundo formato (13i6) es para los índices de las filas, al igual que el anterior indica que deben ser números enteros; puede tener un máximo de "13" por línea; y deben ocupar "6" espacios o rellenarlos con vacíos. El tercer formato, el

“3D21.15” indica que los coeficientes se representan con decimales (“D”), un máximo de 3 números por fila y debe ocupar “21” espacios de los cuales “15” deben ser decimales.

Las tres primeras líneas del encabezado indican la estructura del archivo. La primera informa del nombre y código para identificar la matriz con la que se trabaja. La segunda línea informa cuantas y cuáles son las líneas que debe leer para construir la matriz. Así el primer número dice que hay diez líneas con la información para construir la matriz. El siguiente número indica el número de filas dedicadas a los indicadores de columnas, en este caso 1. El siguiente a este número dice que hay 2 filas para los índices de las filas. Luego que siguen 5 filas que son los datos de los coeficiente. Por último que hay dos filas más que corresponden a la solución del sistema.

La tercera fila del encabezado indica en este caso que la matriz es de 6 por 6. Los coeficientes no nulos son catorce; y cuenta con seis soluciones para X. La Sigla RUA indica que la matriz es real no simétrica ensamblada (Real Unsymmetric Assembled).
Se puede observar que la representación en Harwell Boeing es compleja, no porque usa CSC, sino porque se le tiene que indicar el número de filas y colocar la información en una columna adecuada en el archivo de texto (Ver el Anexo 7).

Archivo ELE

El archivo .ele es una representación de la matriz en el que los elementos de la matriz están divididos en submatrices. El tamaño de las submatrices es de NxN. Las submatrices hacen parte de la diagonal principal de la matriz completa. El archivo indica

el tamaño de cada submatriz. Con un vector de tamaño N se indica el índice de la correspondiente fila y columna. De esta forma para posición (i, j) de la submatriz corresponde en la matriz principal a la fila que indique el vector en su posición (i) y la columna al valor que indique el vector en la posición (j). Varias sub-matrices pueden compartir la misma coordenada dentro de la matriz general. En este caso sus valores se suman para obtener el coeficiente.

```
2
1 2
1 0
0 1
0 0
2
2 3
1 -1
-1 1
0 0
2
3 4
1 -1
-1 1
0 0
2
4 5
1 -1
-1 1
0 0
2
5 6
1 -1
-1 1
0 1

```

Gráfica 37 Archivo .ele

La matriz de la Gráfica 34 está representada en el formato .ele por la Gráfica 37 (Ver también Anexo 8). Como se mencionó anteriormente este archivo guarda submatrices de la diagonal principal. La primera fila indica que la su matriz es de 2x2. La siguiente fila son los índices. La siguientes dos filas son la matriz, se toma dos filas porque así lo indicó la primera fila. La fila posterior corresponde a la solución de la

submatriz. Los ceros que aparecen allí corresponden a las filas 1 y 2 de la matriz completa del Sistema, esto lo indica los índices de la línea 2.

Capítulo 8

Pruebas y Validación de Resultados.

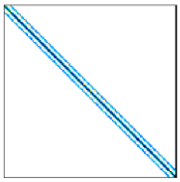
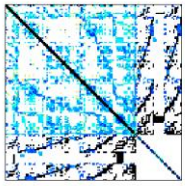
Las pruebas realizadas se llevaron a cabo en el clúster de la Universidad Tecnológica de Bolívar. La infraestructura de cuatro procesadores fueron usados para realizar la pruebas iniciales. CentoOS era el sistema operativo en el cual funcionaba el entorno. Posteriormente se usó el equipo de treinta dos núcleos. El sistema operativo fue cambiado a Ubuntu 14.0.4. En esta segunda fase se utilizó dos solvers. El método iterativo de gradiente conjugado era usado por uno de ellos; el otro usaba método directo en el cuál debía invertir la matriz para obtener la solución al sistema. Los datos usados se tomaron de la colección de matrices dispersas de la Universidad de Florida que se encuentran alojado en http://www.cise.ufl.edu/research/sparse/matrices por (Davis & Hu, 2011). De toda la colección se tomó un conjunto de aquellas matrices que tenían el mismo número de filas y columnas y su formato estaba en Harwell Boeing o Matrizx Market.

Sistema Operativo CentOS instalado sobre el equipo de cuatro procesadores

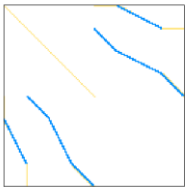
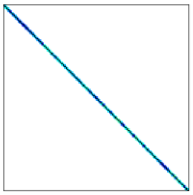
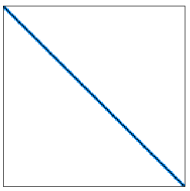
Se hizo unas pruebas con varias matrices con las que se variaba los número de procesadotres los resultados se obtenían en un archivo de texto. La primera prueba que

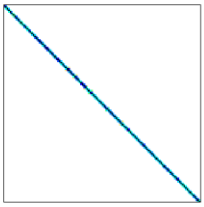
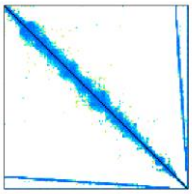
se realizó constaba de una matriz se 19716 incógnitas[1]y los resultados se obtuvieron en poco tiempo.

Para las primeras pruebas se usó Lis sobre un servidor CentOS release 6.5 con Intel(R) Xeon(R) CPU E5310@1.60GHz 4 GBRAM. Las pruebas se realizaron con diferentes tamaños de matrices. Los resultados se observan en las siguientes tablas.

	Incógnitas	No ceros	Ancho Banda	Radio Espectral
 matrix_9 Matriz de simulación de dispositivos semiconductores, Olaf Schenk, Universidad de Basel.	103,430	1,205,518	103,429	1,00
 Scircuit Matriz de simulación de semiconductors de Steve Hamm, Motorola, Inc.	170,998	958,936	341,951	29025,00

[1]La matriz fue tomada de ftp://math.nist.gov/pub/MatrixMarket2/SPARSKIT/fidap/fidap035.mtx.gz

 cont-300 Matriz de Karush–Kuhn–Tucker	180,895	988,195	157,497	8,10
 BenElechi1 Matriz dispersa generada por el grupo BenElechi	245,874	13,150,496	822	1956,00
 boyd2 Matriz generada por el grupo GHS_indef	466,316	1,500,397	373,056	3996339000,00
 af_0_k101 Problema de deformación de metal de la Universidad de Basel	503,625	17,550,675	860	2060893,00

 af_shell2 Problema de deformación de metal de la Universidad de Basel	504,855	17,562,051	4910	3336183,00
 thermal2 Problema térmico en estado estacionario De la Universidad de Oslo	1,228,045	8,580,313	1,226,001	7,81

Gráfica 38 Matrices Iniciales

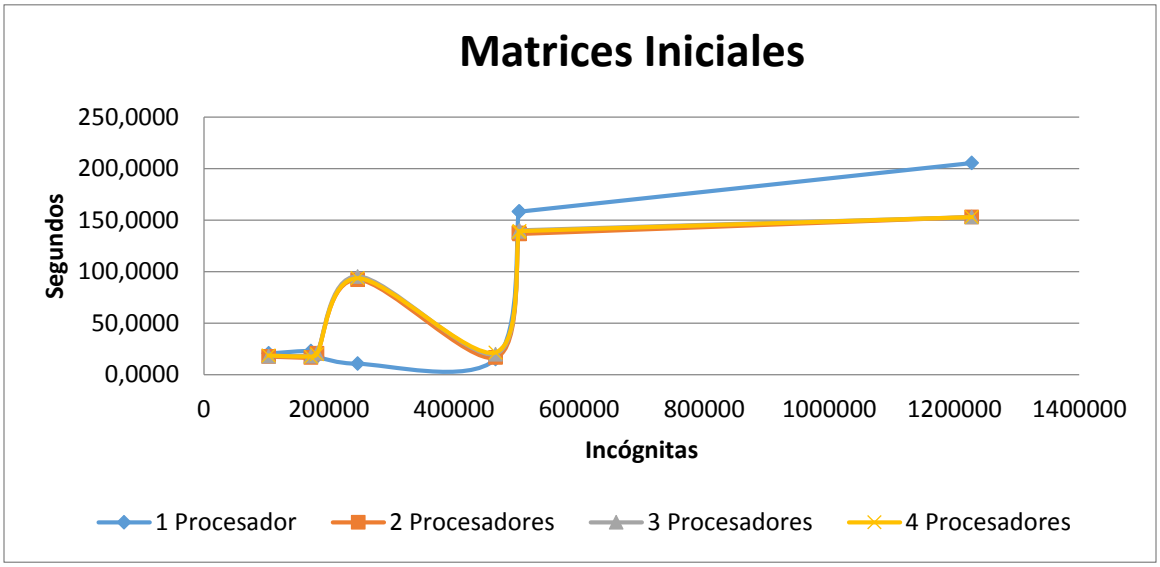
	Segundos			
Incógnitas	1 Procesador	2 Procesadores	3 Procesadores	4 Procesadores
103430	20,8004	17,6947	18,1381	18,6480
170998	23,1325	16,8377	18,2016	17,5715
180895	17,6096	20,8447	20,7316	20,0704
245874	10,7971	92,3446	95,4658	93,6379
466316	15,3072	16,8580	20,0675	21,6228
503625	158,2680	137,4905	138,9994	138,5050
504855	158,3170	136,6697	140,1080	139,3665
1228045	205,4853	153,0131	152,9620	152,9780

Gráfica 39 Prueba Lis con 4 procesadores

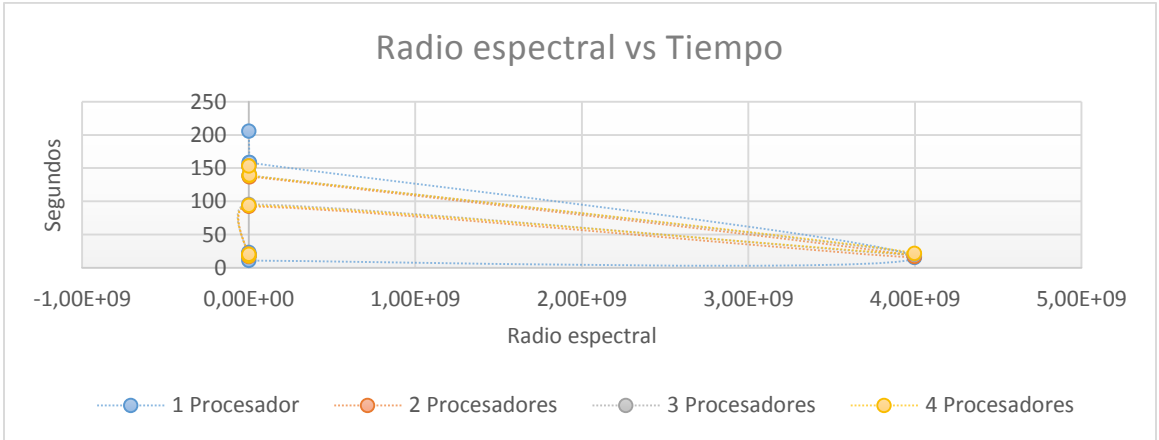
Al graficar las diferentes tablas se observa que en ciertos casos el rendimiento disminuye. Concretamente solo se aprovecha cuando se usa dos procesadores al usar más

no se nota mejoría. También se puede observar que las ventajas para un millón y medio de incógnitas es de solo unos cuantos segundos.

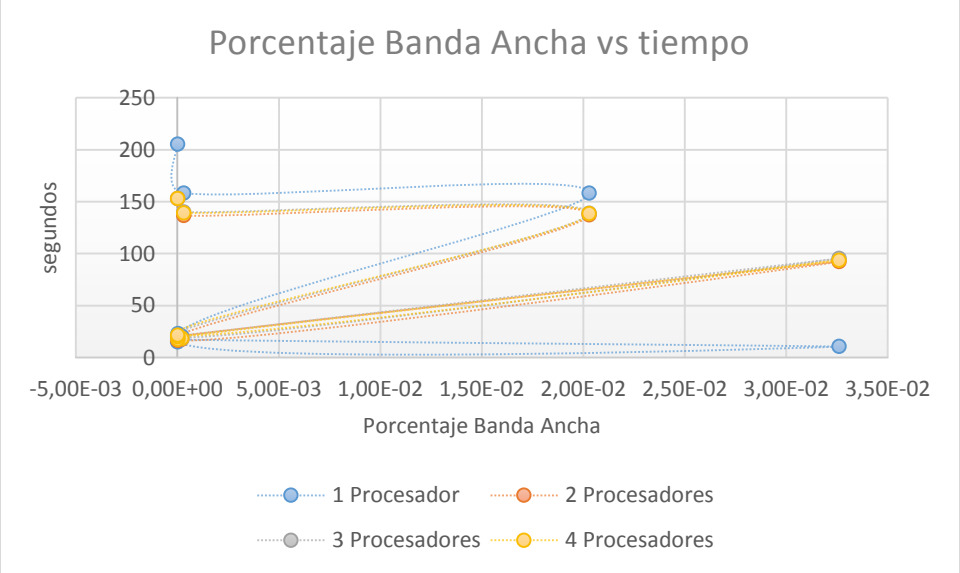
Con estas pruebas se podría prever que el uso de más procesadores puede influir mejor en problemas más complejos.



Gráfica 40 Matrices Iniciales vs tiempo



Gráfica 41 Radio espectral vs Tiempo de Matrices Iniciales

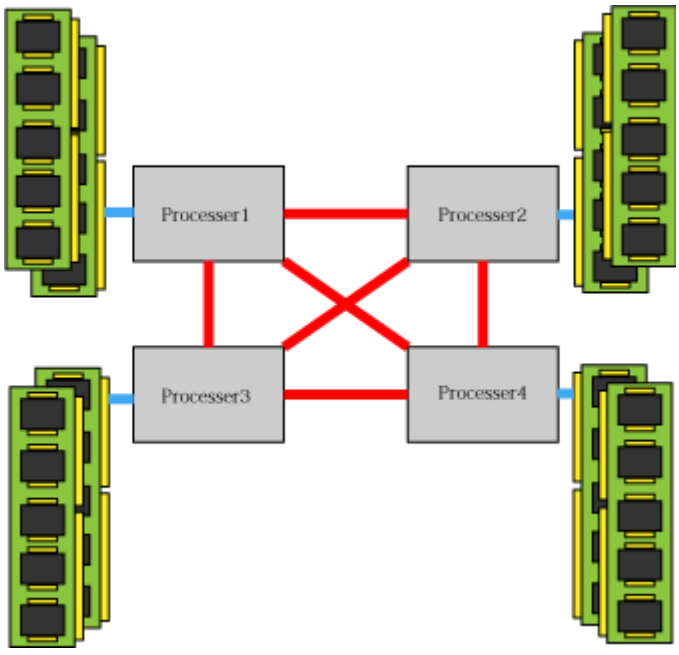


Gráfica 42 Porcentaje de Banda Ancha vs Tiempo de Ejecución en Matrices Iniciales

Estos resultados muestran que el aprovechamiento no solo depende de la cantidad de procesadores involucrados; la complejidad del sistema de ecuaciones y la infraestructura utilizada agregan variables para que un problema sea escalable dentro de un sistema de cómputo. En el experimento realizado las matrices no presentaban gran complejidad a pesar de sobrepasar el millón de incógnitas.

Ubuntu en el equipo de cómputo de 32 núcleos

Las siguientes pruebas se realizaron en el equipo de 32 procesadores, el cuál arquitectura consta de un diseño tipo NUMA (No-Uniform Memory Access). Allí un grupo de procesadores tiene acceso a un lote de memoria RAM para brindarles acceso rápido al recurso, contrario a una distribución de memoria centralizada.



Gráfica 43 Distribución de memoria en un diseño NUMA

En la siguiente fase se logró compilar la librería de álgebra lineal MUMPS. La compilación se tuvo que hacer usando la herramienta CMAKE. Se tuvieron que enlazar las dependencias y asignar banderas de compilación antes de tener la herramienta funcional. Construir la librería tomó un poco más de dos horas. La memoria de 257962 MB fue suficiente para que se alojaran las variables durante la etapa de compilación. En un equipo de escritorio, en el que se testearon las opciones de CMAKE, la compilación fallaba porque no podía reservar memoria. Se tuvo entonces que crear memoria SWAP.

A pesar de tener compilada la herramienta en CentOS en Ubuntu 14.0.4 las configuraciones y procedimientos variaron. SWIG, la herramienta que brinda una capa intermedia entre PYTHON y C++, debía ser compilado desde el código fuente. Ubuntu distribuía por medio de sus repositorios la versión 2.0.11.La distribución para compilar los paquetes no podía superar la 2.0.8. Distribuciones de subrutinas como BLAS y LAPACK, a diferencia de CentOS, tenían distribuciones en el gestor de paquetes de Ubuntu.

En esta prueba se tomaron matrices de hasta un millón quinientas mil incógnitas aproximadamente (Tabla 1).Estos tamaños fueron apropiados para evaluar el rendimiento de las librerías de álgebra lineal. Se ejecutaban en un tiempo prudente para observar su comportamiento. Se analizó el también el uso de la memoria RAM y que significaba la cantidad de información que alojaba en el proceso.

Matriz	Incógnitas	Ancho de banda	Radio Espectral

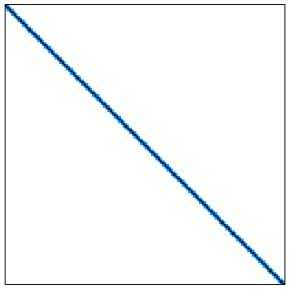
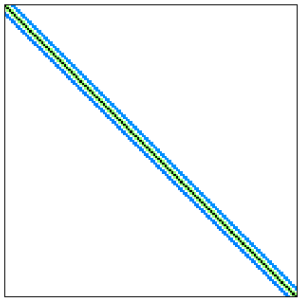
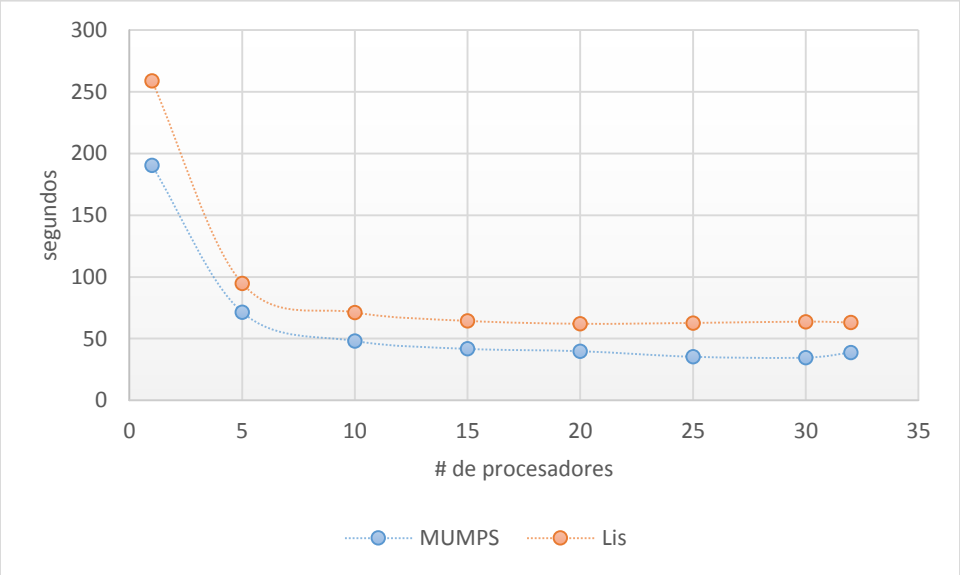
 af_0_k101 Problema de deformación de metal de la Universidad de Basel	503,625	860	620445,3
 Atmosmodl Model atmosférico de Andrei Bourchtein	1,489,752	78,409	2060893

Tabla 1 Matrices Utilizadas en el HPC

Matriz de quinientas mil incógnitas

Se hizo nuevamente los experimentos. Lis y MUMPS fueron probados esta vez con una matriz se más de quinientas mil variables. El desempeño tanto de Lis como para MUMPS mantuvo la tendencia de mejorar con el número de procesadores que se

ejecutaba simultáneamente. En esta ocasión MUMPS presentó mejor desempeño para resolver el sistema.

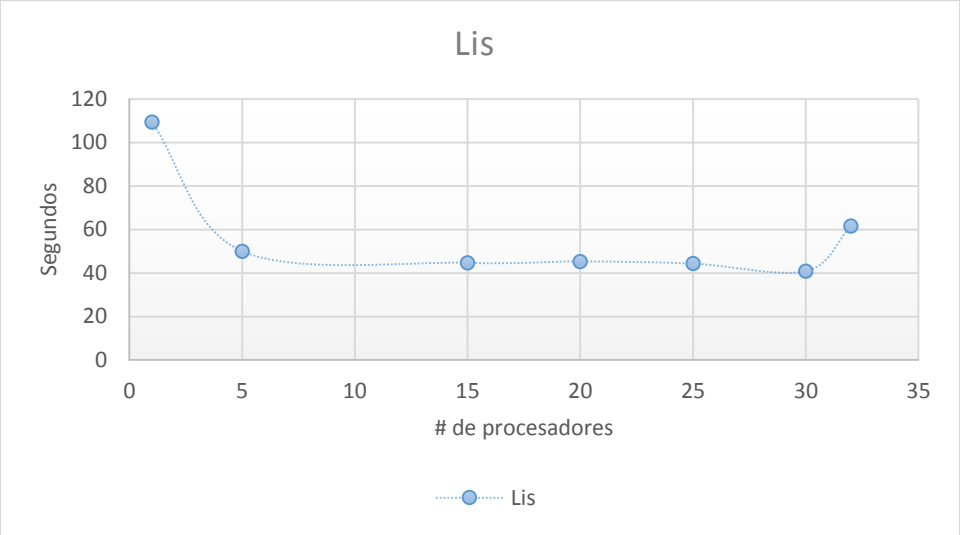


Gráfica 44 MUMPS vs Lis. 503625 de incógnitas

Matriz de más de un millón quinientas mil incógnita

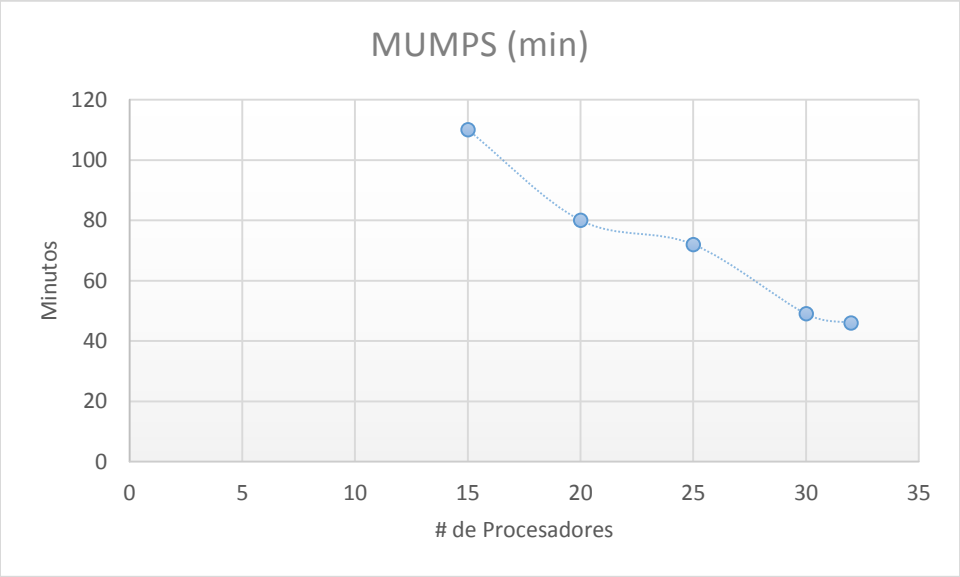
Al aumentar aún más la complejidad del problema el desempeño de las librerías que se utilizaron variaron drásticamente. La matriz de un millón cuatrocientos ochenta y nueve mil setecientos cincuenta y dos incógnitas hizo MUMPS empleara muchos más tiempo que su contraparte Lis. Principalmente porque MUMPS usa métodos directos para solucionar el sistema y Lis usa métodos iterativos. La ventaja de esto que la solución por el primer método es precisa mientras que la restante es aproximada.

En las siguientes gráficas se observa que Lis toma menos de dos minutos en resolver el problema lineal. Mumps por su parte toma más tiempo pero ese tiempo se beneficia al usar los 32 procesadores del equipo.



Gráfica 45 Lis con 1489752

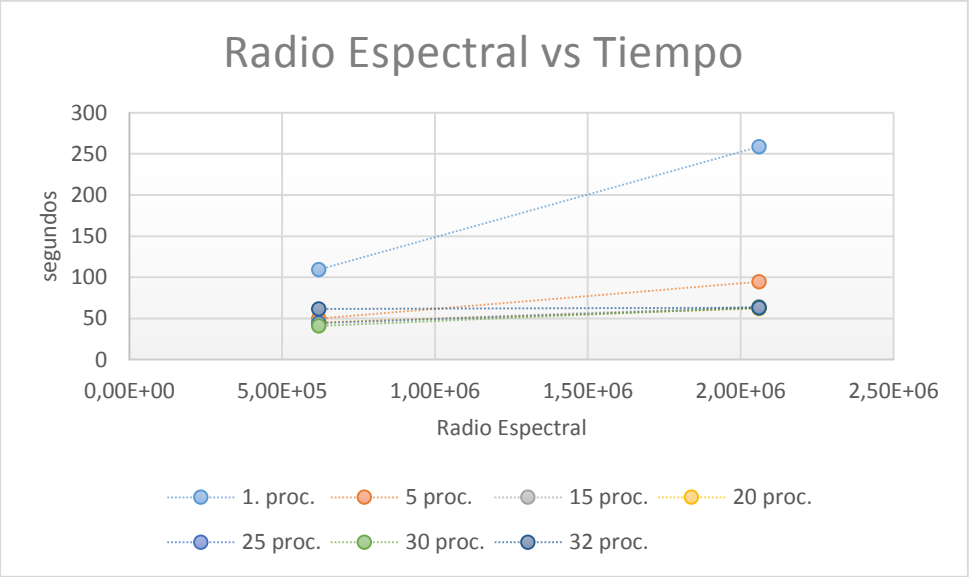
MUMPS efectivamente utiliza más tiempo en resolver el problema pero se compensa con la precisión y el número de procesadores. Tal como se observa usando los treinta y dos procesadores se puede resolver el problema en menos de una hora. Como contraparte un solo procesador trabajando en serie para resolver el problema por métodos directos tomaría todo el día de trabajo de un investigador.



Gráfica 46 MUMPS con 1489752 de incógnitas

Radio espectral con respecto a tiempo de ejecución

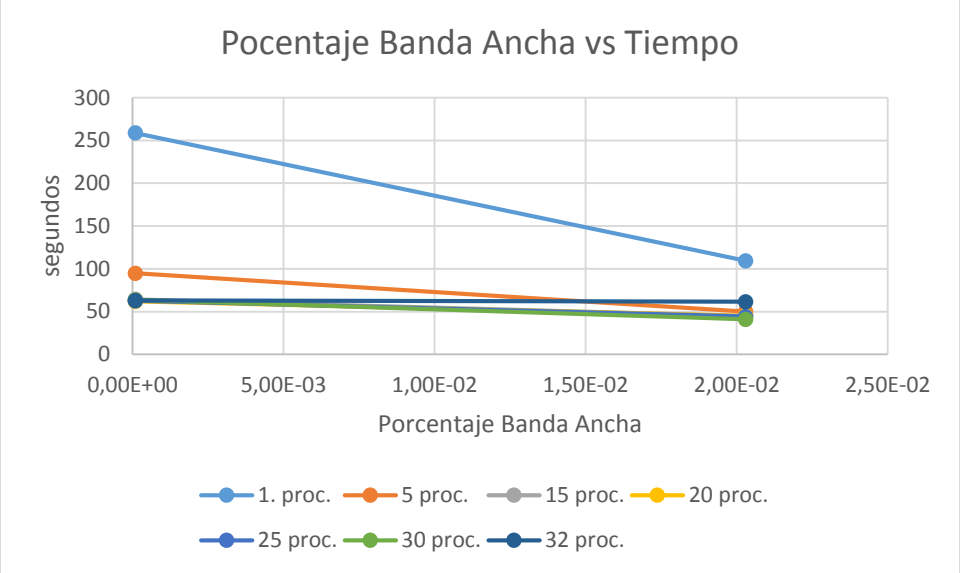
Al analizar el radio espectral de las matrices anteriores se obtuvo que en ambos casos para todos los procesadores reducían el tiempo de ejecución al tener un radio espectral menor.



Gráfica 47 Radio espectral vs Tiempo de ejecución de las segundas matrices

Porcentaje de banda ancha con respecto a tiempo de ejecución

En este aparatado se presenta el comportamiento de las pruebas realizadas en las matrices teniendo en cuenta la concentración de los datos dentro la región de confinación regida por la banda ancha. Se observa que si la concentración es mayor para todos los procesadores el tiempo de ejecución disminuye.



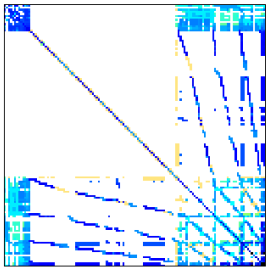
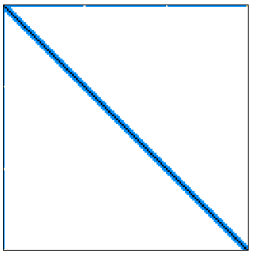
Gráfica 48 Porcentaje de Banda Ancha vs Tiempo de ejecución de las segundas matrices

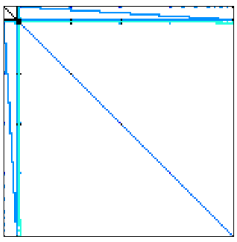
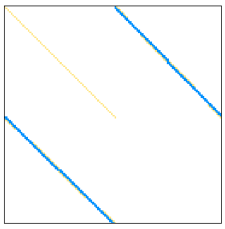
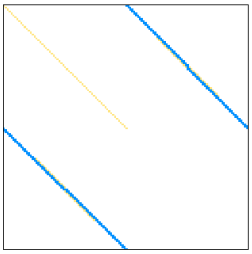
Prueba con matriz de hasta Dieciséis millones de incógnitas

Las siguientes pruebas son más exigentes, se usaron pruebas con matrices de 3 millones hasta diesciseis millones de incógnitas (ver Tabla 2). Mumps no pudo procesar este tipo de matrices, esto era de esperarse puesto que el método directo que utiliza para resolver el problema lineal termina consumiendo una cantidad exagerada de memoria; y consecuentemente Mumps termina la operación sin resolver la ecuación.Por su parte Lis pudo cargar el archivo de los problemas y resolverlos con un cargue en memoria mínimo.

En la Tabla 2 se muestra las matrices utilizadas en este experimento. Se muestra el número de incógnitas, el ancho de banda y el radio espectral. El radio espectral solo se

obtuvo para las dos primeras matrices debido a que para las demás la respuesta no convergía.

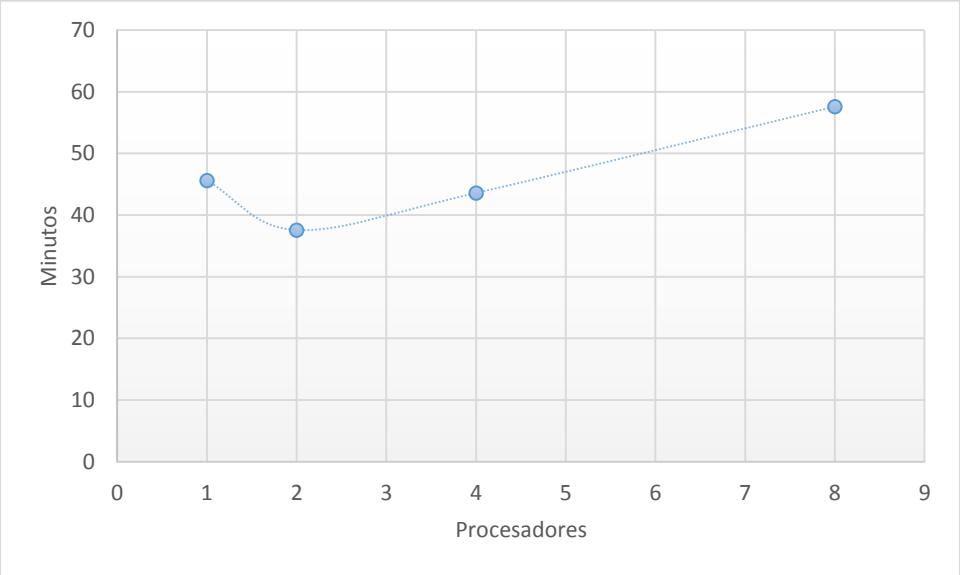
Matriz	Incógnitas	Ancho Banda	Radio Espectral
 Freescale1 Problema de circuito del grupo Freescale	3,428,755	6,850,083	620445,3
 rajat31 Matrices de simulación de circuitos de Rajat y Raj. De una empresa que desarrolla herramientas de simulación del	4,690,002	9,377,503	2060893

circuito comercial.			
 circuit5M circuito de gran tamaño de K. Gullapall del grupo Freescale	5,558,326	11,112,499	
 nlpkkt160 Matriz de Karush–Kuhn–Tucker Del grupo Schenk, Universidad de Basel	8,345,600	4,249,762	
 nlpkkt200 Matriz de Karush–Kuhn–Tucker	16,240,000	8,240,202	

Del grupo Schenk, Universidad de Basel			

Tabla 2 Matrices de hasta 16 millones de incógnitas aproximadamente

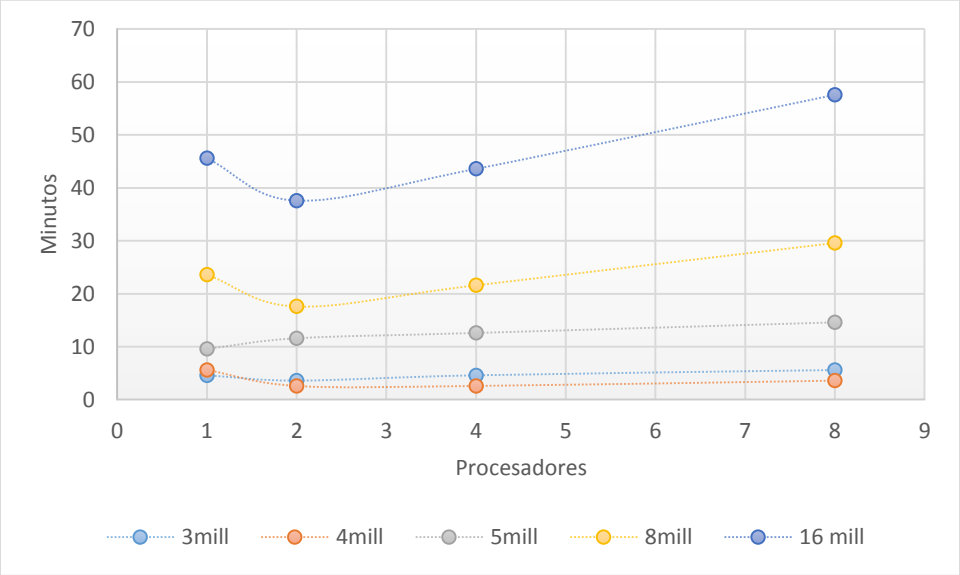
En las pruebas se observó que Lis escalaba el problema cuando se usaba dos procesadores. El mejor de los casos se presentó con la ecuación de más de 16 millones de incógnitas por ser un problema más complejo, se logró reducir en aproximadamente 8 minutos. Un tiempo aproximado de 45 minutos utilizó Lis para resolver la ecuación lineal sin paralelizar el proceso. Este tiempo requerido se elevaba al superar los dos procesadores por proceso.



Gráfica 49 Lis con 16 millones de incógnitas

Escalabilidad en matrices de tres, cuatro, cinco, ocho y dieciséis millones de incógnitas

En términos generales las pruebas hechas con esta complejidad de matrices mostró que Lis en este entorno de hardware solo mejoraba el tiempo para resolver estos sistemas complejos con un máximo de dos procesadores. Se notó que el tamaño de la matriz aumentaba directamente el tiempo requerido y la ganancia obtenida al usar dos procesadores igualmente aumentaba. Solo en el caso particular de la matriz de cinco millones de incógnitas no se presentó escalabilidad al aumentar los procesadores; por el contrario los resultados se obtenían con más tiempo de procesos. Este comportamiento demostró que no necesariamente el problema a resolver mejoraría al dividir las tareas en procesadores distintos; sin embargo si se logró un aumento en el rendimiento cuando en las demás matrices por lo menos con un par de procesadores.

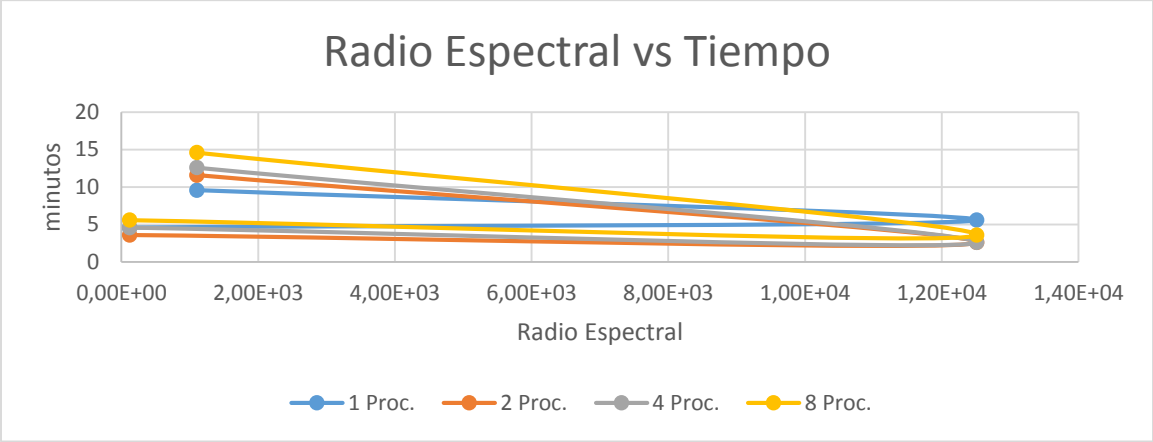


Gráfica 50 Lis con más de tres millones de incógnitas

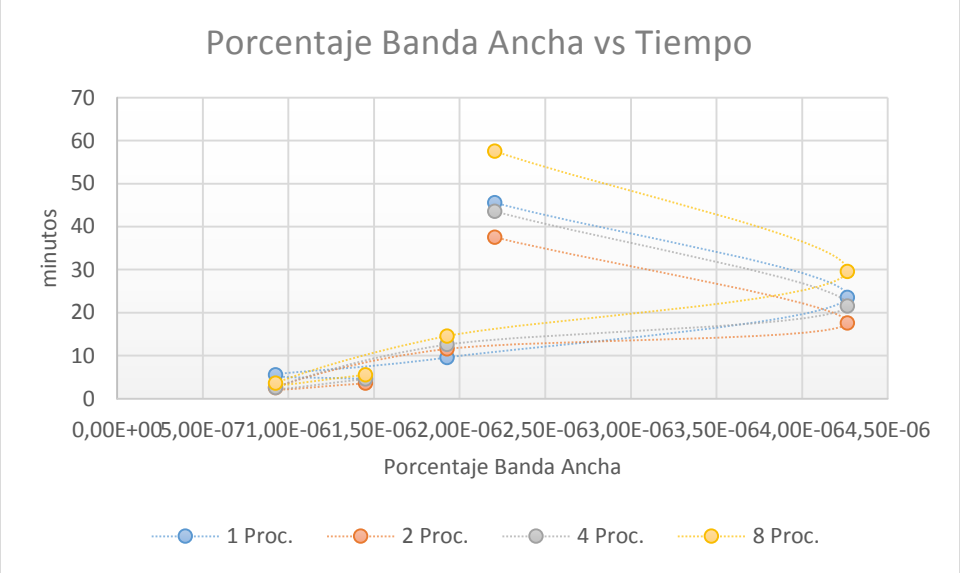
A diferencia de la pruebas iniciales que se tomaron problema de un máximo de millón quinientas incógnitas; el tiempo de proceso debió tomarse en minutos por y no en segundos. La solución de más incógnitas demandaba más tiempo en proceso sin embargo

se obtuvo que la escalabilidad no mejoraba más allá de dos procesadores. Esto puede atribuirse a que la complejidad no es lo suficientemente compleja para ser dividida en más de dos tareas.

En cuanto al radio espectral, al igual que las matrices anteriores el tiempo de ejecución aumenta al ser más grande el radio espectral. Siendo más favorable un radio espectral pequeño para converger más rápido a la respuesta.



Gráfica 51 Radio Espectral vs Tiempo en Matrices de hasta 16 millones de incógnitas



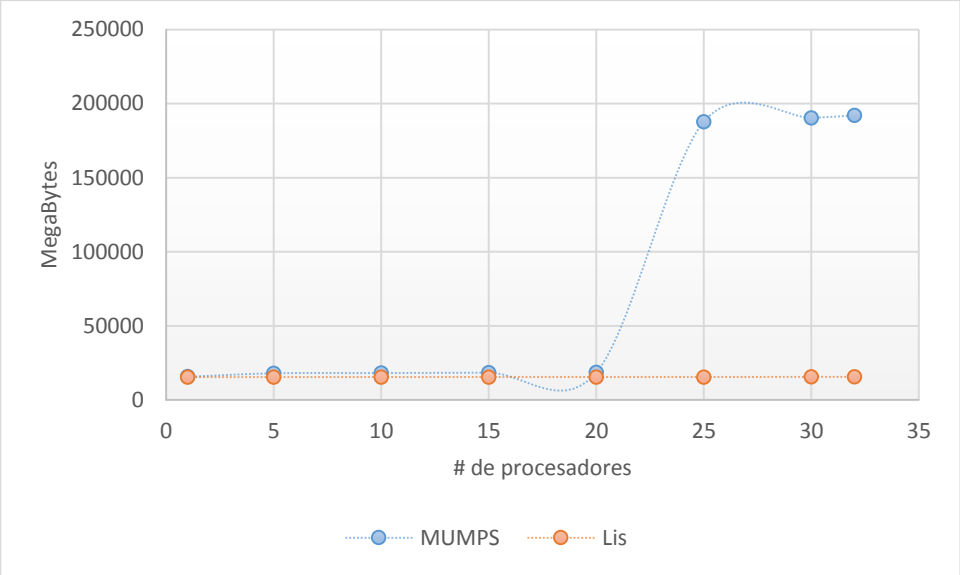
Gráfica 52 Radio Concentración en la Banda Ancha vs Tiempo en Matrices de hasta 16 millones de incógnitas

Uso de la memoria RAM

En las pruebas anteriores se evidenció que para matrices de más de un millón y medio de incógnitas los métodos directos para resolver el sistema de ecuaciones lineales se privilegian de la distribución de tareas entre los procesadores. Para un sistema de este tamaño puede tomar un tiempo considerable procesando por métodos directos y cómo se puedo evidenciar usando la infraestructura de la Universidad el mismo procedimiento se reduciría a tres cuartas partes de una hora.

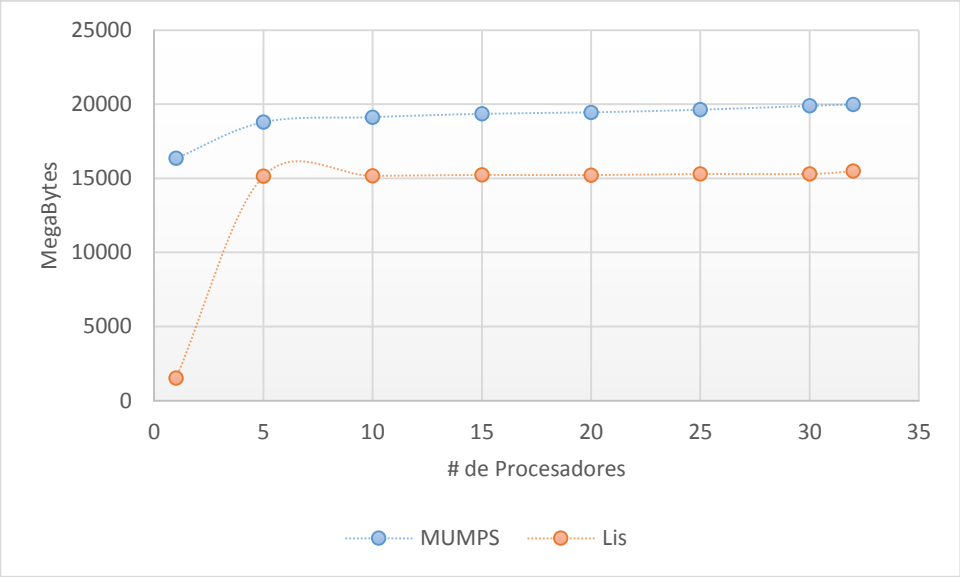
Para tener una ganancia en tiempo de procesamiento no solo las CPUS juegan en la mejora de desempeño de la actividad, también la memoria de acceso rápido. Estudiar el uso de la RAM es este tipo de procesos permite estar entendido si un equipo de cómputo es ideal para un tipo de problema específico.

En el anterior experimento se recogió la cantidad de memoria que utilizaba mientras realizaba el proceso de solución de las ecuaciones lineales. La matriz de dos cientos cincuenta mil incógnitas incrementó el uso de la memoria al usar más de veinte procesadores.



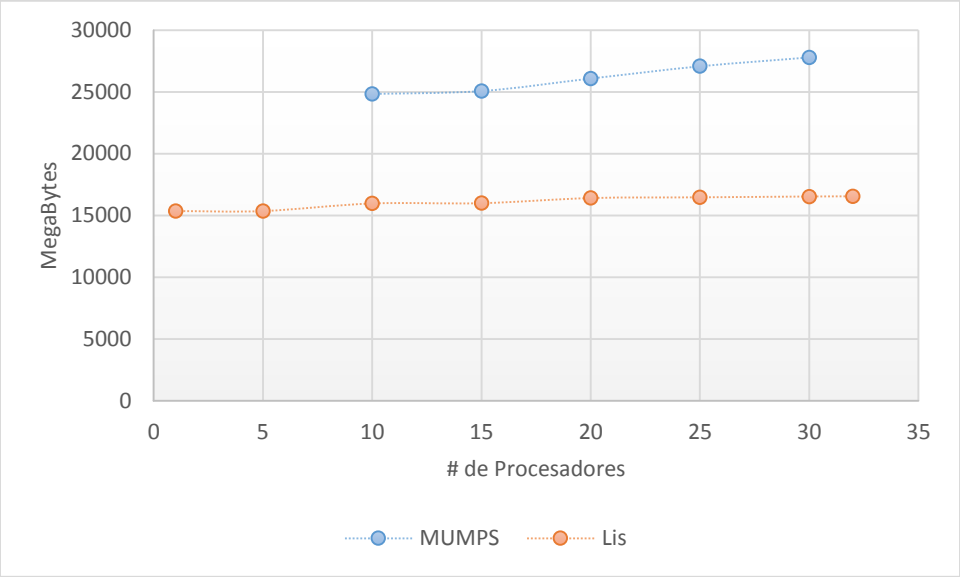
Gráfica 53 Uso de memoria con 255469

Se esperaría que con la matriz de 503625 incógnitas el uso de memoria se duplicara. No obstante, para el caso de MUMPS, el incrementó se evidenció con un número de procesadores inferiores a veinte. Lis consumió más RAM desde que usó cinco procesadores. MUMPS se mantuvo con más uso del recurso que Lis durante el uso de este experimento.



Gráfica 54 Uso de memoria con 503625 incógnitas

La diferencia en el consumo de Megabytes se evidenció más con la matriz de 1489752. El sistema registró su mayor registro de aproximadamente de dos mil setecientos Megabytes cuando usó todos los procesadores disponibles. Lis al igual que en las demás pruebas se mantuvo en alrededor de los mil quinientos.



Gráfica 55 Uso de memoria para 1489752 incógnitas

Comparación entre el método directo e Iterativo

En las pruebas realizadas con el método directo se usó Mumps y para el Iterativo Lis. Como se observa en las Gráfica 44 suele ser más efectivo en matrices pequeñas en menos de un millón de incógnitas. Usa más memoria pero la respuesta converge más rápidamente.

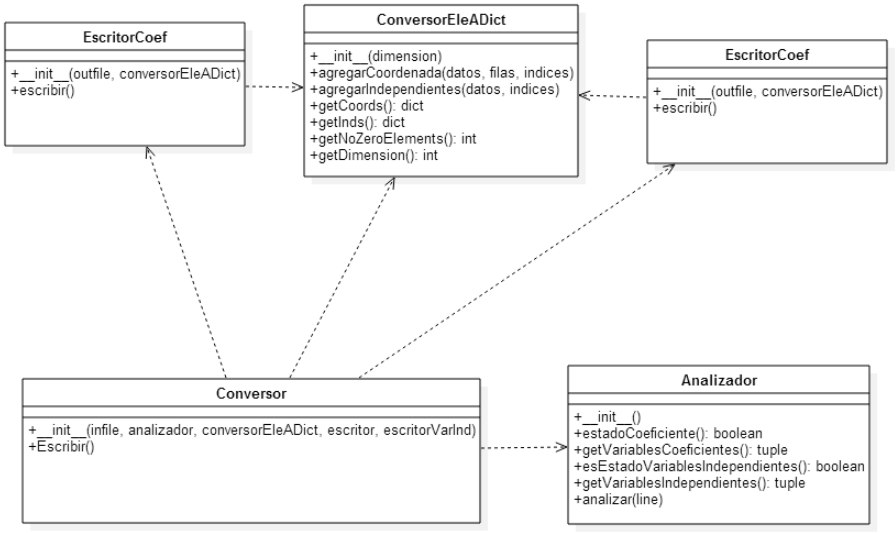
Para matrices más grandes Lis mostró mejor rendimiento (Gráfica 55). En este caso el método iterativo es mejor para matrices de más de un millón de incógnitas, pues usa menos memoria y converge más rápido. Con Mumps se notó que para las pruebas hechas solo funcionó con 15Procesadores (Gráfica 46) debido a que con menos procesadores reservaba más memoria al buffer para cada procesador tal como se explica en la sección Configuración Trilinos Mumps.

Resumiendo de acuerdo a las pruebas realizadas es de considerar usar el método directo para una Matriz no de más de un millón quinientos de incógnitas. Para matrices más grandes es mejor usar métodos iterativos.

Pruebas con una Matriz Generada Por la Universidad

Se procedió a realizar pruebas con una matriz generada a partir de triangulación a estructuras mecánicas. El problema estaba en que estaba guardado en formato .ele. Como ya se mencionó este formato guarda las submatrices de la diagonal principal, en un archivo de texto. Sin embargo el Solver necesitaba la matriz en formato Matriz Market. Para ello se tuvo que crear un script para generar el archivo de entrada requerido.

El script realizado en Python necesita una gran cantidad de memoria para poder generar el archivo. Realiza un recorrido por el archivo .ele. Identifica cuales son los índices, el tamaño de cada submatriz y sus soluciones. Guarda la ubicación de las coordenadas de los coeficientes y su valor en un diccionario. Una vez completa el diccionario crea el archivo en cuestión. LaGráfica 56 muestra el diagrama de clases en el que se muestra las dependencias entre los objetos. El código completo con sus métodos privados se puede ver en los Anexos 9, 10, 11, 12 y 13.



Gráfica 56 Diagrama de clase del programa que convierte archivos .ele a Matrix Market

El modelo proporcionado fue el de una estructura de torres. Se intentó convertir usando el script desarrollado, sobre un equipo de 1GB de memoria RAM. El programa se detuvo abruptamente por falta de memoria para crear el diccionario.

```
$ python main.py
Creado el coversor torresSBT10_0.ele
Leer Lineas

Segmentation fault
```

Gráfica 57 Error al convertir archivo .ele

Para mejorar este aspecto se desarrolló otra versión del convertidor que usaba que escribía el diccionario en un archivo. Sin embargo ese procedimiento no permitía convertir el archivo a una velocidad adecuada. En una prueba al finalizar 24 horas solo había recorrido y creado aproximadamente 2 mil entradas.

Por último se probó el convertidor en el clúster de la Universidad. El script que usaba diccionario pudo convertir sin dificultad el archivo. Tomó aproximadamente un minuto para hacer el trabajo ver (Gráfica 58).

```
$ time python main.py
Creado el coversor ../torresSBT10_0.ele
Leer Lineas
abrio archivo -...
encabezado

real    1m1.954s
user    1m0.746s
sys     0m1.060s
```

Gráfica 58 Conversión de archivo .ele

El archivo resultante se generó con una reducción de más del 50% en el tamaño (ver Gráfica 59). El formato Matrix Market solo guarda las coordenadas que no son ceros; mientras que además el archivo .ele guarda coordenadas repetidas debido a que hay matrices que comparten elementos de la misma posición de la matriz más grande. Estos se suman en el archivo final como parte del proceso de conversión.

```
$ ls -h -all new.mtx independientes1.txt ../torresSBT10_0.ele
-rwxrwx--- 1 jose jose 1,6M abr  7 21:41 independientes1.txt
-rwxrwx--- 1 jose jose 121M abr  7 21:41 new.mtx
-rw-r--r-- 1 jose jose 358M abr  2 21:15 ../torresSBT10_0.ele
```

Gráfica 59 Archivo .mtx resultante

Una vez terminada esa etapa se usó Lis para solucionar el Sistema con los archivos generados. Se probó con las configuraciones por defecto en Lis pero se notó que tomaba muy poco tiempo. Luego de analizar las respuestas del Solver se pudo cerciorar que terminaba el proceso porque el límite de las iteraciones era de 1000.

```
precon    : none
conv_cond : ||b-Ax||_2 <= 1.0e-12 * ||b-Ax_0||_2
storage   : CSR
lis_solve : LIS_MAXITER(code=4)

BiCG: number of iterations = 1001 (double = 1001, quad = 0)
BiCG: elapsed time         = 2.105165e+01 sec.
BiCG:   preconditioner     = 9.536743e-07 sec.
BiCG:     matrix creation  = 9.536743e-07 sec.
BiCG:   linear solver      = 2.105165e+01 sec.
BiCG: relative residual    = 9.235438e+01


real    0m28.152s
user    0m27.996s
sys     0m0.128s
```

Gráfica 60 Torres con 1000 iteraciones

Se tuvo que incrementar el número de iteraciones para obtener una mejor respuesta. Pero después de dos horas ejecutándose el sistema no encontraba una respuesta.

```
$ time ./test1 mf.mtx mfb.mtx solution.txt rhistory.txt -maxiter 727379968

number of processes = 1
matrix size = 129936 x 129936 (4694664 nonzero entries)
initial vector x = 0
precision : double
solver    : BiCG 2
precon    : none
conv_cond : ||b-Ax||_2 <= 1.0e-12 * ||b-Ax_0||_2
storage   : CSR
```

Gráfica 61 Lis con más de 700 millones de iteraciones

Por ello se redujo las iteraciones porque según lo visto el método iterativo no iba alcanzar una respuesta usando más iteraciones. Solo iba a agotar los intentos e iba a subutilizar el equipo de cómputo.

Se optó por utilizar 10 mil iteraciones para resolver el problema. Al probar con un solo proceso tomaba aproximadamente 30 minutos. Se probó con diferente número de procesadores y los resultados son los mostrados en la Gráfica 62.

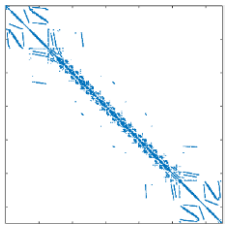
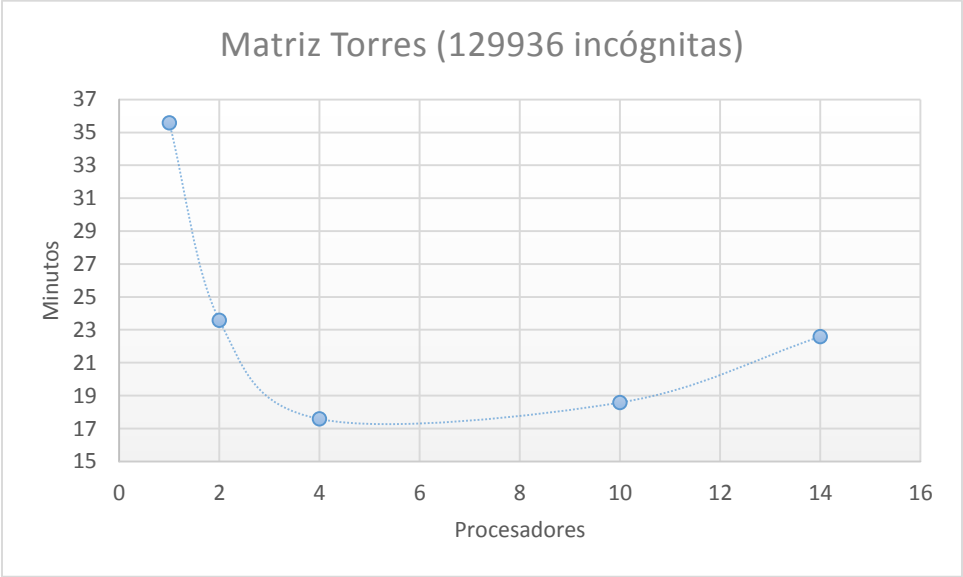
Descripción	No ceros	Incógnitas
 Torres	4,776,265	129,936

Tabla 3 Matriz de Torres



Gráfica 62 Tiempo proceso para matriz de Torres

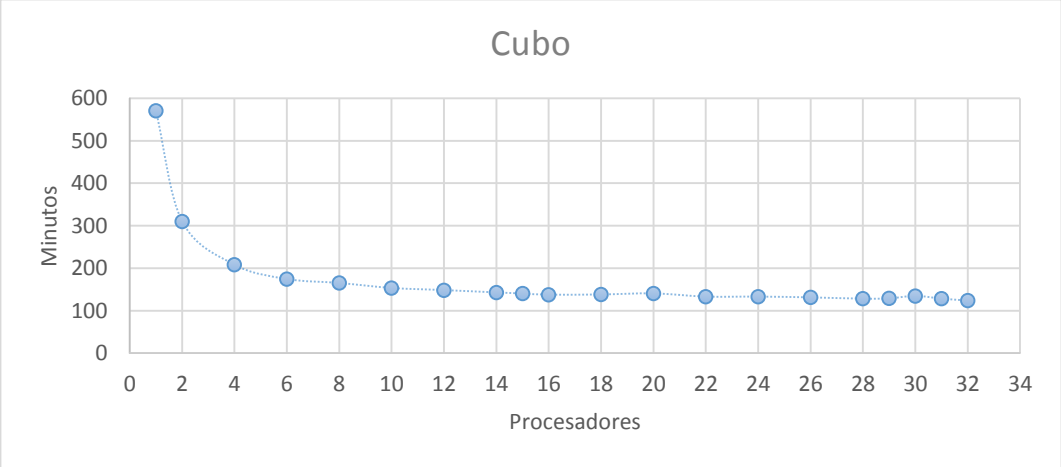
Se observó que hubo una reducción del tiempo de proceso con cuatro procesadores. Se mejoró el tiempo de respuesta en casi un 50 por ciento. Sin embargo hay varios detalles que se observaron. Uno de ellos es que en esta prueba el punto de inflexión era los cuatro procesadores. Al escalar más allá, el sistema gastaba más recursos para ser menos eficiente en el tiempo invertido para terminar su proceso. Por otro lado se observa que no todos los procesadores tienen pruebas en la gráfica. La razón por la que no están es porque Lis arrojaba un error y no terminaba de resolver el problema. Este error se presentaba solo cuando se utilizaba un número de procesadores concreto. Se consultó sobre este problema y por parte de los creadores Lis comunicaron que sucede cuando en su proceso iterativo realiza una división por cero que se presenta en el cociente de la direcciones de búsquedas(ρ) dentro del proceso del algoritmo del gadiente conjugado (ver Gráfica 3).

Con esta aclaración se puede concluir que no se puede solo usar una cantidad arbitraria de procesos para un sistema y esperar que el proceso termine satisfactoriamente. Para evitar esto es mejor usar pocas iteraciones para averiguar con qué cantidad de procesadores se puede solucionar el sistema.

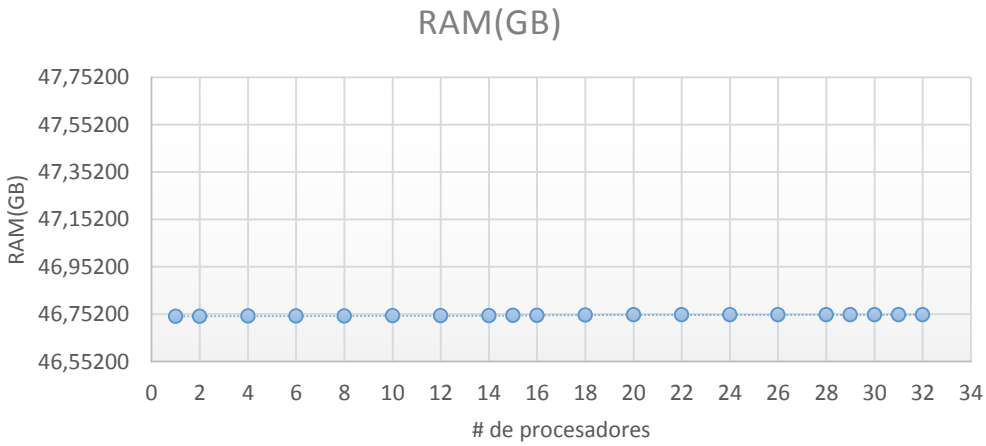
La siguiente Matriz proporcionada fue la de un cubo que se le aplica una fuerza. Esta matriz era extremadamente grande. El archivo .ele ocupaba 52 GB en disco. Por este motivo se presentaron inconvenientes tales como que nningún editor pudo cargar en memoria la matriz para mirar su contenido, el convertidor inicial no fue adecuado para trabajar con tanta información en memoria y se tuvo que reformar la estructura de datos.

Descripción	No ceros	Incógnitas
Cubo	1,934,549,999	10,328,853

Tabla 4 Martriz de Cubo



Gráfica 63 Matriz de cubo



Gráfica 64 Memoria usada al resolver la Matriz de Cubo

En este caso se logró utilizar eficientemente 16 procesadores, con más de estos el tiempo de proceso se mantenía estable alrededor de 120 minutos. La reducción en tiempo se mejoró cuatro veces. En este caso si no quisiera desperdiciar recursos se podría

ejecutar dos problemas similares al tiempo con 16 procesadores y con un consumo de RAM de 100GB de los 256GB disponibles por el computador de alto desempeño.

Capítulo 9

Análisis y Evaluación de Resultados.

A través de las diferentes pruebas realizas con las matrices usando El método directo implementado por Mumps y El iterativo de Lis se analizaron los resultados y los resultados se exponen a continuación.

En primer lugar luego de realizar las pruebas con el método directo e iterativo, se puede observar claramente que el método directo es más consistente para matrices no tan grandes. Específicamente en las pruebas realizadas se observa que para matrices menores a un millón quinientas mil. Esto se debe a la cantidad de memoria que necesita para funcionar, particularmente con Mumps esto se apreciar en que la librería falla al no reservar un tamaño adecuado para el Buffer de cada procesador. Es por esta razón que en la Gráfica 46 se observa que solo funciona con una cantidad mínimo quince procesadores. Por otra parte el método iterativo es ideal para trabajar con matrices sumamente grandes como se observó con una matriz de 16 millones de incógnitas. Sin embargo se observó que Lis puede fallar para una configuración arbitraria de procesadores debido a una división por cero por cero que se presenta en el cociente de la direcciones de búsquedas(ρ) dentro del proceso del algoritmo del gadiente conjugado (ver Gráfica 3).

El radio espectral en todas las pruebas se observó que al ser menor el tiempo requerido para dar solución a la ecuación disminuye y este comportamiento es idéntico al aumentar el número de procesadores. Este comportamiento es visible en la Gráfica 47, y

en las matrices con más incógnitas (Gráfica 51) se muestra que la convergencia es ligeramente más rápida con un radio espectral mayor con respecto al menor radio espectral registrado.

Con respecto al ancho de banda, en las evaluaciones se tomó el porcentaje de ocupación de los elementos no ceros dentro del confinamiento especificado por dicho ancho de banda. Los resultados mostraron para los primeros casos la disminución con el tiempo de proceso se obtenía con porcentajes bajos. Para las matrices más complejas en la que el tiempo de ejecución era mayor presentó que la convergencia a la solución se minimizaba cuando los elementos estaban más confinados en el ancho de banda. En las matrices más complejas probadas se obtuvo que el tiempo de ejecución menor se obtenía en un término medio con respecto al porcentaje de elementos dentro de la zona de banda. Por ende el porcentaje correspondiente a los elementos no ceros dentro de la banda ancha no influyen directamente en el tiempo de solución al usar un número de procesadores en concreto.

Un detalle a resaltar es el comportamiento asintótico (Gráfica 44, Gráfica 63), esto se debe a que a cierto número de procesadores el beneficio de dividir el trabajo en grupos pequeños se pierde. Sin embargo si se lograra usar más procesadores lograría un beneficio contrario (Gráfica 45, Gráfica 49). En esta última la utilización de más de dos procesadores excede el número máximo para obtener beneficios de cómputo para este problema. Volviendo al caso del Cubo se observa un comportamiento asintótico porque debido a la complejidad del problema se necesitaría muchos más procesadores para que

llegar un punto en que el trabajo sea ineficiente, es por esta razón que desde 16 a 32 procesadores consume prácticamente el mismo tiempo.

De acuerdo a las pruebas realizadas con las variables radio espectral y porcentaje del ancho de banda con respecto al tiempo y la longitud de la matriz se puede afirmar que la convergencia para un número de procesadores es menor cuando el radio espectral es bajo; sin embargo para el porcentaje de ancho de banda es un punto medio cuando las matrices son tan complejas que pueden tomar horas ejecutándose. Y respecto a la escalabilidad con el número de procesadores se obtiene beneficio si el problema es complejo, pero hay que tener en cuenta que existe un número máximo de procesadores en donde se obtiene la disminución en tiempo de ejecución para un problema particular, si se sigue agregando más problemas inicialmente presentará un comportamiento asintótico hasta que se eleven los tiempo de ejecución

Capítulo 10

Conclusiones y trabajos futuros.

Las librerias y procedimientos para resolver ecuaciones lineales son variadas. Principalmente se dividen en aquellos que usan métodos directos para resolver las incógnitas; y los iterativos que hacen uso de aproximaciones sucesivas. Se mostró que utilizar las librerias de código libre implica estar entendido con el ecosistema de linux como desempaquetar,empaquetar, compilar, enlazar librerias, entre otros. Especialmente se debe tener conocimiento de los distintos compiladores y de la utilidad CMAKE para construir los paquetes.

Todas estas librerías necesitan casi siempres las mismas dependencias. LIBBLAS y LAPACK son indispensables y se debe tener disponible la versión adecuada para proceder.Además de estar entendido de como compilarlo en caso de que el sistema operativo no poseea una distribución para el procesador del equipo.

De acuerdo a las observaciones de los experimentos se pudo comprender varios aspectos a a tener en cuenta cuando se decida usar librerias de algebra lineal con capacidad de procesamiento en paralelo. Aunque se obtuvo problemas de aproximandamente de un cuarto de millón de incógnitas el uso de método directo o iterativo no genera una gran diferencias; sin embargo se debe usar el métodos directo puesto que este obtiene la respuesta exacta al sistema y no usando métodos de aproximación. En la prueba más extrema que se implementó se pudo observar que el método directo requería mucho tiempo en comparación al iterativo, si en embargo gracias

a que se utilizaron treinta y dos procesadores y sufiente memoria RAM el método directo pudo resolver el sistema en menos de una hora.

Para matrices con mayor tamaño, como son de tres millon y más incógnitas, Mumps resultaba inapropiado pues el métodos directo requería más desplegar un tamaño no apropiado del vector de la matriz en los buffers de los procesadores y por consiguiente producía error la misma aplicación sin dar la solución al sistema.. Por el contrario Lis pudo manejar los procesos y generar los resultados sin usar demasiada memoria de accesos rápido a pesar que la matriz de dieciseis millones ocupaba aproximadamente cuatro Gigabytes y medio de espacio en el disco duro. Por estas evidencias se recomienda usar Lis para resolver matrices con más de tres millones de incógnitas. Así se evitará el uso excesivo de memoria y se obtendrían los resultados de la ecuación. Debido a que los métodos iterativos obtienen una solución aproximada basado en una tolerancia se debe usar el método directo que usa Mumps para matrices de menores de millón quinientos de incógnitas.

En las pruebas con la matriz proporcionada por la Universidad se mostró que a pesar de no ser tan grande la solución era compleja. Lis tomaba mas de treinta minutos en llegar a una solución aproximada. Y al escalarlo a cuatro procesadores reducía el tiempo a más de la mitad. Sin embargo cierto numero de procesadores generaba error y el programa terminaba el procesos sin acercarse a las soluciones debido a que en la ejecución del algoritmo necesita hacer una división en el cual su cociente no puede ser cero. Con la prueba del cubo el tiempo se puso reducir cuatro veces con 16 procesadores

Con las evidencias expuestas anteriormente se puede dar respuesta a los objetivos del trabajo. Se van exponer las respuestas en orden y los criterios que se tomaron.

Las pruebas indicaron que al aumentar el número de procesadores si se mira con una matriz de gran tamaño puede reducir el tiempo de ejecución, sin embargo se observó que la matriz de dieciséis millones de incógnitas la escalabilidad es solo perceptible con un par de procesadores. En la prueba con la matriz de torres la escalabilidad se vió con cuatro procesadores. Para ambos casos el tiempo computacional empeoraba al usar más de los procesadores mencionados. Por tanto la escalabilidad depende de la complejidad del la matriz. Se mostró además que el tiempo de ejecución para cualquiere número de procesadores es más rápido si el radio espectral es más pequeño y que el porcentaje de elementos en el ancho de banda no influye con exactitud en el tiempo puesto que en los experimentos se mostro que para las matrices menos complejas se solucionaban más rápido con una menor contingencia en la banda ancha, pero en matrices más complejas se solucionaban más rápido con una banda acha a término medio.Se mostró que calcular el número de condición para las matrices usadas en el experimento no era posible porque se necesita reservar más de 200 GB de memoria y otros casos más de mil GB.

La viabilidad de la infraestructura HPC de la Universidad para resolver matrices con más de 10 millones de incógnitas es favorables. Como se puede ver en las pruebas el sistema fue capas de cargar en memoria hasta dieciseis millones. El tiempo de solución fue mejorado al dividir el trabajo con un par de procesadores. Y se espera que problemas más complejos en un futuro puedan aprovechar mejor las características del clúster. También que para aprovechar la librerías se debe primero convertir los datos al formato

que lo requiera. La conversión de matrices con millones de incógnitas está limitado por la cantidad de memoria que debe usar algún script. Sin embargo la infraestructura es capas de procesar la conversión sin problema con las prestaciones del sistema actual tal como se pudo observar con la matriz de cubo.

Para el último objetivo se cocluyó que las herramientas de software libre disponibles para la solución de estos problemas pueden ser aprovechadas si se tiene conocimiento de linux. Aunque hay multitudes de librerías de algebra lineal, si se toma en consideración las que se usaron en este trabajo Lis y Mumps resultaron herramientas adecuadas para hacer el trabajo. No se debe probar la matriz con todos los procesadores disponibles sin antes ver su comportamiento con un par de ellos, de contrario se podría estar desperdiciando recuerso y tiempo como se pudo ver con las prueba de la matriz de 16 millones de incógnitas, y del mismo modo ajustar una tolerancia adecuada. El formato de entrada de éstas matrices es fundamental. Si no se cuenta con ello las implicaciones de convertir matrices extremadamente grandes dependerá en principal medida de la cantidad de memoria del sistema, puesto como se evidenció en las pruebas que la conversión de un archivo .ele tomó más de un minuto y se pudo solo ejecutar en el clúster. El único inconveniente visto en la herramientas Open Source radica en la documentación desde la configuración, compilación y por último en la implementación, como es el caso de SuperLu. Se espera que se pueda configurar exitosamente esta última herramienta en un futuro.

La hipótesis planteada en el trabajo desde el principio planteaba que al usar un número mayor de procesadores para resolver un sistema de ecuaciones lineales de gran

tamaño debía minimizar el tiempo empleado, pero como se evidenció esto no siempres es el caso. La complejidad de la matriz influye en la escalabilidad del problema.

La pregunta de investigación se respondería recordando que en las pruebas realizadas el HPC de la Universidad pudo resolver un sistema de 16 millones de incógnitas sin ninguna dificultad pero solo los pudo escalar exitosamente a dos procesadores. Y en otro caso se pudo escalar a cuatro procesadores con una matriz con menos de un millón de incógnitas. Así que aprovechar la disponibilidad de los 32 procesdores depende en medida también a la complejidad del problema.

En definitiva la herramienta de solución directa que se utilizó resulta muy adecuada para resolver ecuaciones lineales con las infraestructura de computación de alto desempeño con la que cuenta la Universad. Sin embargo no fué útil con matrices superiores a tres millones de incógnitas.Para éstas matriceses recomendable utilizar los métodos iterativos. Para ello se probó con Lis y se observó que pudo resolver problemas con más de tres millones de incógnitas sin embargo su escalabilidad en paralelo solo fué productivo con dos procesadores.Por otro lado al probar con un problema generado por la universidad la escalabilidad fue evidente. Pudo restar a más del 50% el tiempo de respuesta. Deacuerdo a esta observación el tiempo de respuesta debe ser mejor con problemas más complejos que use la Universidad.

Se espera que estas conclusiones sean de ayuda para ver más posibilidades de hacer un mejor uso de la infraestructura con la que se cuenta. Se podría continuar probando otras librerías que no se pudieron configurar exitosamente o probarlos en otros sistemas de computo de al to desempeño, en los que estas librerias fueron diseñadas,

como Cray XT3, Cray XT4 o ALTIX. Quizá los inconvenientes que se presentaron no se repitan en esas arquitecturas de supercomputadores, teniendo en cuenta que la documentación de una de los software que se usaron estaba pensado para esta arquitecturas específicas.

Referencias

Allaire, G., & Kaber, S. M. (2008). *Numerical Linear Algebra*. Springer Science & Business Media.

Amestoy, P. R., Duff, I. S., & L'Excellent, J.-Y. (2000). Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, *184*(2–4), 501-520. http://doi.org/10.1016/S0045-7825(99)00242-X

Balis, B., Figiela, K., Malawski, M., & Jopek, K. (2015). Leveraging Workflows and Clouds for a Multi-frontal Solver for Finite Element Meshes. *Procedia Computer Science*, *51*, 944-953. http://doi.org/10.1016/j.procs.2015.05.230

Bhujade, M. R. (2009). *Parallel Computing*. New Age International.

Boisvert, R. F., Pozo, R., Remington, K. A., Barrett, R. F., & Dongarra, J. (1996). Matrix Market: a web resource for test matrix collections. En *Quality of Numerical Software* (pp. 125–137). Recuperado a partir de ftp://ftp.idsa.prd.fr/pub/mirrors/netlib/utk/people/JackDongarra/pdf/matrixmarket.pdf

Bouwmeester, H., Dougherty, A., & Knyazev, A. V. (2015). Nonsymmetric Preconditioning for Conjugate Gradient and Steepest Descent Methods1. *Procedia Computer Science*, *51*, 276-285. http://doi.org/10.1016/j.procs.2015.05.241

Caldwell, J., & Zisserman, A. (1984). Program F. MATINV for the solution of the sparse linear system Ax = b including normalisation. *Advances in Engineering Software (1978)*, *6*(3), 168-170. http://doi.org/10.1016/0141-1195(84)90029-9

Calo, V. M., Collier, N. O., Pardo, D., & Paszyński, M. R. (2011). Computational complexity and memory usage for multi-frontal direct solvers used in p finite element analysis. *Procedia Computer Science*, *4*, 1854-1861. http://doi.org/10.1016/j.procs.2011.04.201

Carballo, F. P., Delgado, J. C., Canteli, A. F., & Rey, M. J. L. (2006). *Cálculo matricial de estructuras*. Universidad de Oviedo.

CECAD. (s. f.). Recuperado a partir de http://cecad.udistrital.edu.co/index.php/quienes-somos/proyectos-realizados

Chang, L.-W., Stratton, J. A., Kim, H.-S., & mei W. Hwu, W. (2012). A scalable, numerically stable, high-performance tridiagonal solver using GPUs. En J. K. Hollingsworth (Ed.), *SC* (p. 27). IEEE/ACM. Recuperado a partir de http://dblp.uni-trier.de/db/conf/sc/sc2012.html#ChangSKH12

Couturier, R., & Jezequel, F. (2010). Solving large sparse linear systems in a grid environment using Java. En *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)* (pp. 1-7). http://doi.org/10.1109/IPDPSW.2010.5470713

Davis, T. A. (2006). *Direct Methods for Sparse Linear Systems*. SIAM.

Davis, T. A., & Hu, Y. (2011). The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, *38*(1), 1:1–1:25. http://doi.org/10.1145/2049662.2049663

Di Felice, P., & Clementini, E. (1987). A pascal program for the solution of sparse linear systems – Part I. *Advances in Engineering Software (1978)*, *9*(2), 84-89. http://doi.org/10.1016/0141-1195(87)90029-5

Dongarra, J. J., & Eijkhout, V. (2000). Numerical linear algebra algorithms and software. *Journal of Computational and Applied Mathematics*, *123*(1–2), 489-514. http://doi.org/10.1016/S0377-0427(00)00400-3

Ercan, M. F., Fung, Y., Ho, T., & Cheung, W. (2003). Parallel Linear System Solution and Its Application to Railway Power Network Simulation. En H. Kosch, L. Böszörményi, & H. Hellwagner (Eds.), *Euro-Par 2003 Parallel Processing* (Vol. 2790, pp. 537-540). Berlin, Heidelberg: Springer Berlin Heidelberg. Recuperado a partir de http://bibliotecadigital.usbcali.edu.co:2136/chapter/10.1007/978-3-540-45209-6_77

Falgout, R. D., Jones, J. E., & Yang, U. M. (2006). Conceptual interfaces in hypre. *Future Generation Computer Systems*, *22*(1–2), 239-251. http://doi.org/10.1016/j.future.2003.09.006

Flynn, M. (1966). Very high-speed computing systems. *Proceedings of the IEEE*, *54*(12), 1901-1909. http://doi.org/10.1109/PROC.1966.5273

Flynn, M. (2007). Computer Architecture. En *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc. Recuperado a partir de http://onlinelibrary.wiley.com/doi/10.1002/9780470050118.ecse071/abstract

Gao, J., Liang, R., & Wang, J. (2014). Research on the conjugate gradient algorithm with a modified incomplete Cholesky preconditioner on GPU. *Journal of Parallel and*

Distributed Computing, *74*(2), 2088-2098. http://doi.org/10.1016/j.jpdc.2013.10.002

Gropp, W., Lusk, E., & Skjellum, A. (2014). *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press.

Guo, X., Gorman, G., Lange, M., Mitchell, L., & Weiland, M. (2013). Exploring the Thread-level Parallelisms for the Next Generation Geophysical Fluid Modelling Framework Fluidity-ICOM. *Procedia Engineering*, *61*, 251-257. http://doi.org/10.1016/j.proeng.2013.08.012

Hayder, M. E., Keyes, D. E., & Mehrotra, P. (1998). A comparison of PETSc library and HPF implementations of an archetypal PDS computation. *Advances in Engineering Software*, *29*(3–6), 415-423. http://doi.org/10.1016/S0965-9978(97)00080-X

HB Files Harwell Boeing Sparse Matrix File Format. (2014, octubre 14). Recuperado a partir de http://people.sc.fsu.edu/~jburkardt/data/hb/hb.html

http://www.eafit.edu.co/. (s. f.). Recuperado a partir de http://www.eafit.edu.co/investigacion/revistacientifica/Paginas/supercomputador-apolo.aspx

Johnson, C., & Mathematics. (2009). *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Mineola, N.Y: Dover Publications.

Karniadakis, G., & II, R. M. K. (2003). *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation*. Cambridge University Press.

Kincaid, D. R., & Young, D. M. (1988). A brief review of the ITPACK project. *Journal of Computational and Applied Mathematics*, *24*(1–2), 121-127. http://doi.org/10.1016/0377-0427(88)90347-0

Knibbe, H., Oosterlee, C. W., & Vuik, C. (2011). GPU implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method. *Journal of Computational and Applied Mathematics*, *236*(3), 281-293. http://doi.org/10.1016/j.cam.2011.07.021

Kolman, B., & Hill, D. R. (2006). *Álgebra lineal*. Pearson Educación.

Kotakemori, H., Hasegawa, H., Kajiyama, T., Nukada, A., Suda, R., & Nishida, A. (2008). Performance Evaluation of Parallel Sparse Matrix–Vector Products on SGI Altix3700. En M. S. Mueller, B. M. Chapman, B. R. de Supinski, A. D. Malony, & M. Voss (Eds.), *OpenMP Shared Memory Parallel Programming* (Vol. 4315, pp. 153-163). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kuźnik, K., Paszyński, M., & Calo, V. (2013). Grammar-Based Multi-Frontal Solver for One Dimensional Isogeometric Analysis with Multiple Right-Hand-Sides. *Procedia Computer Science*, *18*, 1574-1583. http://doi.org/10.1016/j.procs.2013.05.325

Laboratorio de Super Cómputo y Visualización en Paralelo. (s. f.). Recuperado a partir de http://supercomputo.izt.uam.mx/?q=historia-lsvp

Laboratorio Super Computo Medellín. (s. f.). Recuperado a partir de http://www.elmundo.com/portal/vida/tecnologia/nuevo_laboratorio_de_simulacio

n_y_computacion_cientifica_en_la_universidad_de_medellin.php#.VZgAN_l_Oko

Lang, J., & Rünger, G. (2014). An execution time and energy model for an energy-aware execution of a conjugate gradient method with CPU/GPU collaboration. *Journal of Parallel and Distributed Computing, 74*(9), 2884-2897. http://doi.org/10.1016/j.jpdc.2014.06.001

Li, X., & Li, F. (2014). GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method. *Electric Power Systems Research, 116*, 87-93. http://doi.org/10.1016/j.epsr.2014.05.005

Li, X. S. (2005). An Overview of SuperLU: Algorithms, Implementation, and User Interface. *toms, 31*(3), 302-325.

Li, X. S., Demmel, J. W., Gilbert, J. R., Grigori, iL, Shao, M., & Yamazaki, I. (1999). *SuperLU Users' Guide* (No. LBNL-44289). Lawrence Berkeley National Laboratory.

Logan, D. L. (2011). *A First Course in the Finite Element Method* (5 edition). Stamford, CT: Cengage Learning.

Mallya, J. U., Zitney, S. E., Choudhary, S., & Stadtherr, M. A. (1999). Matrix reordering effects on a parallel frontal solver for large scale process simulation. *Computers & Chemical Engineering*, *23*(4–5), 585-593. http://doi.org/10.1016/S0098-1354(98)00295-6

Manguoglu, M. (2011). A domain-decomposing parallel sparse linear system solver. *Journal of Computational and Applied Mathematics, 236*(3), 319-325. http://doi.org/10.1016/j.cam.2011.07.017

Matrix Market Exchange. (s. f.). Recuperado 14 de octubre de 2014, a partir de http://math.nist.gov/MatrixMarket/formats.html#MMformat

Michailidis, P. D., & Margaritis, K. G. (2011). Parallel direct methods for solving the system of linear equations with pipelining on a multicore using OpenMP. *Journal of Computational and Applied Mathematics, 236*(3), 326-341. http://doi.org/10.1016/j.cam.2011.07.023

Miztli. (s. f.). Recuperado a partir de http://www.super.unam.mx/index.php/content-layouts

Nishida, A. (2010). Experience in Developing an Open Source Scalable Software Infrastructure in Japan. En D. Taniar, O. Gervasi, B. Murgante, E. Pardede, & B. O. Apduhan (Eds.), *Computational Science and Its Applications – ICCSA 2010* (Vol. 6017, pp. 448-462). Berlin, Heidelberg: Springer Berlin Heidelberg.

O'Connor, J. L. de la F. (1997). *Técnicas de cálculo para sistemas de ecuaciones, programación lineal y programación entera: códigos en FORTRAN y C con aplicaciones de sistemas de energía eléctrica*. Reverte.

Pacheco, P. S. (1997). *Parallel Programming with MPI*. Morgan Kaufmann.

Paszyński, M., Pardo, D., & Paszyńska, A. (2010a). Parallel multi-frontal solver for multi-physics adaptive problems. *Procedia Computer Science, 1*(1), 1983-1992. http://doi.org/10.1016/j.procs.2010.04.222

Paszyński, M., Pardo, D., & Paszyńska, A. (2010b). Parallel multi-frontal solver for p adaptive finite element modeling of multi-physics computational problems. *Journal of Computational Science*, *1*(1), 48-54. http://doi.org/10.1016/j.jocs.2010.03.002

Paszyński, M., Pardo, D., Paszyńska, A., & Demkowicz, L. (2011). Out-of-core multi-frontal solver for multi-physics hp adaptive problems. *Procedia Computer Science*, *4*, 1788-1797. http://doi.org/10.1016/j.procs.2011.04.194

Pujii, A., Nishida, A., & Oyanagi, Y. (2005). Evaluation of Parallel Aggregate Creation Orders: Smoothed Aggregation Algebraic Multigrid Method. En M. K. Ng, A. Doncescu, L. T. Yang, & T. Leng (Eds.), *High Performance Computational Science and Engineering* (Vol. 172, pp. 99-122). New York: Springer-Verlag. Recuperado a partir de http://link.springer.com/10.1007/0-387-24049-7_6

Quarteroni, A., & Saleri, F. (2007). *Cálculo Científico con MATLAB y Octave*. Springer Science & Business Media.

Scott, J. A. (2001). The design of a portable parallel frontal solver for chemical process engineering problems. *Computers & Chemical Engineering*, *25*(11–12), 1699-1709. http://doi.org/10.1016/S0098-1354(01)00731-1

Skiba, Y. (2005). *Metodos Y Esquemas Numericos : Un Analisis Computacional*. UNAM.

Stafylopatis, A., & Drigas, A. (1988). On the processing time of a parallel linear system solver. En E. N. Houstis, T. S. Papatheodorou, & C. D. Polychronopoulos (Eds.), *Supercomputing* (Vol. 297, pp. 994-1010). Berlin, Heidelberg: Springer Berlin

Heidelberg. Recuperado a partir de http://bibliotecadigital.usbcali.edu.co:2136/chapter/10.1007/3-540-18991-2_59

Super Computador Medellín. (s. f.). Recuperado a partir de http://www.mdeinteligente.co/estrategia/supercomputador-mas-grande-de-medellin/

Tewarson. (1973). *Sparse matrices*. Academic Press.

Wang, L., Cao, H., Han, X., Liu, J., & Xie, Y. (2015). An efficient conjugate gradient method and application to dynamic force reconstruction. *Journal of Computational Science*, *8*, 101-108. http://doi.org/10.1016/j.jocs.2015.03.008

Wang, L., Sun, W., de Sampaio, R. J. B., & Yuan, J. (2015). A Barzilai and Borwein scaling conjugate gradient method for unconstrained optimization problems. *Applied Mathematics and Computation*, *262*, 136-144. http://doi.org/10.1016/j.amc.2015.04.046

Yuan, G., & Zhang, M. (2015). A three-terms Polak–Ribière–Polyak conjugate gradient algorithm for large-scale nonlinear equations. *Journal of Computational and Applied Mathematics*, *286*, 186-195. http://doi.org/10.1016/j.cam.2015.03.014

Zitney, S. E., Mallya, J., Davis, T. A., & Stad therr, M. A. (1996). Multifrontal vs frontal techniques for chemical process simulation on supercomputers. *Computers & Chemical Engineering*, *20*(6–7), 641-646. http://doi.org/10.1016/0098-1354(95)00198-0

Zitney, S. E., & Stadtherr, M. A. (1993). Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers. *Computers & Chemical Engineering*, *17*(4), 319-338. http://doi.org/10.1016/0098-1354(93)80024-H

Anexos

Anexo 1: Anotaciones pruebas para instalar Trilinos con Mumps

```
cd$home
sudo apt-get install libmumps-4.10.0 libmumps-dev
sudo apt-get install cmake

wget http://trilinos.sandia.gov/download/files/trilinos-11.8.1-Source.tar.gz
tar -xvf trilinos-11.8.1-Source.tar.gz

mkdirtrilinosbuild
cdtrilinosbuild/

apt-get install libblas-devlibblas-doc libblas-test
apt-get install liblapack-devliblapack-doc liblapack-doc-man

##probando python
##aptitude search python2.7*dev
apt-get install python2.7-dev
##funciono

sudo apt-get install python-numpy

## probando swig2.0
## aptitude search swig
sudo apt-get install  swig2.0 swig2.0-doc swig2.0-examples
## aparecio que no encontraba swig. como se esta usando swig2.0 se le indico a trilinos
con -D SWIG_EXECUTABLE:STRING=swig2.0
## perodijoque  SWIG version 2.0.11 is greater than maximum version 2.0.8

## se instala swig 2.0.8 desde el codigo fuente
cd$home
wget http://hivelocity.dl.sourceforge.net/project/swig/swig/swig-2.0.8/swig-
2.0.8.tar.gz
tar -xvf swig-2.0.8.tar.gz
cdswig-2.0.8/
./configure
##pide PCRE deeloper: Cannot find pcre-config script from PCRE (Perl Compatible Regular
Expressions) library package. This dependency is needed for configure to complete ...
sudo apt-get install libpcre3-dev
##se intenta de nuevo instalar SWIG 2.0.8
./configure
## se configuro bien
## se hace make
make
## y se instala en /usr/local
sudo make install
```

```
## instalo sin problemas. La instalacion se obtuvo de
http://sourceforge.net/projects/swig/files/swig/swig-2.0.8/
##se coloco de ejecutable swig SWIG_EXECUTABLE:STRING=swig

cmake -D Trilinos_ENABLE_TESTS:BOOL=ON -D Trilinos_ENABLE_EXAMPLES:BOOL=ON -D
TPL_ENABLE_MPI:BOOL=ON -D TPL_ENABLE_MUMPS:BOOL=ON -D Trilinos_ENABLE_Epetra:BOOL=ON -D
Trilinos_ENABLE_EpetraExt:BOOL=ON -D CMAKE_INSTALL_PREFIX:STRING="/home/jose/trilinos-
install"    -D Trilinos_ENABLE_PyTrilinos:BOOL=ON -D BUILD_SHARED_LIBS:BOOL=ON -D
Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON -D
TPL_BLAS_LIBRARIES:STRING=/usr/lib/libblas.so -D
TPL_LAPACK_LIBRARIES:STRING=/usr/lib/liblapack.so  -D SWIG_EXECUTABLE:STRING=swig
../trilinos-11.8.1-Source

## configuro bien trilinos

##intenta instalar trilinos para varios prcesadore make j3 install si tiene 3
procesadores, etc.
make install
##instalo perfecto

## para ejecutar pytrilinos se debe registrar las librerias
exportPYTHONPATH=/home/jose/trilinos-install/lib/python2.7/site-packages/
exportLD_LIBRARY_PATH=/home/jose/trilinos-install/lib/


## seejecutoprueba_aztec pero no tenia aztec00, se agrego -D
Trilinos_ENABLE_AztecOO:BOOL=ON


cmake -D Trilinos_ENABLE_TESTS:BOOL=ON -D Trilinos_ENABLE_EXAMPLES:BOOL=ON -D
TPL_ENABLE_MPI:BOOL=ON -D TPL_ENABLE_MUMPS:BOOL=ON -D Trilinos_ENABLE_Epetra:BOOL=ON -D
Trilinos_ENABLE_EpetraExt:BOOL=ON -D CMAKE_INSTALL_PREFIX:STRING="/home/jose/trilinos-
install"    -D Trilinos_ENABLE_PyTrilinos:BOOL=ON -D BUILD_SHARED_LIBS:BOOL=ON -D
Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON -D
TPL_BLAS_LIBRARIES:STRING=/usr/lib/libblas.so -D
TPL_LAPACK_LIBRARIES:STRING=/usr/lib/liblapack.so  -D SWIG_EXECUTABLE:STRING=swig  -D
Trilinos_ENABLE_AztecOO:BOOL=ON ../trilinos-11.8.1-Source

cdtrilinosbuild
make
make install

## sevolvio a ejecutar. se corrigieron unos errores y funciono bien.
## se tuvo que corregir lod ceros por o mayucula en AztecOO.
## se pudo ejecutar sin colocar python al inicio porque en la cabecera del archivo
tenia #!/usr/bin/envpython
## primero se volvio ejecutable el archivo
chmod +x /prueba_aztec.py
#se probo con un procesador y funciono como se esperaba
./prueba_aztec.py
#con dos. Funciono bien
mpirun -np2 ./prueba_aztec.py


#se ejecutotest_amesos con SolverType = "Lapack"  pero parece tomar bastante tiempo
./test_amesos.py
#termino sn problema despues de bastante tiempo
```

```
#se ejecutotest_amesos con SolverType = "Klu"  pero parece tomar bastante tiempo
./test_amesos.py


#se ejecutotest_amesos con SolverType = "Mumps"  pero parece tomar bastante tiempo
./test_amesos.py
#ejecuto bien y muestra la respuesta pero al finalizar muestra error al terminar el
proceso con los procesadores

#se instaloumfpack pero no funciona en test_amesos.py
./test_amesos.py

# para ver lista disponibles de parametros se usocmake -L ../trilinos-11.8.1-Source
dentro de trilinosbuild/
# sequitopardiso porque no se pudo instalar -D Amesos_ENABLE_PARDISO:BOOL=ON
# luego se quitaron -D  Trilinos_ENABLE_UMFPack:BOOL=ON -D
Trilinos_ENABLE_UmfPack:BOOL=ON -D Amesos_ENABLE_DSCPACK:BOOL=ON porque no eran
parametrosreconociblesa
# secoloco -D TPL_ENABLE_UMFPACK:BOOL=ON salio error se tuvo que intalar con
sudo apt-get install libumfpack5.6.2 libumfpack4-dev
sudo apt-get install libsuitesparse-dev#estereemplaza a libumfpack4-dev. con
libumfpack4-dev se obtenia Note, selecting 'libsuitesparse-dev' instead of
'libumfpack4-dev'
#se quito -D Amesos_ENABLE_DSCPACK:BOOL=ON porque al compilar no encontraba #include
"dscmain.h"

cmake -D Trilinos_ENABLE_TESTS:BOOL=ON -D Trilinos_ENABLE_EXAMPLES:BOOL=ON -D
TPL_ENABLE_MPI:BOOL=ON -D TPL_ENABLE_MUMPS:BOOL=ON -D Trilinos_ENABLE_Epetra:BOOL=ON -D
Trilinos_ENABLE_EpetraExt:BOOL=ON -D CMAKE_INSTALL_PREFIX:STRING="/home/jose/trilinos-
install"    -D Trilinos_ENABLE_PyTrilinos:BOOL=ON -D BUILD_SHARED_LIBS:BOOL=ON -D
Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON -D
TPL_BLAS_LIBRARIES:STRING=/usr/lib/libblas.so -D
TPL_LAPACK_LIBRARIES:STRING=/usr/lib/liblapack.so  -D SWIG_EXECUTABLE:STRING=swig  -D
Trilinos_ENABLE_AztecOO:BOOL=ON -D TPL_ENABLE_UMFPACK:BOOL=ON -D
TPL_UMFPACK_INCLUDE_DIRS="/usr/include/suitesparse/"  ../trilinos-11.8.1-Source

#se probótest_amesos con SolverType = "Umfpack"
./test_amesos.py
#funciono bien
#tambien con
mpirun -np4 ./test_amesos.py
#funciono bien
```

Anexo 2: Versión de Mumps.py con varias opciones comentadas

```
#! /usr/bin/envpython
fromPyTrilinosimportAmesos, TriUtils, Epetra
Comm=Epetra.PyComm()
Map, A, x, b, Exact=TriUtils.ReadHB("fidap035.rua", Comm)
#print b
Problem =Epetra.LinearProblem(A, x, b)
Factory =Amesos.Factory()

#SolverType = "Paraklete"
#SolverType = "Klu"
#SolverType = "Lapack"
#SolverType = "superlu"
SolverType="Mumps"
#SolverType = "ScaLapack"
#SolverType = "Taucs"
#SolverType = "Umfpack"
#SolverType = "Pardiso"
#SolverType = "Teuchos"
#SolverType = "Dscpack"

#SuperLUDist
#Superlu
#"MUMPS"

Solver =Factory.Create(SolverType, Problem)
AmesosList= {
"MaxProcs": 500,
"PrintStatus": True
}

#[method for method in dir(Factory) if callable(getattr(Factory, method))]
#[method for method in dir(Solver) if callable(getattr(Solver, method))]

Solver.SetParameters(AmesosList)
Solver.SymbolicFactorization()
Solver.NumericFactorization()
Solver.Solve()

file=open("newfile.txt", "w")
foriin x:
         file.write(str(i) +"\n")
         #print i
file.close()

#for i in x:
#  printi

#print len(x)

#print x
```

Anexo 3: Archivo make.inc utilizado en SuperLu_DIST

```
############################################################################
#
#  Program:         SuperLU_DIST
#
#  Module:          make.inc
#
#  Purpose:         Top-level Definitions
#
#  Creation date:   February 4, 1999    version alpha
#
#  Modified:          September 1, 1999  version 1.0
#                   March 15, 2003     version 2.0
#
#                      January 18, 2006   Sam Adams
#                                    General Dynamics - Network Systems
#                        works for i386 Linux, with LAM-MPI 7.1.1 and GCC 4.
#
############################################################################
#
#  The machine (platform) identifier to append to the library names
#
PLAT              = _i386

#
#  The name of the libraries to be created/linked to
#
DSuperLUroot      =${HOME}/SuperLU_DIST_4.0
DSUPERLULIB       =$(DSuperLUroot)/lib/libsuperlu_dist_4.0.a
#
BLASDEF           = -DUSE_VENDOR_BLAS
BLASLIB   = /usr/lib/libblas.so.3
METISLIB= -L/home/jose101/parmetis-4.0.3/metis/include/ -lmetis

############################################################################
## parmetis 4.x.x, 32-bit integer
PARMETIS_DIR      :=${HOME}/Carver/lib/parmetis-4.0.3
## parmetis 4.x.x, 64-bit integer
# PARMETIS_DIR     := ${HOME}/Carver/lib/parmetis-4.0.3_64

METISLIB := -L${PARMETIS_DIR}/build/Linux-x86_64/libmetis -lmetis
PARMETISLIB := -L${PARMETIS_DIR}/build/Linux-x86_64/libparmetis -lparmetis
I_PARMETIS := -I${PARMETIS_DIR}/include -I${PARMETIS_DIR}/metis/include
############################################################################

# Define the required Fortran libraries, if you use C compiler to link
FLIBS             =

# Define all the libraries
LIBS=$(DSUPERLULIB)$(BLASLIB)$(PARMETISLIB)$(METISLIB)
```

```
#
#  Thearchiver and the flag(s) to use when building archive (library)
#  If your system has no ranlib, set RANLIB = echo.
#
ARCH     =ar
ARCHFLAGS=cr
RANLIB   =ranlib

################################################################################
# C compiler setup
CC       =mpicc
# CFLAGS should be set to be the C flags that include optimization
CFLAGS=  -Wall -std=c99 -pipe -O2 ${I_PARMETIS}
#
# NOOPTS should be set to be the C flags that turn off any optimization
NOOPTS             =
################################################################################
# FORTRAN compiler setup
FORTRAN= mpif77
F90FLAGS =
################################################################################
LOADER   = mpif77
LOADOPTS =
################################################################################
#  C preprocessor defs for compilation (-DNoChange, -DAdd_, or -DUpCase)
#
#  Need follow the convention of how C calls a Fortran routine.
#
CDEFS= -DAdd__
```

Anexo 4: Comandos para instalar SuperLu_DIST

```
#compilacionparmetis
wget http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/parmetis-4.0.3.tar.gz
tar -xvf parmetis-4.0.3.tar.gzta ci
cdparmetis-4.0.3
make
makeconfigprefix=~/parmetisinstall
make
make installs

#instalacionsuperlu
cdSuperLU_DIST_4.0
#Compiló bien. Salió make[1]: se sale del directorio
«/home/jose/SuperLU_DIST_4.0/EXAMPLE»
make
# seprobo un ejemplo de SuperLU
cdEXAMPLE/
#con cuatroapareciaRequires at least 10 processes
mpirun -np4 ./pddrive4 g20.rua
```

Anexo 5 Comandos utilizados para instalar Lis

```
## module load openmpi-$(uname -i) ## sol usar si el ubuntu tiene varios mpi
wget  http://www.ssisc.org/lis/dl/lis-1.4.54.tar.gz
tar -xvf lis-1.4.54.tar.gz
cdlis-1.4.54
sudo apt-get install libcr-dev mpich2 mpich2-doc
./configure --prefix=/home/jose/lis-install --enable-mpi## debeaparecer   Build with
MPI library                     = yes


sudoapt-get install build-essential## Con esto se instala make y los demas compiladores

make
make check
make install
makeinstallcheck

cd test
mpirun -np4 ./test1  testmat.mtx0 solution.txt rhistory.txt

## hastaaqui todo funciona bien
##Probando con otras matrices

wget ftp://math.nist.gov/pub/MatrixMarket2/SPARSKIT/fidap/fidap035.rua.gz
gunzip fidap035.rua.gz

wget ftp://math.nist.gov/pub/MatrixMarket2/SPARSKIT/fidap/fidap035.mtx.gz
gunzip fidap035.mtx.gz

module load openmpi-$(uname -i)
mpirun -np4 ./test1 fidap035.mtx 1 solution.txt rhistory.txt
mpirun -np4 ./test1 mahindas.mtx1 solution.txt rhistory.txt
mpirun -np4 ./test1 fidap035.mtx fidap035_rhs1.mtx solution.txt rhistory.txt
mpirun -np4 ./test1  solution.txt rhistory.txt
```

Anexo 6 matriz1.mtx

```
%%MatrixMarketmatrixcoordinate real general
661400
32-1.0
332.0
661.0
45-1.0
442.0
54-1.0
56-1.0
23-1.0
43-1.0
222.0
34-1.0
111.0
65-1.0
552.0
```

Anexo 7 Matriz1.rua

```
matriz1                                                                 125
           10             1             2             5            2
RUA           6             6            14             6
(11i7)          (13i6)          (3D21.15)          (3D21.15)
F             1             0
      1      2      4      7     10     13
     1     2     3     3     4     5     3     4     5     4     5     6     5
     6
-.100000000000000E+010.200000000000000E+01-.100000000000000E+01
-.100000000000000E+010.200000000000000E+01-.100000000000000E+01
-.100000000000000E+010.200000000000000E+01-.100000000000000E+01
-.100000000000000E+010.200000000000000E+01-.100000000000000E+01
-.100000000000000E+010.100000000000000E+01
0.000000000000000E+000.000000000000000E+000.000000000000000E+00
0.000000000000000E+000.000000000000000E+001.000000000000000E+00
```

Anexo 8: Matriz.ele

```
2
1 2
1 0
0 1
0 0
2
2 3
1 -1
-1 1
0 0
2
3 4
1 -1
-1 1
0 0
2
4 5
1 -1
-1 1
0 0
2
5 6
1 1
-1 1
0 1
```

Anexo 9: ConversorEleADict.py

```
classConversorEleADict:

'''
dimension de la matriz
'''
def__init__(self, dimesion):
self.coordDict= {}
self.indDict= {}
self.noZeroElements=0
self.__dimesion=dimesion

self.__initIndDict()

def__initIndDict(self):
i=1
whilei<=self.__dimesion:
self.indDict[i] =float(0)
i=i+1


'''
indices son las columnas
    '''
defagregarCoordenada(self, datos, fila, indices):

j =0
foriin indices:

iffloat(datos[j]) !=0 :
row=int(str(indices[fila-1]).strip())
col=int(str(i).strip())
key= (row, col)
val=float( datos[j] )

self.coordDict[(row, col)] =self.__getValor(key, val)

ifself.coordDict[(row, col)] ==float(0):
delself.coordDict[(row, col)]
self.noZeroElements=self.noZeroElements-1


            j = j +1

def__getValor(self, key, val):
valor=0
if key inself.coordDict:
valor=val+self.coordDict[key]
else:
valor=val
self.noZeroElements=self.noZeroElements+1

return  valor
```

```
defagregarIdependientes(self, datos, indices):

j =0
tam=len(datos)
while j < tam:
idx=int ( indices[j] )
dat=float( datos[j] )

ifidxinself.indDict:
self.indDict[idx] =dat+self.indDict[idx]
else:
self.indDict[idx] =dat

            j = j +1



defgetCoords(self):
returnself.coordDict

defgetInds(self):
returnself.indDict

defgetNoZeroElements(self):
returnself.noZeroElements

defgetDimension(self):
returnself.__dimesion
```

Anexo 10: EscritorCoef.py

```
importsys
classEscritorCoef:
'''
outfile el archivo de salida
dimesion la dimension de la matriz nxn
'''
def__init__(self, outfile, conversorEleADict):
self.outfile=outfile
self.estaAbierto=False
self.conversorEleADict=conversorEleADict

defescribir(self):
self.__abrirArchivo()
self.__escribirEncabezado()
self.__escribirContenido()
self.__cerrarArchivo()


def__abrirArchivo(self):
self.f=open(self.outfile,'w')

self.estaAbierto=True

def__escribirEncabezado(self):
self.f.write('%%MatrixMarket matrix coordinate real general\n')
dim=self.conversorEleADict.getDimension()
self.f.write(str(dim) +' '+str(dim) +'
'+str(self.conversorEleADict.getNoZeroElements()) +' 0 0\n')


def__cerrarArchivo(self):
self.verificarAbierto()
self.f.close()
self.estaAbierto=False

defverificarAbierto(self):
ifself.estaAbierto==False:
sys.exit("No esta abierto el archivo. Abra primero el archivo ")

def__escribirContenido(self):
coords=self.conversorEleADict.getCoords()
fori, value incoords.iteritems():
linea=str(i[0]) +" "+str(i[1]) +" "+str(value) +"\n"
self.__escribirLinea(linea)

def__escribirLinea(self, linea):
self.f.write(linea )
```

Anexo 11: EscritorvarInd.py

```
importsys
classEscritorVarInd:
def__init__(self, outfile, conversorEleADict):
self.outfile=outfile
self.estaAbierto=False
self.conversorEleADict=conversorEleADict

defescribir(self):
self.__abrirArchivo()
self.__escribirEncabezado()
self.__escribirContenido()
self.__cerrarArchivo()

def__escribirContenido(self):
inds=self.conversorEleADict.getInds()
fori, value ininds.iteritems():
linea=str(i) +" "+str(value) +"\n"
self.__escribirLinea(linea)

def__abrirArchivo(self):
print"abrioarchivo -..."
self.f=open(self.outfile,'w')
self.__estaAbierto=True

def__escribirEncabezado(self):
print"encabezado "
self.f.write('%%MatrixMarket vector coordinate real general\n')
self.f.write(str(self.conversorEleADict.getDimension() ) +'\n')

def__escribirLinea(self, linea):
self.f.write(linea )


def__cerrarArchivo(self):
self.__verificarAbierto()
self.f.close()
self.estaAbierto=False

def__verificarAbierto(self):
ifself.__estaAbierto==False:
sys.exit("No esta abierto el archivo. Abra primero el archivo ")
```

Anexo 12: Analizador.py

```
importsys

classAnalizador:

def__init__(self):
self.line=""
self.numeroFila=0
self.indices= []
self.items= []
self.dimension=0# dimension de la submatriz
self.__filasSubMatriz=0
#lo que necesita con cada iteracion
self.necesitaLogintud=True
self.necesitaIndices=False
self.necesitaCoeficientes=False
self.necesitaVarIndependientes=False
#las variables que estan disponibles para recuperar
self.variablesCoeficiente=False
self.variablesIndependientes=False
#print("Creado el Analizador ")

defanalizar(self, line):
self.items=line.split()
ifself.esperaLongitud() ==True :
iflen(self.items) !=1:
sys.exit("Esta linea debe contener la longitud de la submatriz")

self.inicializarVariables()

self.dimension=int(self.items[0])
self.cambiarAIndices()

return0

ifself.esperaIndices() ==True:
#print self.items, self.dimension, len(items)
tam=len(self.items)
ifself.dimension!= tam:
sys.exit("Se esperaba un string con "+str(self.dimension) +" numeros (Indices)")

self.inicializarVariables()

self.indices=self.items
self.cambiarACoeficientes()

return0

ifself.esperaCoeficientes() ==True:
tam=len(self.items)
ifself.dimension!=tam:
sys.exit("Se esperaba un string con "+str(self.dimension) +" numeros (Coeficientes)")
```

```
self.actualizarFilasSubMatriz()
self.setTrueVariablesCoeficiente();
#print( "fila " + str(self.__filasSubMatriz) )
self.cambiarDesdeEsperaCoeficientes()

return0

ifself.esperaIndependiente() ==True:
#print "entraindependiente"
self.inicializarVariables()

tam=len(self.items)
ifself.dimension!=tam:
sys.exit("Se esperaba un string con "+str(self.dimension) +" numeros (Variables
independientes)")


self.setTrueVariablesIndependientes();
self.cambiarALongitud()

return0

definicializarVariables(self):
self.__filasSubMatriz=0
self.setFalseVariablesEscritura()

defactualizarFilasSubMatriz(self):
self.__filasSubMatriz=self.__filasSubMatriz+1

defcambiarDesdeEsperaCoeficientes(self):
ifself.__filasSubMatriz==self.dimension :
self.cambiarAVarIndependientes()
#self.__filasSubMatriz = 0

defcambiarAIndices(self):
self.necesitaLogintud=False
self.necesitaIndices=True
self.necesitaCoeficientes=False
self.necesitaVarIndependientes=False

defcambiarACoeficientes(self):
self.necesitaLogintud=False
self.necesitaIndices=False
self.necesitaCoeficientes=True
self.necesitaVarIndependientes=False

defcambiarAVarIndependientes(self):
self.necesitaLogintud=False
self.necesitaIndices=False
self.necesitaCoeficientes=False
self.necesitaVarIndependientes=True

defcambiarALongitud(self):
self.necesitaLogintud=True
self.necesitaIndices=False
self.necesitaCoeficientes=False
self.necesitaVarIndependientes=False
```

```
defesperaLongitud(self):
ifself.necesitaLogintud==Trueandself.necesitaIndices==Falseandself.necesitaCoeficientes
==Falseandself.necesitaVarIndependientes==False:
returnTrue

returnFalse

defesperaIndices(self):
ifself.necesitaLogintud==Falseandself.necesitaIndices==Trueandself.necesitaCoeficientes
==Falseandself.necesitaVarIndependientes==False:
returnTrue

returnFalse

defesperaCoeficientes(self):
ifself.necesitaLogintud==Falseandself.necesitaIndices==Falseandself.necesitaCoeficiente
s==Trueandself.necesitaVarIndependientes==False:
returnTrue

returnFalse

defesperaIndependiente(self):
ifself.necesitaLogintud==Falseandself.necesitaIndices==Falseandself.necesitaCoeficiente
s==Falseandself.necesitaVarIndependientes==True:
returnTrue

returnFalse


defsetTrueVariablesCoeficiente(self):
self.variablesCoeficiente=True
self.variablesIndependientes=False

defsetTrueVariablesIndependientes(self):
self.variablesCoeficiente=False
self.variablesIndependientes=True

defsetFalseVariablesEscritura(self):
self.variablesCoeficiente=False
self.variablesIndependientes=False

defisTrueVariablesCoeficiente(self):
ifself.variablesCoeficiente==Trueandself.variablesIndependientes==False:
returnTrue

returnFalse

defisTrueVariablesIndependientes(self):
ifself.variablesCoeficiente==Falseandself.variablesIndependientes==True:
returnTrue

returnFalse

defesEstadoCoeficiente(self):
returnself.isTrueVariablesCoeficiente()

defesEstadoVariableIndependientes(self):
returnself.isTrueVariablesIndependientes()
```

```
defgetVariablesCoeficiente(self):
ifself.isTrueVariablesCoeficiente:
return (self.items, self.__filasSubMatriz, self.indices)

sys.exit("No se puede ejecutar la funcion porque no esta en estado Variable de
coeficientes")

defgetVariablesIdependientes(self):
ifself.isTrueVariablesIndependientes:
return (self.items, self.indices)

sys.exit("No se puede ejecutar la funcion porque no esta en estado Variable
independientes")
```

Anexo 13 Main.py

```
fromConversorimportConversor
fromConversorEleADictimportConversorEleADict
fromEscritorCoefimportEscritorCoef
fromEscritorVarIndimportEscritorVarInd
fromAnalizadorimportAnalizador
fromEscritorimportEscritor

defmain():

conversorEleADict=ConversorEleADict(6)

analizador=Analizador()
escritorCoef=EscritorCoef("new.mtx", conversorEleADict, )
escritorVarInd=EscritorVarInd("independientes1.txt", conversorEleADict);
conversor= Conversor('matriz1.ele', analizador, conversorEleADict, escritorCoef,
escritorVarInd)
conversor.convertir()



if __name__ =='__main__':
main()
```

Anexo 14MatrizBanda.py

```
classMatrizBanda:

def__init__(self, n, nozeros):
self.__n =int(n)
self.__nozeros =int(nozeros)
self.__bandaSup =0
self.__bandaInf =0

defactualizarBandas(self,  x, y):
if x == y: return0
        ancho =self.__getAnchobanda(x,y)

#banda superior
if y > x and ancho >self.__bandaSup:
self.__bandaSup = ancho

# banda ancha inferior
if x > y and ancho >self.__bandaInf:
self.__bandaInf = ancho


defgetDensidad(self):
        t =self.totalElem()
        densidad =self.__nozeros *1./ t

return densidad

def__getAnchobanda(self, x, y):
        ancho =abs(y - x)
return ancho

deftotalElem(self):
       t =self.__n +self._elemBanda(self.__bandaInf) +self._elemBanda(self.__bandaSup)

return t

def_elemBanda(self, banda):
returnself.__n * banda -self.__elemARestar(banda)

def__elemARestar(self, banda):
        b = banda +1
        total = b * (b -1) *1./2
return total


defgetBandaSup(self):
returnself.__bandaSup

defgetBandaInf(self):
returnself.__bandaInf
```