Reporte de la Búsqueda de Bibliotecas Numéricas de Algebra Lineal, de Alto Rendimiento, de Dominio Público en INTERNET.

Golfredo Catalani [1]**, y Javier Gutiérrez**[2]
Centro Nacional de Cálculo Científico
Universidad de Los Andes (CeCalCULA)*,*
Corporación Parque Tecnológico de Mérida, Mérida 5101, Venezuela

Luis A. Núñez [3]
Centro de Astrofísica Teórica,
Departamento de Física, Facultad de Ciencias,
Universidad de Los Andes, Mérida 5101, Venezuela, y
Centro Nacional de Cálculo Científico
Universidad de Los Andes (CeCalCULA)*,*
Corporación Parque Tecnológico de Mérida, Mérida 5101, Venezuela

Versión α 2.0, Junio 2000

[1]golfredo@cecalc.ula.ve
[2]gutierre@cecalc.ula.ve
[3]nunez@ciens.ula.ve

Resumen

Hacemos una descripción de ocho de las bibliotecas disponibles en INTERNET para resolver problemas de álgebra lineal en computadores de alto rendimiento. Se muestra también una tabla donde se resume las capacidades de 37 bibliotecas de alto rendimiento de algebra lineal disponibles en internet y de dominio público.

Índice

1. Introducción

Debido a la forma de crecimiento de la INTERNET, se ha ido formando en ella un gigantesco repositorio de herramientas para la programación. Sin embargo, la mayoría de esas herramientas se presentan sin la documentación necesaria para el máximo aprovechamiento por parte de usuarios ajenos al proyecto que las generó. Es por ello que se requiere hacer una selección de funciones, de dominio público, para resolver problemas de álgebra lineal. Esta selección será evaluada y documentada, *a posteriori*, para permitir a los usuarios llegar rápidamente a la rutina que necesitan para resolver su problema y saber cuales son las capacidades y limitaciones de dichas rutinas. La idea detrás de esto es reducir el tiempo que transcurre entre la concepción de una idea y la implantación de ésta en una plataforma, es decir, que se pueda pasar con mínimo esfuerzo de los prototipos desarrollados en aplicaciones tipo **Matlab** a programas escritos en **C** y **Fortran** con su interfaz gráfica.

El énfasis en la selección de las bibliotecas se hará sobre rutinas con la capacidad para tratar con matrices, esparcidas o densas, de gran tamaño (del orden de 10^8 elementos, i.e., 10000×10000 en matrices densas), que puedan ser utilizadas desde **Fortran90** y **C**, y que sean escalables a equipos de alto rendimiento.

Este documento está dividido en dos partes principales, la sección 2 incluye una revisión de los conceptos básicos utilizados en la Computación de Alto Rendimiento y tiene por objeto establecer un lenguaje común. En la sección 3 se hace la descripción de las características más relevantes de ocho bibliotecas con rutinas para la solución de problemas de álgebra lineal y por último en la sección 4 se muestra una tabla que resume las capacidades de las bibliotecas de álgebra lineal numérica disponibles en Internet según **HPCNetLib** (*High Performance Computing NetLib*)[1]. En el Apéndice se describe el efecto que tiene sobre el rendimiento del código el orden en que se accede a los elementos de un arreglo.

2. El Algebra Lineal y las Computadoras de Alto Rendimiento

En esta sección vamos a revisar algunas características de la computación de alto rendimiento así como particularidades de algunas plataformas actuales y las tendencias existentes. Esto con el fin de conocer las características relevantes que deben tener los códigos de Algebra Lineal Numérica de alto rendimiento.

2.1. Tendencias en el diseño de Computadoras

En la década de los 1980s, la tecnología de microprocesadores experimentó un desarrollo verdaderamente sorprendente, llevando la operación de los dispositivos cerca del límite teórico. Los pulsos eléctricos en una computadora moderna viajan entre 9,1 y 27,4 cm por nanosegundo y la velocidad de la luz es 30 cm por nanosegundo. Sin embargo, pronto las capacidades de los microprocesadores se quedaron pequeñas ante los problemas que se planteaban en la ciencia, la ingeniería y los negocios, esto motivó el diseño y la construcción de máquinas de procesamiento paralelo.

La primera tendencia en este sentido fue la fabricación de máquinas propietarias altamente paralelas basadas en la última tecnología existente. Ahora surge como alternativa a estas máquinas la utilización de arreglos de máquinas, de uno o varios procesadores, interconectadas con una red rápida, para resolver un problema en forma paralela o distribuida.

[1] http://rib.cs.utk.edu/cgi-bin/catalog.pl?rh=222&term=0!2

La gran desventaja del paralelismo es la dificultad para la programación de aplicaciones, lo que ha traido como consecuencia que el desarrollo de *softwre* se ha rezagado con respecto al desarrollo del *hardware*.

2.2. El paralelismo y la tecnología de microprocesadores

El primer nivel en el que encontramos procesamiento paralelo en las máquinas modernas es el nivel de los microprocesadores, a través de tecnologías como la inclusión de múltiples unidades funcionales, el *pipelining* y el solapamiento de instrucciones de operación.

El uso de múltiples unidades funcionales se refiere a que los procesadores actuales contienen una **Unidad Aritmético-lógica**, una **Unidad de Adición en Punto Flotante**, una **Unidad de Multiplicación en Punto Flotante** y en los procesadores más modernos hay una **Unidad de División en Punto Flotante**. Estas unidades pueden, en general, trabajar en forma independiente y concurrente.

En el *pipelining*, cada unidad funcional se divide en etapas, de modo que cada etapa realiza una parte en la decodificación, interpretación y ejecución de una operación. Si se tiene un conjunto de operaciones similares consecutivas, éstas se hacen concurrentemente de modo de obtener un resultado de varias de ellas en cada ciclo de reloj. En la actualidad esta técnica está incluida en todos los procesadores **RISC**, y en las últimas generaciones de CPU para PC.

2.3. Organización y manejo de la Memoria

El modelo ideal de una computadora secuencial es representado a menudo como un flujo de datos que van de la memoria a las unidades de procesamiento y vuelven a la memoria, como se muestra en la figura 1

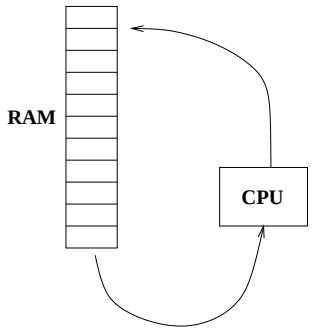

Figura 1: Máquina de von Neuman

En un intento de acercarse a este modelo ideal, se ha implantado una jerarquía de memorias, de diferentes tamaños y velocidades, como la que se ve en la figura 2. Esta jerarquía intenta disminuir la latencia en el acceso a los datos, es decir, el tiempo que el procesador tiene que esperar cada vez que solicita un dato almacenado en la memoria [1, 2, 3].

La existencia de esta jerarquía de memoria exige al programador un cuidado mayor al momento de acceder a los datos desde una aplicación, ya que el tiempo de acceso a los mismos depende de su ubicación dentro de la estructura de memoria. Los tiempos promedio de acceso a cada una de estas memorias y sus tamaños típicos son:

En el caso de códigos de álgebra lineal no es extraño tener matrices que se encuentran distribuidas por toda la estructura de memoria, por lo cual, el orden en que se accede a los datos

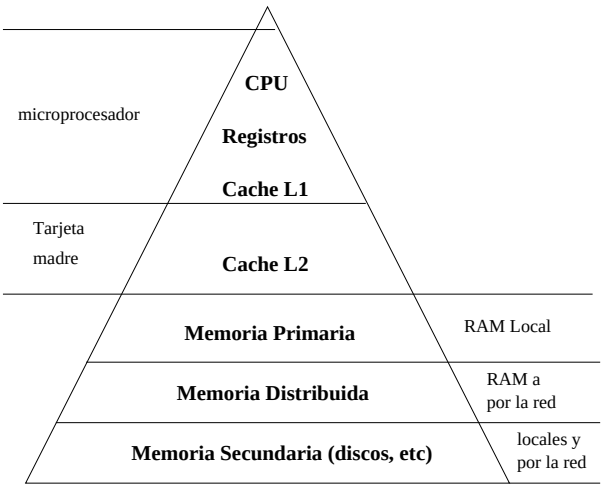

Figura 2: Jerarquía de memoria de los computadores modernos

tipo de memoria	latencia	tamaño
registros	1 ciclo	32 - 64 bits
cache L1	1-2 ciclos	128 -256 KB
cache L2	2-4 ciclos	1 - 4 MB
RAM	25 -100 ciclos	64MB - 2GB
Memoria Distribuida	500 ciclos	$RAM \times \#$ de nodos
Memoria Secundaria	$>$ 1000 ciclos	muy grande

Cuadro 1: Jerarquía de memoria de los computadores modernos[1]

afecta significativamente el rendimiento de la aplicación, como muestra Jack Dongarra *et al*[3] en un ejemplo del tipo de optimizaciones utilizadas en la biblioteca **BLAS** [2] (ver apéndice).

2.4. Paralelismo

Existen dos tendencias básicas en el procesamiento paralelo:

1. **SIMD** (*single instruction stream/multiple data stream*). En este caso, múltiples unidades de procesamiento reciben simultaneamente una instrucción desde la unidad de control, y operan sobre datos diferentes.

2. **MIMD** (*Multiple Instruction Stream/Multiple Data Stream*). Este esquema supone un conjunto de procesadores que interactuan con la memoria por medio de una red. En este caso el trabajo de particionamiento y asignación de los datos y las tareas es esencial. En esta categoría se encuentran la mayoría de las máquinas multiprocesadores actualmente.

Los modelos SIMD y MIMD pueden ser aplicados tanto en el *hardware* como en el *software*. La tendencia actual, sin embargo, es que el *hardware* cumple con el modelo **MIMD** y que el programador utilice la técnica que más conveniene para resolver el problema.

[2] http://www.netlib.rog/blas/index.html

En el diseño de *hardware* tiene tres vertientes importantes en el mercado: Los computadores de Memoria Compartida, los de Memoria Distribuida y los CLUMPS, que son un híbrido de los anteriores.

En las máquinas de memoria compartida, se tiene un grupo de CPUs que acceden a un banco de memoria a través de un bus. Estas máquinas poseen un sistema operativo con la capacidad de balancear la carga entre los CPU. En este tipo de plataforma el programador utiliza mayormente las "hebras", o sub-procesos (que se generan y destruyen según se requieran), cada una de estas hebras se encargan de realizar una parte del trabajo. La asignación de las hebras a cada CPU es, por lo general, aleatoria y está a cargo del sistema operativo.

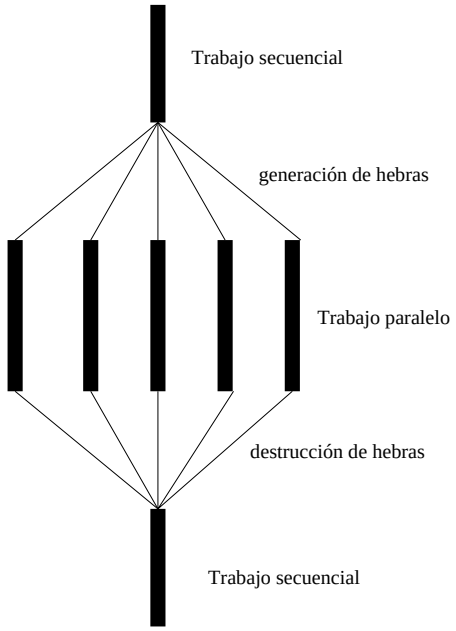

Figura 3: parelelismo multi-hebras

Por otro lado las máquinas de memoria distribuida consisten en un grupo de computadoras cada una con CPU y memoria local. El trabajo paralelo en esta plataforma se hace enviando mensajes a través de una red, para pasar la información de la memoria local a una remota y viceversa.

Los *Clusters*, son básicamente computadores de memoria distribuida, la diferencia entre los *clusters* y las máquinas "propietarias" de memoria distribuida es el tipo de red que se utiliza, por ejemplo, la máquina SP-2 de IBM es de memoria distribuida y está formada por un conjunto de nodos unidos por un *switch* de muy alta velocidad, el cual permite comunicación a velocidades de 80 MBytes por segundo y con una latencia de 35 μs; por otro lado las redes más utilizadas en los clusters son **Fast Ethernet** a 10 MBytes por seg. y con una latencia de 130 μs o **FDDI** a 20 MBytes por seg. y con latencia del orden de los 500 μs. También exiten redes "propietarias", como **Myrinet**[3], diseñada para dar a los *cluster* una comunicación de alto rendimiento y que se hacen cada vez más estándar en su utilización.

Por otro lado, la aparición en el mercado de PCs de máquinas con dos o cuatro CPUs, ha llevado a una nueva generación de clusters llamados **CLUMPS** por las iniciales de *CLUster of MultiProcesorS*.

[3]http://www.myri.com

2.5. Pase de Mensajes

El esquema de pase de mensajes está basado en la suposición de que cada proceso tiene una memoria local privada, y que puede comunicarse con los otros procesos a través de la tranferencia de mensajes. El pase de mensajes es la técnica de programación por excelencia en las plataformas de memoria distribuida.

En el pase de mensajes, la transferencia de datos entre procesadores se hace a través de un simple mecanismo de envio y recepción, donde las variantes radican en la forma como el emisor y el receptor verifican el envio y la posibilidad de solapamiento entre envios y cálculo.

Las principales características del pase de mensaje son:

- Cada procesador tiene su propia memoria y los datos deben ser transferidos de una memoria a otra.
- Si cada uno de los n procesos declara una variable p, habrá n variables llamadas p y con valores que pueden ser diferentes.
- Si la evaluación de una función en un proceso requiere datos que están en la memoria de otro proceso, estos datos deben ser transmitidos en un mensaje.
- El programador debe dividir explícitamente las estructuras de datos y repartirlas entre los procesos.
- Salvo el **HPF** (*High Performace Fortran*), los compiladores no proporcionan ayuda en el particionamiento de los datos, ni en la comunicación, cuando se programa en pase de mensajes.

Existen varios paquetes y bibliotecas de programación que permiten simular el ambiente de pase de mensajes en máquinas de memoria compartida, con la intención de lograr un estándar que permite la portatilidad de las aplicaciones. En la actualidad el estándar más utilizado en la programación por pase de mensajes es el **MPI** (*Message Passing Interface*)[5, 4], y está disponible prácticamente en todas las plataformas del mercado y en una variedad de distribuciones tanto públicas como comerciales.

3. Bibliotecas de Algebra Lineal

Dentro de este capítulo se recogen las bibliotecas de álgebra lineal seleccionadas dentro de las 37 reportadas por J. Dongarra en la matriz del software público que se muestra en la sección 4. De cada una de las bibliotecas se hace una descripción basada en la información provista por los desarrolladores.

3.1. ScaLAPACK

ScaLAPACK[4] (*Scalable Linear Algebra PACKage*) es una biblioteca de subprogramas para cálculos de álgebra lineal en computadores paralelos de memoria distribuida. Está basada en algoritmos de particionamiento en bloques lo que minimiza el movimiento de datos entre los diferentes niveles de la jerarquía de memoria, ver sección 2.3. Esta biblioteca soporta computación heterogénea, es decir, puede correr simultáneamente y en la solución de un solo problema sobre máquinas de distinto Sistema Operativo.

[4] http://www.netlib.org/scalapack/

Una de las principales características de **ScaLAPACK** es que su utilización es similar en casi todos los aspectos a la versión secuencial llamada **LAPACK**[5].

3.1.1. Capacidades

- Sistemas Lineales densos, en bandas y tridiagonales
  - Generales
  - Simétricos definido positivos
- Mínimos cuadrados lineales
- Factorizaciones Ortogonales Estándarer y Generalizadas
- Rutinas Utilitarias
  - Descomposición LU
  - Sistemas "*Out-of-core*"
  - Problemas de Autovalores
    - Autovalores simétricos
    - Autovalores no simétricos
    - Autovalores simétricos generalizados
- Utiliza un espacio de comunicación privado para garantizar la integridad de los datos durante el cálculo
- Actualmente está en desarrollo una interfaz para ser utilizada desde **HPF**

La partición de los datos y la asignación de los mismos a los procesos debe hacerla el programador.

3.1.2. Requerimientos de *hardware* y *software*

Las rutinas de **ScaLAPACK** requieren la instalación de alguna distribución de **MPI**[6] o de **PVM**[7] (*Parallel Virtual Machine*).

También se requieren las bibliotecas de subrutinas: **PBLAS**[8] (*Parallel Basic Linear Algebra Subprograms*), **BLACS** [9] (*Basic Linear Algebra Comunication Subprograms*) y **BLAS**[10] (*Basic Linear Algebra Subprograms*). Estas últimas pueden ser obtenidas del mismo sitio web[6].

Es recomendable una interconexión escalable entre los procesadores, de modo que se pueda aprovechar la escalabilidad de las rutinas.

ScalaPACK se ve favorecido en el uso de **CLUMPS**.

[5] http://www.netlib.org/lapack
[6] http://www.netlib.org/mpi/index.html
[7] http://www.epm.ornl.gov/pvm/pvm_home.html
[8] http://www.netlib.org/scalapack/
[9] http://www.netlib.org/blacs/index.html
[10] http://www.netlib.org/blas/index.html

3.1.3. Portatilidad

ScaLAPALCK está desarrollada sobre los estándares de **F77**, **F90** y **ANSI C** y compila con los compiladores de **GNU**, así como con compiladores nativos en una gran cantidad de plataformas. Debido a que utiliza la instalación de **MPI** (o **PVM**, **IBM MP**, etc) existente, puede ser compilado tanto en máquinas paralelas, como cluster de PC o estaciones de trabajo.

3.1.4. Aplicaciones

ScaLAPALCK ha sido incorporada a las bibliotecas comerciales **IBM-Parallel ESSL**, **NAG Numerical Libraries**, **SGI Cray Scientific Software Library** y **VNI Distributed IMSL**, así como en bibliotecas de **Fujitsu**, **HP/Convex**, **Hitachi** y **NEC**.

3.1.5. Documentación

ScaLAPACK cuenta con una extensa colección de tutoriales y ejemplos en línea a través de la Web[6], además de una Guía del Usuario publicada por **SIAM**[7].

3.2. The NIST Sparse BLAS

Las **NIST Sparse BLAS** (*National Institute of Standards and Technology Sparse Basic Linear Algebra Subprogram*) son un conjunto de subrutinas que proveen un núcleo computacional para operaciones fundamentales con matrices esparcidas.

3.2.1. Capacidades

NIST S-BLAS posee rutinas para las siguientes operaciones:

- Producto de matrices esparcidas

$$\begin{aligned} C &\leftarrow \alpha AB + \beta C \\ C &\leftarrow \alpha A^T B + \beta C \end{aligned}$$

- Solución de sistemas triangulares

$$\begin{aligned} C &\leftarrow \alpha D_L A^{-1} D_R B + \beta C \\ C &\leftarrow \alpha D_L A^T D_R B + \beta C \end{aligned}$$

donde A es la matríz esparcida, B y C son matrices densas o vectores densos, y D_L y D_R son matrices diagonales.

La última versión de esta biblioteca tiene soporte para los siguientes formatos de matrices esparcidas: *compressed-row*, *compressed-column*, y *coordinate storage*, junto con las versiones *block* y *variable-block* de estos formatos. También tiene soporte para las versiones simétricas y antisimétricas de estos formatos.

Esta biblioteca posee un grupo de subrutinas que conforman las **Sparse BLAS Lite**, a las cuales se les ha eliminado el sobrepeso del manejo de errores y los bloques *CASE* de modo que sean más rápidas para el manejo de matrices pequeñas.

3.2.2. Lenguajes

Las **NIST S-BLAS** están desarrolladas en **ANSI C** y pueden ser llamadas desde **Fortran** a través del **Sparse BLAS Toolkit** que se incluye en la distribución.

Existe una versión completamente desarrollada en **Fortran**[8] para evitar el uso de la interfaz con las rutinas escritas en **C**, sin embargo, la versión en **Fortran** no está optimizada.

3.2.3. Requerimientos de *hardware* y *software*

La instalación estándar (sólo rutinas en **C**) de las **NIST S-BLAS** sólo requiere de un compilador **ANSI C**. Si se desea instalar la versión completa se requiere, además, un compilador de **Fortran 77**.

3.2.4. Portatilidad

El desarrollo de esta biblioteca ha sido hecho en **ANSI C** para asegurar la portatilidad a cualquier plataforma que tenga un compilador que cumpla con esta norma.

3.2.5. Aplicaciones

El rendimiento de esta biblioteca ha sido probado en el modelado de una planta química [11] y de una planta de Etileno [12]

3.2.6. Documentación

NIST S-BLAS cuenta con documentación en línea [8, 9].

3.3. PARPACK

PARPACK es la versión paralela de **ARPACK**[13], que es una colección de rutinas, en **Fortran 77**, para resolver problemas de autovalores de matrices esparcidas de gran tamaño, simétricas o generales.

3.3.1. Capacidades

- Interfaz de Comunicación reversa.
- Problemas generalizados simétricos y antisimétricos en precisión simple o doble.
- Problemas Estándar o Generalizados con números complejos en precisión simple o doble.
- Rutinas para Matrices en Bandas - Estándar o Generalizadas.
- Rutinas para Descomposición en Valor Singular.
- Rutinas de ejemplo que pueden ser utilizadas como modelo para implantar numerosos *Shift-Invert*
- Estrategias para todos lo tipos de probemas, datos y precisiones.

[11] *http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/chemwest/west0156.html*
[12] *http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/chemimp/impcol_c.html*
[13] http://www.caam.rice.edu/software/ARPACK/

3.3.2. Lenguajes

PARPACK es escrita en el estándar de **Fortran 77** y puede ser utilizado desde **Fortran**, **C** y **C++**.

3.3.3. Requerimientos de *hardware* y *software*

PARPACK requiere de una instalación de **MPI** o de **PVM** y utiliza la biblioteca de dominio público **BLACS** que no está incluida en la distribución

3.3.4. Portatilidad

PARPACK compila en cualquier máquina que tenga un compilador de **f77** y dado que hace uso de la instalación existente de **MPI** o **PVM**, puede ser utilizada en máquinas paralelas tanto de memoria compartida como distribuida.

3.3.5. Aplicaciones

Algunas de las aplicaciones en las que se utiliza actualmente esta biblioteca son:
[14]

- Descomposición de Valor Simple (SVD) para Reconstrucción de Imágenes del Virus *Herpes Simplex* (*http://ncmi.bioch.bcm.tmc.edu/facultylibrary/zhou.html*)
- SVD a gran escala para Simulación en dinámica molecular. (*http://www.bioc.rice.edu/ tromo/Sprez/*)
- Análisis de estabiblidad del crecimiento de cristales
- Dispersión Reactiva

3.3.6. Documentación

La distribución incluye una guía de usuarios y se consiguen en la página de **PARPACK**[10] algunas referencias con información sobre el uso de esta biblioteca.

3.4. Spooles

Spooles es una biblioteca de funciones para resolver sistemas de ecuaciones esparcidas en variables reales o complejas. Está diseñada para trabajar en máquinas de memoria compartida.

3.4.1. Capacidades

- Factoriza y resuelve sistemas lineales de ecuaciones con estructura simétrica, con o sin pivote. La factorización puede ser simétrica LDL^T, Hermítica LDL^H, o no-simétrica LDU. Puede calcular factorizaciones directas o factorizaciones "*drop tolerance*". Todos estos cálculos pueden ser realizados en modo serial, con *multithreaded* en las normas **Solaris** o **POSIX**, o en multitarea vía **MPI**.
- Factoriza y resuelve sistemas sobredeterminados "full rank" utilizando la factorización multifrontal QR, en modo secuencial o utilizando hebras **POSIX**.

[14] *http://www.caam.rice.edu/~kristyn/applications.html*

- Resuelve sistemas lineales utilizando una variedad del método iterativo de Krylov. El precondicionador es una factorización *drop tolerance*, construido con o sin pivote.

3.4.2. Lenguajes

Esta biblioteca está desarrollada en **C** y tiene un diseño orientado a objetos. Hasta el momento no se ha desarrollado una interfaz para utilizar esta biblioteca desde el **Fortran**

Esta biblioteca puede ser utilizada desde **C** y **C++**.

3.4.3. Requerimientos de *hardware* y *software*

Dependiendo del modo de multitarea que se quiera utilizar se requiere una instalación de **MPI** y/o un sistema operativo que cumpla con las normas **POSIX** y **Solaris** para la creación y manejo de multi-hebras.

3.4.4. Portatilidad

Aún cuando está escrito en **C** debe ser compilado en una plataforma (sistema operativo + compilador) que cumpla con las normas de generación de hebras que utiliza el paquete.

3.4.5. Documentación

La documentación está incluida en la distribución de la biblioteca, en formato LaTeX y está disponible también en la pagina [11] en formato *postscript*.

3.5. SuperLU

SuperLU es una biblioteca de propósitos generales para la solución directa de sistemas lineales de ecuaciones, esparcidos y grandes, en máquinas de alto rendimiento.

3.5.1. Capacidades

Esta biblioteca realiza una descomposición LU con pivote parcial, y resuelve sistemas triangulares con substitucion *backward* o *forward*. La factorización LU puede se realizada sobre matrices cuadradas o rectangulares, mientras que la solución de sistemas triangulares sólo puede trabajar sobre matrices cuadradas.

Esta biblioteca provee rutinas de refinamiento iterativo de la precisión para mejorar la estabilidad en la substitución *backward*.

3.5.2. Lenguajes

SuperLU está escrita en **C** y es utilizable tanto desde **C** como desde **Fortran**

3.5.3. Requerimientos de *hardware* y *software*

Existen tres versiones de la biblioteca **SuperLU**:

- **SupreLU**: Para máquinas secuenciales.
- **SuperLU_MT**: Para máquinas de memoria compartida. Utiliza directivas para algunas plataformas **SMP** y posee también la posibilidad de utilizar hebras **POSIX**.

- **SupreLU_DIST**: Para máquinas de memoria distribuida. Utiliza **MPI** para la comunicación inter-procesos.

3.5.4. Portatilidad

El desarrollo de esta biblioteca se basa en los estándares para asegurar la portatilidad, sin embargo cada una de las versiones requiere algunas características de los compiladores y del sistema operativo.

La versión **SuperLU_MT** requiere de un sistema operativo y de compiladores que cumplan con las normas **POSIX** para la creación y manejo de hebras.

La versión **SupreLU_DIST** requiere de una instalación de **MPI**.

3.5.5. Documentación

La documentación está provista en formato **postscript** en la página Web de la biblioteca [12]. En esta misma página se presentan enlaces a diferentes documentos que resumen el uso de esta bibloteca, así como reportes del rendimiento de la misma en diferentes plataformas.

3.6. PETSc

PETSc (*Portable, Extensible Toolkit for Scientific Computation*) es un conjunto de herramientas de manejo de datos para solución de problemas científicos de gran escala, entre otras características, incluye rutinas para manejo de matrices esparcidas, rutinas gráficas, resolvedores (solvers) de ecuaciones diferenciales en derivadas parciales, entre otras facilidades.

3.6.1. Capacidades

PETSc es una biblioteca diseñada para resolver problemas que involucran sistemas de ecuaciones diferenciales en derivadas parciales, debido a esto, incluye una extensa variedad de rutinas para realizar operaciones con matrices tanto densas como esparcidas. Utiliza, para las matrices esparcidas y como formato por omisión, el *Yale sparse matrix format* conocido también como *compress-row format*.

Entre las herremientas para manejo de matrices provee precondicionadores, métodos iterativos de Kyrlov, método truncado de Newton, pruebas y monitoreo de convergencia de los métodos iterativos. Posee un módulo grafico que permite monitorear en tiempo real la convergencia de los métodos iterativos de Kyrlov.

3.6.2. Lenguajes

La biblioteca **PETSc** ha sido desarrollada en **C** y puede ser utilizada desde **C**, **C++** y **Fortran**.

3.6.3. Requerimientos de *hardware* y *software*

Requiere una instalación de **MPI**.

Utiliza rutinas de **BLAS**, **LAPACK**, entre otras. Tiene interfaz para interactuar con **BlockSolve95**, ver sección 3.8, para los precondicionadores IC(0)[15] e ILU(0)[16] paralelos, **ESSL** [17] para

[15] IC es el precondicionador para la factorización incompleta de Cholesky para matrices esparcidas simétricas y definidas positivas[3]

[16] ILU(0) es el caso antisimétrico de IC(0)[3]

[17] IBM - Engineering and Scientific Subroutine Library (http://www.rs6000.ibm.com/resource/aix_resource/sp_books/essl/)

factorización LU rápida en máquinas IBM, **Matlab**[18], **ParMeTiS**[19] para particionamiento paralelo de grafos, **PVODE**[20] un integrador paralelo de ODEs, **SPAI**[21] para el precondicionador inverso aproximado y esparcido en paralelo. Todos estos paquetes son opcionales.

3.6.4. Portatilidad

La biblioteca **PETSc** ha sido compilada y utilizada con éxito en las siguientes plataformas:

- IBM RS6000 (incluyendo SP, rs6000, rs6000_sp for SP, rs6000_gnu para compiladores GNU, rs6000_64 para procesadores Power3 en 64 bit)
- SGI Workstations (IRIX)
- SGI Power Challenge (IRIX64, IRIX para 32 bit)
- SGI Origin (IRIX64,IRIX para 32 bit)
- Dec Alpha sobre OSF (alpha)
- HP (incluyendo Convex Exemplar, HPUX)
- Sun Sparcstations corriendo Solaris (solaris, solaris_gnu para compiladores GNU)
- Solaris sobre Intel (solaris_x86)
- Cray T3E
- Linux sobre Intel
- Windows NT o Windows 95 (win32 para los compiladores Microsoft developers studio, win32_gnu para los compiladores GNU)
- FreeBSD sobre Intel (freebsd)

3.6.5. Aplicaciones

La biblioteca **PETSc** es activamente utilizada en el **Argone National Laboratory** (ANL):

- En colaboración con el *Center for Subsurface Modeling* y el *Center for Petroleum and Geosystems Engineering* en la *Universidad de Texas* en **Simulación de reservorios de petroleo** (*http://www-fp.mcs.anl.gov/petsc/apps/structured.htm* y *http://www.pe.utexas.edu/CPGE/new_generation/*)
- Métodos computacionales en **Aerodinámica y Acústica** (*http://www-fp.mcs.anl.gov/petsc/apps/unstructured.htm*)

Además ha sido utilizada fuera del ANL en:

- Análisis de flujo multifase (*http://cnls-www.lanl.gov/~qzou/*)
- Simulaciones en electromagnetismo (*http://www.llnl.gov/casc/emsolve/*)
- Mecánica estructural (*http://www.cs.berkeley.edu/~madams/#Prometheus*)

[18] http://www.mathworks.com/
[19] http://www-users.cs.umn.edu/ karypis/metis/
[20] http://www.llnl.gov/CASC/PVODE/
[21] http://www.math.ethz.ch/ grote/spai/spai-page.html

3.6.6. Documentación

PETSc Cuenta con una excelente documentación en su página web [13], la cual puede ser obtenida en formato **postscript** para ser impresa. Cuenta además con una gran cantidad de ejemplos con el uso de cada rutina.

3.7. ATLAS

ATLAS (*Automatically Tuned Linear Algebra Software*) es un enfoque para la genración y optimización automática de *software* numérico para procesadores con uso avanzado de la jerarquía de memoria y de las unidades funcionales con *pipelining*. En la versión actual (3.0Beta), se han concentrado los esfuerzos en la afinación de las rutinas de la biblioteca **BLAS 3**[22]. Aún cuando se han incluido los tres niveles de **BLAS** y algunas rutinas de **LAPACK**.

3.7.1. Capacidades

Las capacidades de **ATLAS** son las mismas de **BLAS**, es decir, operaciones básicas con matrices y vectores densos, además incluye las rutinas de **LAPACK** relacionadas con la descomposiciones LU y factorización de Cholesky.

Una de las características remarcables de esta biblioteca es que logra, en la mayoría de las plataformas, un rendimiento mayor que las mismas rutinas provistas en bibliotecas comerciales.

3.7.2. Lenguajes

Las rutinas de **ATLAS** pueden ser invocadas desde **Fortran** y **C**, y en este último tiene optimizaciones para acceso a los elementos de las matrices tanto por columnas como por filas.

3.7.3. Requerimientos de *hardware* y *software*

ATLAS requiere compiladores de **Fortran** y **C**, en el caso de Linux las pruebas de compilación y rendimiento han sido hechas con los compiladores de GNU.

3.7.4. Portatilidad

La principal meta en el desarrollo de la biblioteca **ATLAS** es la portatilidad, es por ello que está disponible tanto el código fuente, como versiones precompiladas para casi todas las plataformas existentes, y se hace especial énfasis en los procesadores utilizados frecuentemente en los *Cluster* de Linux: Pentium-Pro y Pentium II y III.

3.7.5. Documentación

Tanto el software, incluido el código fuente y las versiones precompiladas, como la documentación están disponibles en la pagina de **ATLAS**[14].

3.8. BlockSolve

BlockSolve95 es un *software* paralelo escalable diseñado para solucionar sistemas lineales esparcidos que provienen de modelos físicos con múltiples grados de libertad en cada nodo. **BlockSolve95** también es razonablemente eficiente al resolver problemas con un solo grado de libertad

[22] http://www.netlib.org/blas/blas3-paper.ps

en cada nodo, tal como el problema tri-dimensional de Poisson. **BlockSolve95** es de propósitos generales, y sólo requiere que las matrices sean esparcidas y de estructura simétrica (aunque no necesariamente simétrica elemento a elemento).

3.8.1. Capacidades

Todos los cálculos hechos por **BlockSolve95** par la solución de sistemas esparcidos son realizados en paralelo, y las rutinas están configuradas para manejar matrices que ocupen por completo la memoria disponible en la plataforma paralela, con esto almacenar matrices del orden de 1×10^8 elementos diferentes de cero en una memoria tipica de 1,2GB, es decir una máquina con 8 nodos y un poco más de 256MB de RAM disponible en cada nodo.

Las operaciones básicas de multiplicación de matrices y matrices por vectores son realizadas con rutinas de los niveles 2 y 3 de **BLAS**

3.8.2. Lenguajes

BlockSolve95 puede ser utilizada tanto en **Fortran** como **C**.

3.8.3. Requerimientos de *hardware* y *software*

BlockSolve95 requiere de una instalación de **MPI**, los autores recomiendan la utilización de la distribución **MPICH**[23]. También es necesaria una instalación de la biblioteca **BLAS**.

3.8.4. Portatilidad

BlockSolve95 ha sido compilada y probada en una amplia gama de plataformas entre las que destacan: IBM SP, Intel DELTA, y *cluster* de Sun, RS/6000, y SGI.

3.8.5. Aplicaciones

La biblioteca **BlockSolve95** ha sido utilizada con éxito en las siguientes aplicaciones:

- Simulación de clima, diseño y simulación de vehiculos aero-espaciales.
- Modelado de superconductores a altas temperaturas (*http://www-unix.mcs.anl.gov/sumaa3d/Examples/super.html*)
- Simulación de sistemas comerciales de combustión por **Nalco Fuel Tech** (*http://www-unix.mcs.anl.gov/sumaa3d/Examples/nalco/nalco.html*)
- Análisis de critales piezo-eléctricos por **Motorola** (*http://www-unix.mcs.anl.gov/sumaa3d/Examples/crystal.html*)

3.8.6. Documentación

La documentación de **BlockSolve95** está disponible en la página web[15].

[23] http://www-unix.mcs.anl.gov/mpi/mpich/

4. La matriz del *software* de dominio público para álgebra lineal en Internet

En la tabla (2) se presenta el *software* disponible en Internet y sus capacidades generales. Esta matriz aparece en un reporte presentado por Jack Dongarra de la Universidad de Tennessee y que está disponible en el Web de NetLib[16]. J. Dongarra hace esta clasificación con orientación hacia el alto rendimiento. En la página se proporcionan los enlaces a todas las biblotecas mencionadas.

Las columnas de la tabla (2) contiene la siguiente información:

- Paquete: El nombre de la biblioteca
- Tipo: El tipo de variable que soporta
  - Real: punto flotante real
  - Comp: variable compleja en punto flotante
- Lenguaje: Lenguajes desde los cuales se pueden utilizar las rutinas.
- Modo:
  - Seq: Código secuencial
  - Dist: Código paralelo (M=MPI, P=PVM)
- Denso: resuelve problemas que involucran matrices densas
- E. Directo: Resuelve por métodos directos problemas que involucran matrices esparcidas
  - SPD: Simétricas
  - Gen: Generales
- E. Iterativo: Resuelve por métodos iterativos problemas que involucran matrices esparcidas
  - SPD: Simétricas
  - Gen: Generales
- E. Auto: Resuelve problemas de autovalores que involucran matrices esparcidas
  - Sim: Simétricas
  - Gen: Generales

Paquete	Tipo		Lenguaje			Modo		Denso	E Directo		E Iterativo		E Auto	
	Real	Comp.	F77	C	C++	Seq	Dist		SPD	Gen	SPD	Gen	Sim	Gen
Atlas	X	X	X	X		X		X						
BLAS	X	X	X	X		X		X						
MTL	X				X	X		X						
NIST S-BLAS	X	X	X	X		X			X	X	X	X		
SparseLib++	X	X		X	X	X			X	X	X	X		
PLAPACK	X	X	X	X			M	X						
PRISM	X		X			X	M	X						
ScaLAPACK	X	X	X	X			M/P	X						
MA28	X		X			X			X	X				
MFACT	X			X		X			X					
MP_SOLVE	X	X	X				M			X				
PSPASES	X		X	X			M		X					
SPARSE	X	X		X		X			X	X				
SPARSEQR	X			X	X	X			X	X				
SupeLU	X	X	X	X		X			X	X				
UMFPACK	X	X	X			X			X	X				
Y12M	X		X			X			X	X				
BILUM	X		X			X					X	X		
BlockSolve95	X		X	X	X		M				X	X		
BPKIT	X		X	X							P	P		
IML++	X		X	X	X	X					X	X		
ISIS++	X				X		M				X	X		
ITPACK	X		X			X					X	X		
LASPack	X			X		X					X	X		
PARPRE	X			X			M				P	P		
PCG	X		X	X	X		P				X			
PETSc	X	X	X	X		X	M				X	X		
PIM	X	X	X			X	M/P				X	X		
P-SparsLIB	X		X				M					X		
QMRPACK	X	X	X			X					X	X	X	X
SPLIB	X		X			X					X	X		
SPOOLES	X	X		X		X	M		X	X	X	X		
Templates	X		X	X		X					X	X		
LASO	X		X			X							X	
P_ARPACK	X	X	X	X	X	X	M/P						X	X
PLANSO	X		X			X	M						X	
TRLAN	X		X			X	M						X	

Cuadro 2: Software gratis en Internet para álgebra lineal

Apéndice A

Acceso a los elementos de un arreglo.

Este ejemplo sencillo muestra que una revisión detenida de la forma como se accede a los datos almacenados en memoria puede llevar a una ganancia tremenda en tiempo de ejecución.

Supongamos que tenemos 3 matrices cuadradas de dimensión 1024, lo que da un total de unos 22MB en memoria RAM. Supongamos que la memoria disponible al momento de la ejecución es de unos 7MB lo que quiere decir que caben en memoria solo 64 columnas de cada matriz. Para realizar la operación matricial $C = AB + C$ utilizamos el código (fortran):

```
      DO 30 I=1,N
        DO 20 J=1,N
          DO 10 K=1,N
            C(I,J)=A(I,K)*B(K,J)+C(I,J)
10        CONTINUE
20      CONTINUE
30    CONTINUE
```

Este algoritmo comete 16 fallos de página cada vez que pide un elemento de A, es decir que debe pedir este elemento desde el *swap* en el disco duro. Dado al número de elementos de D, esto lleva a unos 16 millones de fallos de página, a un promedio de 0,5 segundos cada uno, algo así como 93 días en problemas de entrada/salida (I/O).

Si modificamos el algoritmo anterior de la siguiente manera

```
      DO 30 J=1,N
        DO 20 K=1,N
          DO 10 I=1,N
            C(I,J)=A(I,K)*B(K,J)+C(I,J)
10        CONTINUE
20      CONTINUE
30    CONTINUE
```

En este caso el problema está con el índice J, produciendose sólo 1024×16 fallos de página, lo que nos dá unas 2,25 horas de tiempo de I/O.

La optimización utilizadas en algunas bibliotecas como **BLAS** involucra un particionamiento de los lazos para evitar en lo posible los fallos de página, esto se hace como sigue:

```
      DO 40 J=1,N,NB
        DO 30 K = 1,N,NB
          DO 20 JJ = J,J+NB-1
            DO 10 KK = K,K+NB-1
              C(:,JJ) = C(:,JJ)+A(:,KK)*B(KK,JJ)
10          CONTINUE
20        CONTINUE
30      CONTINUE
40    CONTINUE
```

Aqui NB es el número de elemento que cabe en una página, lo cual se determina automáticamente al momento de compilar la biblioteca. Este algoritmo comete solamente 96 fallos de página, es decir 70 segundos de tiempo de I/O.

Referencias

[1] Rubin H. Landau y Manuel J. Páez. *Computational Physics*. John Wiley & Sons, INC., 1997.

[2] IBM. *Optimization and Tuning Guide for Fortran, C and C++. AIX Version 3.2 for RISC System/6000*, 1ra edición edition, 1993.

[3] Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst. *Numerical Linear Algebra for High-Performace Computers*. SIAM, 1998.

[4] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programing with MPI*. The MIT Press, 1996.

[5] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The C omplete Reference*. The MIT Press, 1996.

[6] Documentación de scalapack en la web. *http://www.netlib.org/scalapack/html/scalapack_doc.html*.

[7] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[8] Karin Remington and Roldan Pozo. The nist sparse blas (v. 0.9) sparse matrix computational kernels. *http://math.nist.gov/~KRemington/fspblas/*, Enero 1997.

[9] Karin Remington and Roldan Pozo. The nist sparse blas (v. 0.9) sparse matrix computational kernels. *http://math.nist.gov/spblas/*, Enero 1997.

[10] Parallel arpack home page. *http://www.caam.rice.edu/~kristyn/parpack_home.html*, 1996.

[11] Spooles home page. *http://www.netlib.org/linalg/spooles/spooles.2.2.html*, 1999.

[12] Superlu. *http://www.nersc.gov/~xiaoye/SuperLU/*, Sep. 1999.

[13] Mathematics and Computer Science Division of Argonne National Laboratory. Superlu. *http://www-fp.mcs.anl.gov/petsc/*, May 2000.

[14] R. Clint Whaley and Jack J. Dongarra Automatically Tuned Linear Algebra Software. *http://www.netlib.org/atlas/*

[15] SUMAA3D Project, Argonne National Laboratory BlockSolve95. *http://www-unix.mcs.anl.gov/sumaa3d/BlockSolve/index.html*

[16] Jack J. Dongarra. Freely available software for linear algebra on the web. *http://www.netlib.org/utk/people/JackDongarra/la-sw.html*, Noviembre 1999.