UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACIÓN

Implementación de OpenMp (Open Multi Processing) en deMon2k 6.0.2

PROYECTO FINAL

QUE PARA OBTENER EL GRADO DE:

ESPECIALISTA EN CÓMPUTO DE ALTO RENDIMIENTO

P R E S E N T A :

REYNA ELIZABETH CABALLERO CRUZ

DIRECTOR DE PROYECTO:

DR. JOSÉ JESÚS CARLOS QUINTANAR SIERRA

Ciudad Universitaria, CD. MX., a 3 de octubre de 2020

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años el panorama del cómputo de alto rendimiento ha crecido debido a la disponibilidad de microprocesadores [1] económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran. Debido a esto, se ha incrementado el interés en la computación paralela, cuyo objetivo principal es reducir el tiempo de procesamiento. pero considerando el ahorro de la energía y diseñando nuevas técnicas de enfriamiento para evitar el sobre calentamiento de los circuitos.

Cuando hablamos de computación paralela nos referimos a un número de unidades de cómputo (cores) resuelven un problema de manera cooperativa [3].

Las alternativas para realizar programas en paralelo [2] son variadas y todas ellas son competitivas. La elección de la técnica de programación no es sencilla por las ventajas que cada alternativa presenta. Elegir la mejor técnica de programar en paralelo dependerá del problema a resolverse, el equipo de cómputo y la aplicación de la técnica apropiada del lenguaje de programación.

El presente trabajo esta orientado a implementar OpenMP en el software deMon2k utilizado por la comunidad científica para el cálculo de propiedad físico-químicas de átomos y moléculas.[1]

[1] http://www.demon-software.com/public_html/publications.html#PhD%20Theses

Capítulo 2

Antecedentes

2.1. Delimitación y planteamiento del problema

OpenMP (Open Multi-Processing) es una interfaz de programación de aplicaciones de memoria compartida que permite programación en paralelo. Puede utilizarse por una amplia gama de arquitecturas de memoria compartida, desde computadoras que cuentan con procesadores multinúcleo, procesadores multithreading o incluso también puede usarse para crear programas en computadoras que cuenta con un solo procesador. OpenMP no es un lenguaje de programación, son directivas que al ser agregadas a programas en lenguaje Fortran, C o C++, indican cómo se repartirá el trabajo que será ejecutado en los procesadores o núcleos, así como el acceso a la memoria compartida. Por otra parte deMon2k (density of Montréal) es un software que apareció en 1992, se utiliza para calcular de primeros principios propiedades físico-químicas de átomos y moléculas aplicando la teoría del funcional de la densidad (DFT), empleando una combinación lineal de orbitales de tipo gaussiano para resolver con métodos de autoconsistencia las ecuaciones DFT Kohn-Sham (KS). Este trabajo esta enfocado a analizar, diseñar e implementar directivas de OpenMP que permita mejorar el funcionamiento y optimizar los cálculos efectuados con deMon2K.

La motivación del trabajo surge porque actualmente los sistemas de memoria distribuida pueden contar hasta con 64 cores o 128 hilos[1] cores, deMon2k tiene ya integrada la Interfaz de Paso de Mensajes (Message Passing Interface) entonces, si se utiliza OpenMP en sistemas distribuidos podemos mejorar la eficiencia computacional.

[1] https://www.amd.com/es/products/cpu/amd-epyc-7742

2.2. Justificación

En este trabajo se analizará el funcionamiento del programa deMon2k y se integrarán programa(s) con OpenMp. La primera parte del trabajo se examinarán los programas que conforman deMon2K con el fin de diseñar las acciones que permitan crear y/o modificar los programas que cumplan con el objetivo general de este trabajo. El beneficio que se busca es proporcionarles a los usuarios deMon2k, una versión que utilice de manera óptima los recursos computacionales, es decir dependiendo de donde se ejecuten. Las etapas que conformaran la implementación de OpenMP estarán integradas por: planeación, análisis, diseño e implementación.

2.3. Objetivos

Objetivo general

- Establecer la implementación de OpenMP en el programa deMon2k versión 6.0.2.

Objetivo específicos

- Identificar la comunicación de procesos que conforman al programa.
- Determinar las directivas de OpenMP para eficientizar el problema.

2.4. Marco teórico conceptual

La tecnología actualmente cuenta con procesadores rápidos, de 1986 a 2002 se incremento un 50 % el rendimiento en los microprocesadores por año Pacheco [4]; derivado por el crecimiento de número transistores del circuito integrado, cada año se aumentaba en ese mismo porcentaje Petersen et al., [5], como lo predijo Gordon Moore (uno de los fundadores de Intel) en 1965 conocida como ley de Moore: *el incremento de la velocidad de los procesadores deberá ser duplicada cada dos años.* Sin embargo, en abril de 2015 Gordon Moore reapareció en una entrevista [2] para decir que su ley estaba más que muerta ya que las limitaciones técnicas de la fabricación de los microprocesadores actuales hacen que sus

[2] https://www.youtube.com/watch?v=ylgk3HEyZ_g

arquitecturas tengan un techo de evolución. Será entre el periodo 2015 y 2020 donde la Ley de Moore mostrará su validez o será refutada al 100 %.

Aunque difícil de comprender por su aparente complejidad, la Ley de Moore es uno de esos pilares básicos para entender como la tecnología ha permitido hacer auténticas maravillas, en la grafica 2.1

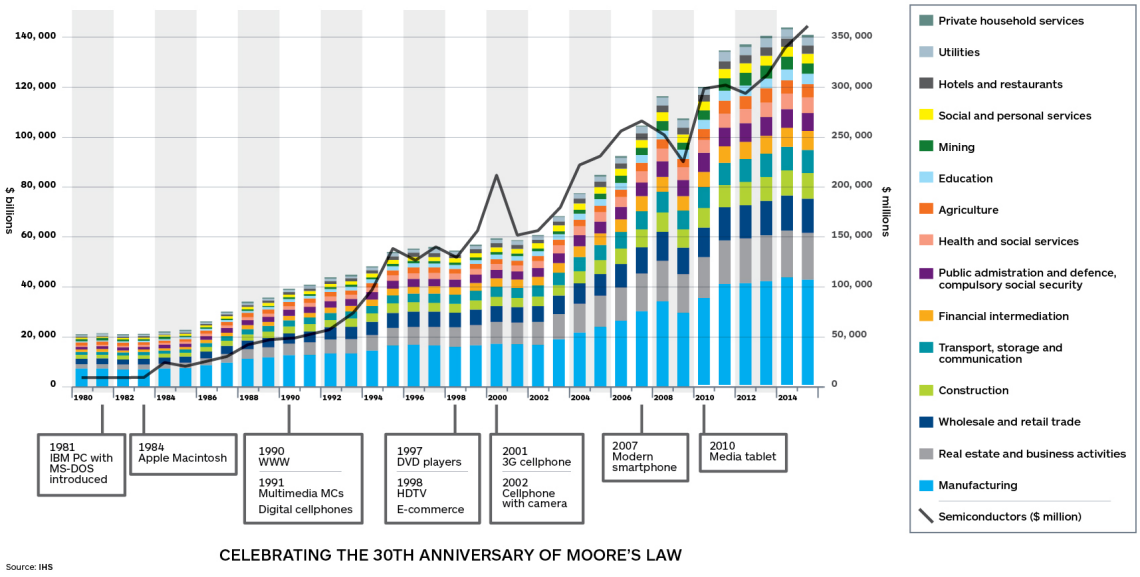

Figura 2.1: La gráfica muestra el comportamiento de la Ley de Moore a partir del año 1981 hasta el 2015. [10]

2.4.1. deMon2K

Es un programa desarrollado por grupos de científicos líderes en su área alrededor del mundo *(http://www.demon-software.com)* que resuelve las ecuaciones de Kohn-Sham dentro de la aproximación de combinación lineal de orbitales gausianos aplicando el ajuste variacional de la densidad electrónica.

Entre sus capacidades incluye:

- Optimización de geometrías y búsqueda de estados de transición.
- Simulaciones de dinámica molecular.

- DFT dependiente del tiempo.
- Paralización basada en MPI.
- Pseudo-potenciales.
- Propiedades: hyper-polarizabilidades, NMR, IR/Raman, Termodinámica, Reactividad, etc.

Es interesante la evolución que deMon2K se muestra en la figura 2.2.

2.4.2. OpenMP

OpenMP es una Interfaz de Programación de Aplicaciones (API) cuyas características se basan en esfuerzos anteriores para facilitar la programación paralela de memoria compartida, Chapman et al. [7]. Las siglas MP denotan "multiprocessing" (multiproceso) lo cual es un sinónimo de programación paralela de memoria compartida. OpenMP es un acuerdo alcanzado entre los miembros de la Architecture Review Board (ARB) para dar un enfoque portátil, fácil de usar y eficaz a la programación paralela de memoria compartida. Al contrario de lo que se podría pensar OpenMP no es un lenguaje de programación nuevo; por el contrario es la notación que se puede agregar a un programa secuencial en Fortran, C o C++, a las cuales se le denomina directivas o pragmas que son instrucciones pre-procesador. Una implementación adecuada de OpenMP en un programa permitirá a las aplicaciones beneficiarse de la memoria compartida de las arquitecturas paralelas. En ocasiones con pocas modificaciones al código se convierte una aplicación serial en una paralela; en la práctica muchas aplicaciones tiene un cierto grado de paralelismo que debe ser explotado. Como menciona Chandra et al.[8], todos los proveedores de computadoras de memoria compartida de alto desempeño soportan la funcionalidad de OpenMP, pero la portabilidad de aplicaciones ha sido casi imposible de alcanzar. Desde un punto de vista general, una aplicación serial se convierte en paralelo simplemente incluyendo una directiva de OpenMP. Estas directivas van desde constructores de hilos hasta la sincronización de los mismos para el acceso a los datos compartidos. Pero un punto muy importante es que en la directiva se indica el tipo de reparto de las cargas de trabajo.

OpenMp funciona con directivas que habilitaran la ejecución en paralelo, se crearán y/o lanzarán hilos según las características que se le indiquen a la directiva, estas directivas

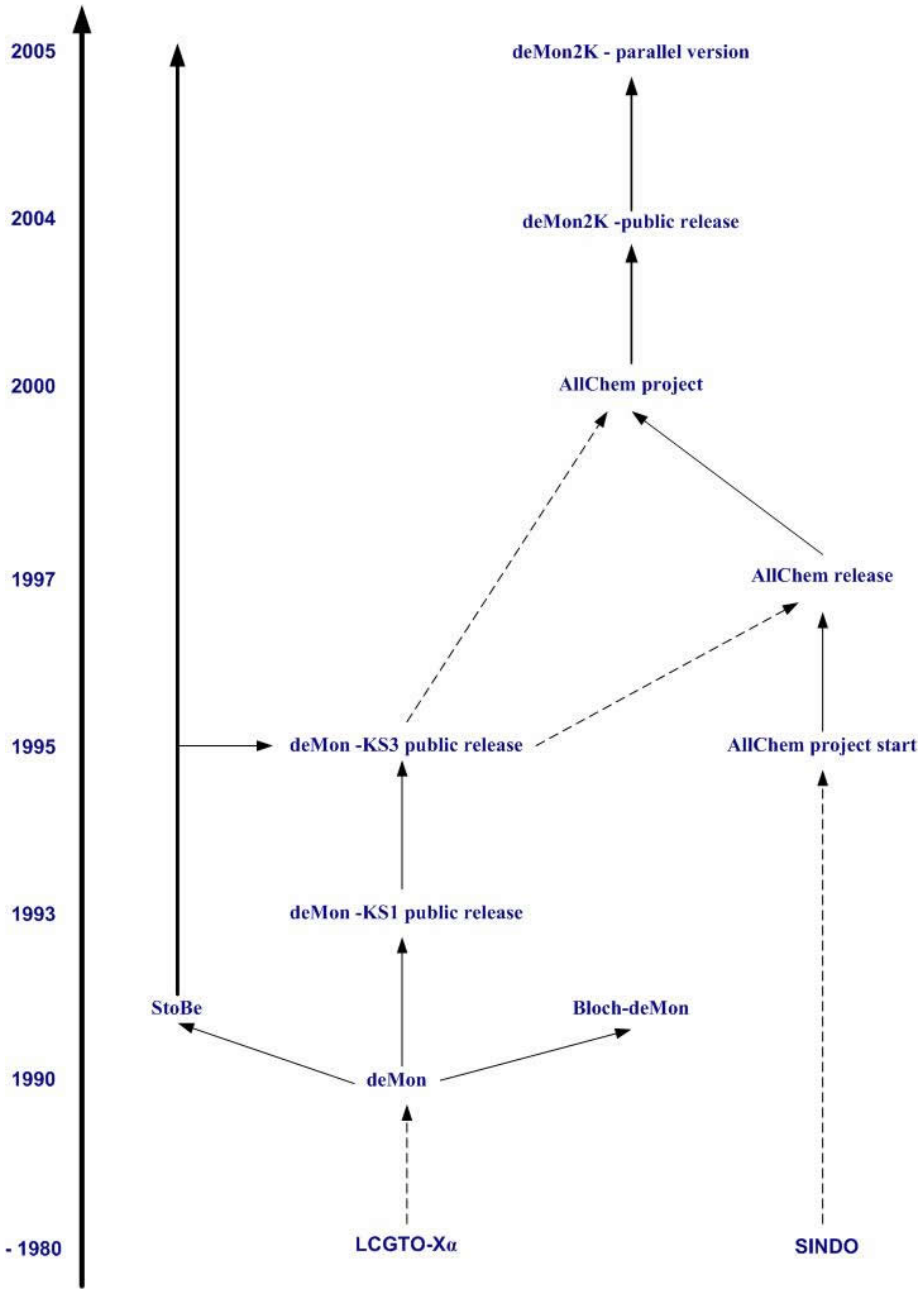

Figura 2.2: En la figura se muestra el inicio del deMon2K, las transformaciones hasta llegar a la versión actual del código. [6]

deben estar sincronizadas en un bloque de código, y al final de ella habrá una barrera para la sincronización de los diferentes hilos, salvo que implícitamente se indique lo contrario. Para comenzar a usarlo se debe :

Lenguaje C/C++

```
# pragma omp <directiva> [clausula [ , ...] ...]
```

Fortran

```
!$OMP  PARALLEL <directiva> [clausulas]
       bloque de codigo
!$OMP END <directiva>
```

A las *directivas*, también se le conoce como constructores, *parallel:* Esta indica que la parte de código que la comprende puede ser ejecutada por varios hilos. *for o do* optimiza los ciclos.

Las Cláusulas

- shared:Los datos de la región paralela son compartidos, lo que significa que son visibles y accesibles por todos los hilos. Por definición, todas las variables que trabajan en la región paralela son compartidas excepto el contador de iteraciones.

- private:Los datos de la región paralela son privados para cada hilo„ lo que significa que cada thread tendrá una copia local que la usará como variable temporal. Una variable privada no es inicializada y tampoco se mantiene fuera de la región paralela. Por definición, el contador de iteraciones en OpenMP es privado.

- section Indica secciones que pueden ejecutarse en paralelo pero por un único hilo.

- single: La parte de código que define esta directiva, sólo se puede ejecutar un único hilo de todos los lanzados, y no tiene que ser obligatoriamente el hilo padre.

- master: La parte de código definida, sólo se puede ejecutar por el hilo padre.

- critical: Sólo un hilo puede estar en esta sección.

- atomic: Se utiliza cuando la operación atañe a sólo una posición de memoria, y tiene que ser actualizada sólo por un hilo simultáneamente. Operaciones tipo x++ o –x son las que usan esta cláusula.

- flush: Esta directiva resuelve la consistencia, al exportar a todos los hilos un valor modificado de una variable que ha realizado otro hilo en el procesamiento paralelo.
- default:Permite al programador que todas las variables de la región paralela sean shared o no para C/C++, o shared, firstprivate, private, o none para Fortran. La opción none fuerza al programador a declarar de que tipo será cada variable en la región paralela.
- firstprivate:Como private pero se inicializa con el valor original.
- reduction:Hace una operación(suma, resta, multiplicación, etc) privada en cada iteración y al final suma cada resultado.

Funciones

- omp_set_num_threads: Fija el número de hilos simultáneos.
- omp_get_num_threads: Devuelve el número de hilos en ejecución.
- omp_get_max_threads: Devuelve el número máximo de hilos que lanzará nuestro programa en las zonas paralelas. Es muy útil para reservar memoria para cada hilo.
- omp_get_thread_num: Devuelve el número del thread dentro del equipo (valor entre 0 y omp_get_num_threads()-1)
- omp_get_num_procs: Devuelve en número de procesadores de nuestro ordenador o disponibles (para sistemas virtuales).
- omp_set_dynamic: Valor booleano que nos permite especificar si queremos que el número de hilos crezca y decrezca dinámicamente.

Variables de Ambiente

Se debe compilar con la opción `-fopenmp` con el compilador de gcc y maneja las siguientes variables de ambiente:

- OMP_NUM_THREADS=*NUM*. Establece el número de hebras que usa una región paralela

- OMP_SCHEDULE type[,chunk]. Establece el tipo planificador y tamaño de las partes. Los tipos validos para el planificador OpenMP son static, dynamic, guied, o auto. Las partes se expresan con un entero positivo.
- OMP_DYNAMIC dynamic. Establece el ajuste dinámico de las hebras que usa la región paralela. Los valores validos para dynamic son true o false.
- OMP_NESTED nested. Habilitado o deshabilitado el anidamiento paralelo. Los valores validos para nested es true or false.
- OMP_STACKSIZE size. Establece stacksize-var (VIC) que especifica el tamaño de la pila de hebras que crea la implementación de OpenMP. Los valores para size (entero positivo) son size, sizeB, sizeK, sizeM, sizeG. Si la unidad B, K, M o G no se especifica, el tamaño se mide en kilobytes (K).
- OMP_WAIT_POLICY policy. Controla el comportamiento esperado de la espera de las hebras. Los valores validos para policy es activo (las hebras consumen ciclos de procesamiento mientras esperan) y pasivo.
- OMP_MAX_ACTIVE_LEVELS levels. Controla el máximo número de anidamientos activos en una región paralela.
- OMP_THREAD_LIMIT limit. Indica el máximo número de hebras que participan en un programa OpenMP.

Para ejecutar se habilitará la variable de ambiente `OPENBLAS_NUM_THREADS` en donde se indica el número de cores que el programa utilizará.

2.4.3. Basic Linear Algebra Subprograms (BLAS)

Las bibliotecas *BLAS (Subprogramas de Álgebra Lineal Básica)* son rutinas que proporcionan bloques de construcción estándar para realizar operaciones básicas de vectores y matrices. Las BLAS de Nivel 1 realizan operaciones escalares y vectoriales, las BLAS de Nivel 2 realizan operaciones de vectores matriciales y las BLAS de Nivel 3 realizan operaciones de matriz-matriz. Debido a que los BLAS son eficientes, portátiles y están ampliamente disponibles, se usan comúnmente en el desarrollo de software de álgebra lineal de alta calidad.

2.4.4. OpenBLAS

OpenBLAS es una biblioteca BLAS optimizada basada en la versión BSD del proyecto GotoBLAS, creado por el Centro de Computación Avanzada de Texas (TACC), este producto ya no está en desarrollo activo por TACC, pero está disponible para que la comunidad lo use, estudie y extienda. OpenBLAS utiliza algoritmos y técnicas de manejo de memoria para un rendimiento óptimo de las rutinas BLAS, para las características de arquitectura en microprocesadores y técnicas de comunicación interprocesador. Además, los controles NUMA mejoran la ejecución de subprocesos múltiples de las rutinas BLAS en el nodo.

La biblioteca incluye las siguientes características:

- Configuraciones para una variedad de plataformas de hardware.
- Incorporación de características a nivel de arquitectura de conjunto de instrucciones (ISA; Instruction Set Architecture).
- Implementación de controles de acceso a memoria no uniforme (NUMA; Non-Uniform Memory Access) para asegurar la mejor afinidad de procesos y la política de memoria.
- Detección dinámica de múltiples componentes de arquitectura, que pueden incluirse en un solo binario (para distribuciones binarias).

Capítulo 3

Metodología de investigación

Actualmente no existe una versión de deMon2k con OpenMP, por el ello el proyecto comenzará identificado aspectos que conforman el funcionamiento de deMon2k para posteriormente insertar las directivas de OpenMp que lo paralelicen, así como también se realizarán pruebas en serie y en paralelo para analizar métricas de rendimiento.

3.1. Análisis de deMon2k

Es un programa desarrollado por un grupo de investigadores, debido a que sus desarrolladores lo utilizan para sus investigaciones esta en un continúa mejora. La versión 6.0.2 esta conformado por 32 directorios y 2872 programas en Lenguaje Fortran, cada directorio contiene un tema específico del programa. El código del programa esta integrado por archivo y directorios agrupados y comprimidos en un archivo denomido `deMon.x.y.z`, como se muestra en la Figura 3.1. Los números `x, y y z` se refieren a la versión, versión y número de revisión de su fuente deMon2k.

Para la instalación se debe descomprimir y desagrupar el archivo deMon.x.y.z, Lo siguiente que debe realizar es editar el archivo `.default-version`, en este archivo indicaremos la versión y el compilador que se utilizará, Figura 3.2.

Enseguida se deberá ejecutar el programa en bourne shell denominado **CREX**, este programa se modificó para considerar una nueva opción de compilación, en la figura 3.4. El programa CREX es el encargado de instalar el programa de deMon con las características deseadas.

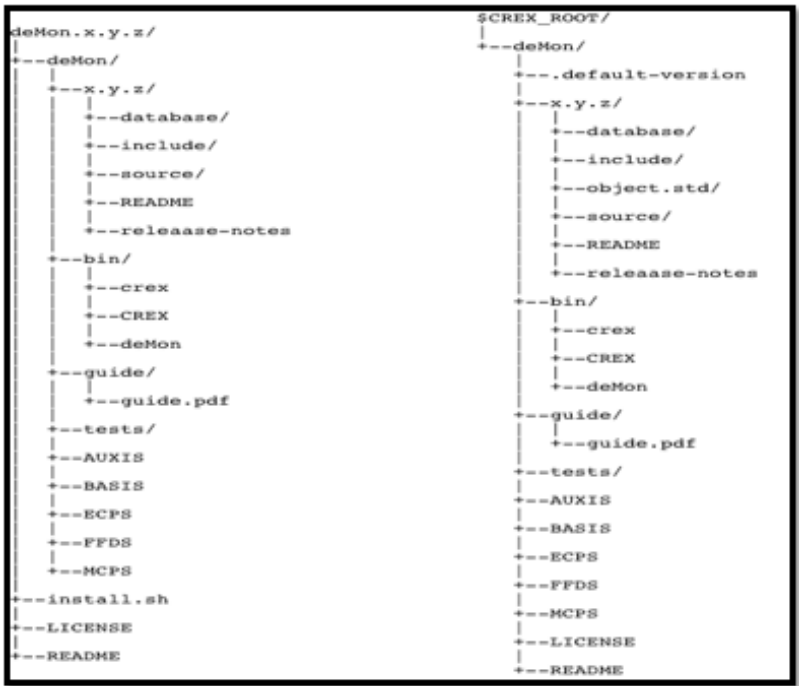

Figura 3.1: Árbol de directorios de la versión deMon2k, en la parte izquierda son los archivos que se encuentran en la versión agrupado y comprimida. En la parte derecha son los archivos ya desagrupados y descomprimos.

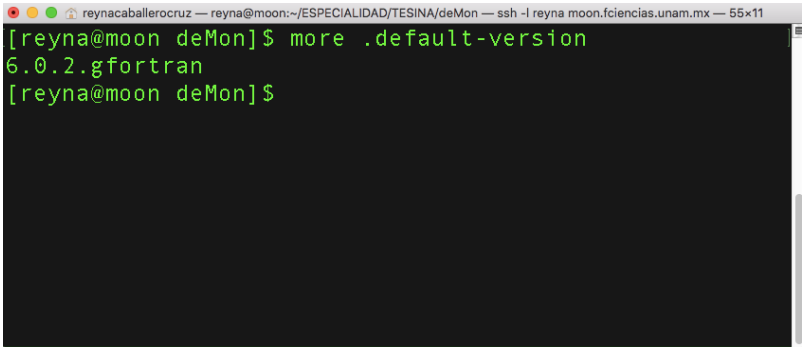

Figura 3.2: Se muestra el archivo `.default-version.` que obtiene información que el programa de instalación utilizará.

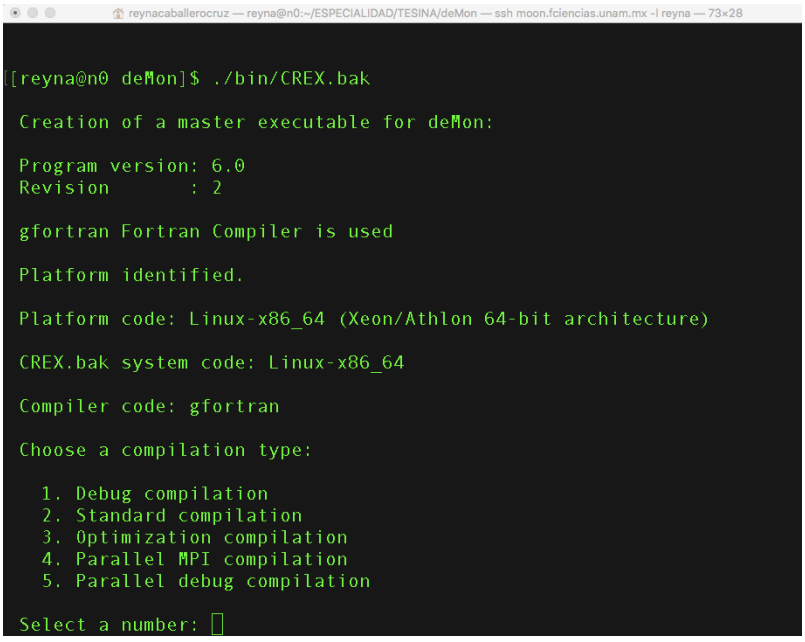

Figura 3.3: Se muestra el programa *CREX* donde aparecen las opciones de instalación.

Los parámetros iniciales que utilizará el programa CREX lo tomará de un archivo `Linux-x86_64.std.` el nombre del archivo se describe de la siguiente forma:

Linux	Sistema Operativo Linux
x86	Arquitectura de 64 bits
std	Versión serial
gfortran	Compilador que utilizará

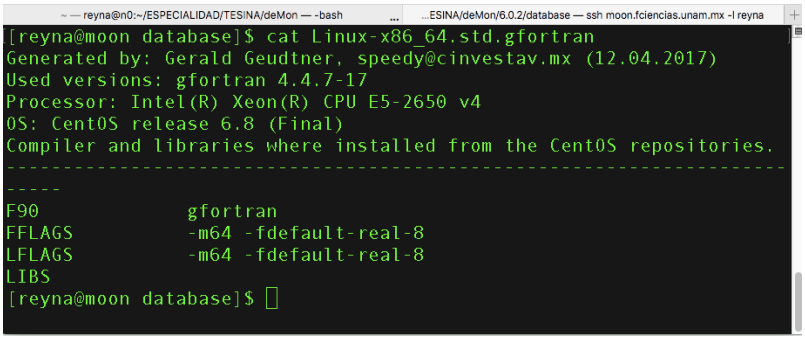

Figura 3.4: Parámetros que utiliza el programa CREX.

3.2. Ambiente de trabajo *deMon2K*

Una vez instalado deMon2k tiene algunos límites pre-establecidos para la memoria y los requisitos de espacio en disco, la cantidad de átomos y la cantidad de funciones básicas y auxiliares. La figura 3.5 muestra los límites del programa deMon2K.

Se realizaron pruebas del flujo de datos, la tabla 3.1 muestra los elementos de una ejecución del programa deMon2K en forma serial.

Como resultado del flujo de datos se comenzó a trabajar con los programas que se encuentran en el directorio `mathlib` que reúne las operaciones matemáticas principales del código, este directorio cuenta con 65 programas escritos en fortran, uno de los principales es el programa `jacobi.f` este programa calcula valores propios y vectores propios de una matriz simétrica real utilizando el método de Jacobi, el resto de los programas realizar operaciones matriciales como son *suma vectores, multiplicación escalar, producto escalar combinaciones lineales y multiplicación* utilizando las bibliotecas BLAS.

Parameter	Setting	Description
MAXDISK	51200	Maximum disk size for scratch files in Mbytes
MAXRAM	2048	Maximum RAM size for program kernel in Mbytes
PRORAM	256	RAM size reserved for program data in Mbytes
MAXATMM	15000	Maximum number of MM atoms
MAXATOM	1000	Maximum number of QM atoms
MAXAUX	20000	Maximum number of auxiliary functions
MAXAUXSET	10000	Maximum number of auxiliary function sets
MAXAUXSHL	10000	Maximum number of auxiliary function shells
MAXCON	21	Maximum degree of contraction
MAXCUBE	500000	Maximum number of (cube) embedding points
MAXECPGTO	10000	Maximum number of ECP Gaussian functions
MAXECPSHL	3000	Maximum number of ECP shells
MAXGTO	20000	Maximum number of primitive Gaussian functions
MAXLAUX	6	Maximum L quantum number for auxiliary functions
MAXLBAS	6	Maximum L quantum number for basis functions
MAXLECP	5	Maximum L quantum number for ECP functions
MAXLINK	4	Maximum number of molecular mechanics links
MAXLMCP	3	Maximum L quantum number for MCP functions
MAXLSUM	8	Maximum L quantum number for double asymptotic ERIs
MAXMCPGTO	10000	Maximum number of MCP Gaussian functions
MAXMCPSHL	3000	Maximum number of MCP shells
MAXMCPSTO	6000	Maximum number of contracted MCP (STO) orbitals
MAXPAT	10	Maximum number of geometry pattern
MAXSHL	5000	Maximum number of (STO) orbital shells
MAXSTO	10000	Maximum number of contracted (STO) orbitals
MAXTBSTO	10000	Maximum number of tight-binding (STO) orbitals

Figura 3.5: Límites establecidos del programa deMon2K

INITIALIZATION
INPUT
DIAGONALIZATION
SYMMETRY ANALYSIS
START GUESS
MATRIX MULTIPLICATION
CORE INTEGRALS
ORTHOGONALIZATION MATRIX
AUXILIARY FUNCTION FIT
CORE HAMILTONIAN
ECP INTEGRALS
SCF ITERATION
NULLSPACE
NEAR J CALCULATION
GRID INITIALIZATION
ADAPTIVE GRID GENERATION
GRID RETRIEVE
XC POTENTIAL
NEAR ERI CALCULATION
ELECTRONIC STRUCTURE ANALYSIS

Tabla 3.1: Elementos reportados en una ejecución del programa deMon2k

3.3. Implementación de OpenMp

Después del análisis se realizarán las siguientes acciones

1. Modificar el programa CREX incluyendo **OpenMP**.
2. Realizar el archivo con los parámetros para el programa CREX.
3. Incluir directivas de **OpenMP** en el programa `jacobi.f`.
4. Implementar **OpenBLAS.**

3.3.1. Programa CREX incluyendo OpenMP.

El programa CREX es un programa escrito en bourne again shell cuenta con 850 líneas, se agregaron las líneas que corresponden con la opción de compilación incluyendo **OpenMp**, se incorporaron las opciones **5 y 6**, Parallel OpenMP compilation y Parallel MPI+OpenMP compilation respectivamente, el resultado se ve en la figura 3.6.

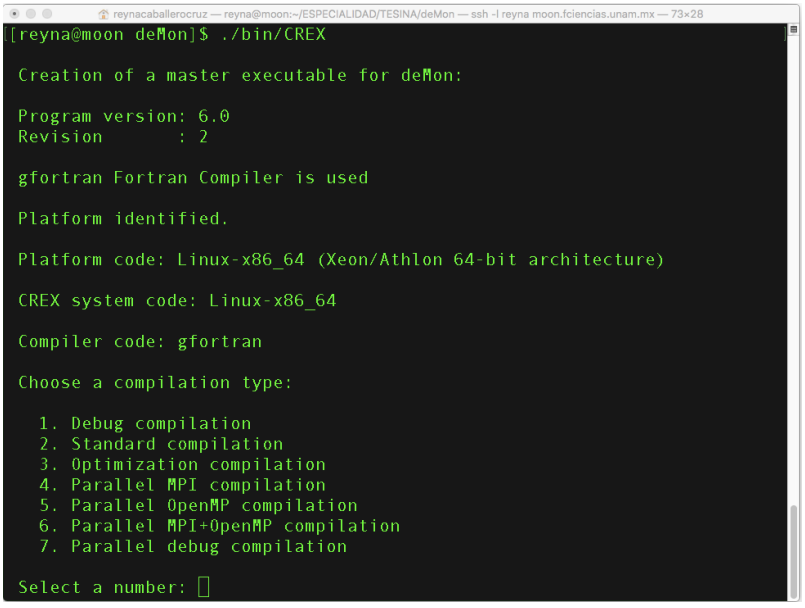

Figura 3.6: Programa CREX con nuevas opciones.

3.3.2. Archivo con los parámetros

Continuando con el formato establecido por los desarrolladores de deMon2k se realizó el archivo `Linux-x86_64.omp.gfortran` en este archivo se indica el uso de -**OpenMP** y **OpenBLAS,** como se observa en la figura 3.7.

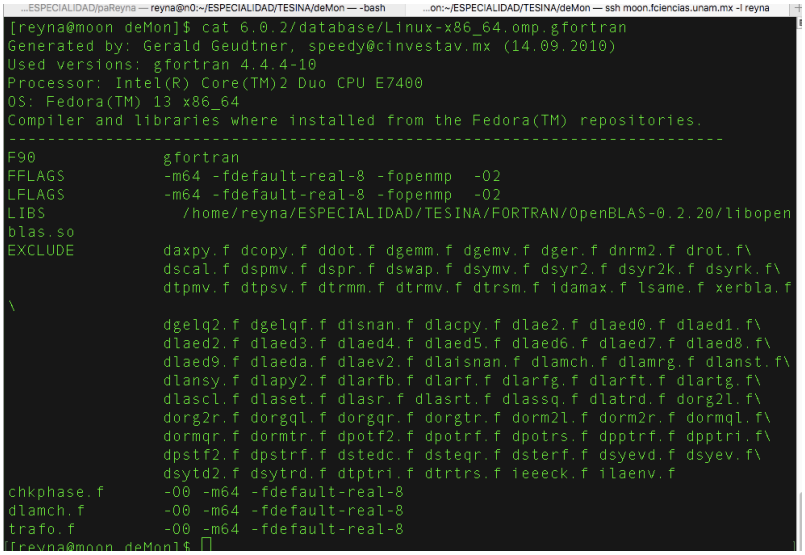

Figura 3.7: Archivo con las especificaciones de compilación con OpenMP.

3.3.3. Implementacion de OpenBLAS

Al inicio de este trabajo se consideró optimizar las bibliotecas BLAS, pero a lo largo del trabajo se encontró que ya existía este proyecto, por lo cual se toma la decisión de analizar su funcionamiento y verificar que si funciona correctamente para después utilizarlo con deMon2k, los pasos que se realizaron fueron:

1. Obtener la versión de OpenBLAS del sitio `https://www.openblas.net/`

2. Descomprimir y desagrupar OpenBLAS.0.2.20.tar.gz
   `tar -xvzf OpenBLAS.0.2.20.tar.gz`

3. Procedemos a la instalacion
   `cd OpenBLAS.0.2.20; make`

4. `make PREFIX=/usr/local/OpenBLAS install`

Ambiente de Trabajo: OPENBLAS_NUM_THREADS

Una vez instalado debemos agregar la opción de compilación `libopenblas.so`. Para ejecutar se habilitará la variable de ambiente `OPENBLAS_NUM_THREADS` en donde se indica el número de cores que el programa utilizará.

3.3.4. Implementacion de OpenMp

Para llevar a cabo la puesta en marcha de deMon con OpenMP, se realizaron las siguientes actividades: *Recopilación de las características del programa durante su ejecución, Medición de tiempo de ejecución, Número de llamadas de funciones*, basado en este análisis se detectaron las secciones de código a optimizar, esto se realizo sin perfiladores comerciales, a continuación se muestra el código modificado.

```
      SUBROUTINE JACOBI(H,N,NDIM,IEGEN,EIGVAL,NR,IQ,U,X)
C
C     Purpose: JACOBI computes eigenvalues and eigenvectors of a real
C              symmetric matrix using the Jacobi method.
C
C     History: - Creation (June 1994, B. Ahlswede; the original program
C                          was written by F. Corbato and M. Merwin)
C
C              - Implemented in deMon (19.10.00, AMK)
C                                     (11.06.14, GG)
C
C     ***************************************************************
C
C     On input:
C
C            H: Contains the matrix to be diagonalized.
C               Only the upper triangle of the matrix need to
C               be supplied. This part is changed on exit,
C               but the diagonal elements are unchanged.
C            N: Order of matrix H.
C         NDIM: Dimension of H as declared in the calling program.
C        IEGEN: Must be 0 if eigenvectors are to be computed.
C
```

```
C     On output:
C
C        EIGVAL: Contains the eigenvalues in ascending order.
C             U: Contains the corresponding eigenvectors.
C            NR: Number of Jacobi-rotations.
C
C     ------------------------------------------------------------------
      use omp_lib
C
      IMPLICIT NONE
C
      REAL EPS
      PARAMETER (EPS = 1.E-10)
      integer, parameter :: out_unit=20
C
      INTEGER I,IEGEN,IMI1,IPIV,J,JPIV,K,N,NDIM,NR
      INTEGER :: NH, nthreads
      REAL COSINE,HDIMIN,HDTEST,HII,HTEMP,P,PYTHAG,SINE,TANG,XDIF,XMAX
C
      REAL EIGVAL(NDIM),H(NDIM,NDIM)
C
      INTEGER IQ(NDIM)
      REAL U(NDIM,NDIM),X(NDIM)
C
C     ------------------------------------------------------------------
C

C     *** Save the diagonal elements ***
C
!$omp parallel do private (I,N)
      DO 10 I=1,N
        EIGVAL(I) = H(I,I)
   10 CONTINUE
!$OMP END PARALLEL DO
C
C     *** Initialize eigenvectors ***
C
      IF (IEGEN.EQ.0) THEN
!$omp parallel do private (I)

        DO 610 I=1,N
!$omp parallel do private (J)
          DO 600 J=1,N
            IF (I.EQ.J) THEN
              U(J,I) = 1.0
            ELSE
              U(J,I) = 0.0
            END IF
  600     CONTINUE
```

```
!$OMP END PARALLEL DO
  610   CONTINUE
!$OMP END PARALLEL DO

      END IF
C
      NR = 0
      IF (N.LE.1) GO TO 1100
C
C     *** Scan for largest off diagonal element in each column      ***
C     *** X(I) contains the largest element in ith column           ***
C     *** IQ(I) holds second subscript defining position of element ***
C
!$omp parallel do private (I,N)
      DO 35 I=2,N
        X(I) = 0.0
        IMI1 = I - 1
        DO 30 J=1,IMI1
          IF (X(I).LE.ABS(H(J,I))) THEN
            X(I) = ABS(H(J,I))
            IQ(I) = J
          END IF
   30   CONTINUE
   35 CONTINUE
!$OMP END PARALLEL DO
C
C     *** Set indicator for shut-off ***
C
      HDTEST = 1.0E38
C
C     *** Find maximum of X(I)'s for pivot element ***
C     *** and test for end of problem              ***
C
   40 CONTINUE
C
      XMAX = 0.0
      DO 70 I=2,N
        IF (XMAX.LT.X(I)) THEN
          XMAX = X(I)
          IPIV = IQ(I)
          JPIV = I
        END IF
   70 CONTINUE

C
      IF (XMAX.LE.0.0) GO TO 1000
      IF ((HDTEST.GT.0.0).AND.(XMAX.GT.HDTEST)) GO TO 148
      HDIMIN = ABS(EIGVAL(1))
C
```

```
C     *** Search for smallest diagonalelement ***
C
      DO 110 I=2,N
        IF (HDIMIN.GT.ABS(EIGVAL(I))) THEN
          HDIMIN = ABS(EIGVAL(I))
        END IF
  110 CONTINUE
C
C     *** End rotations if MAX(H(I,J)).LT.(EPS)*ABS(H(K,K)-MIN) ***
C
      HDTEST = HDIMIN*EPS
      IF (HDTEST.GE.XMAX) GO TO 1000
      IF (XMAX < TINY(1.0)*1000.0) GO TO 1000
C
  148 CONTINUE
C
      NR = NR + 1
C
C     *** Compute TANG, SINE AND COSINE, H(I,I), H(J,J) ***
C     *** Formel: Quantenchemie Bd. 5,  S. 473          ***
C
      XDIF = EIGVAL(IPIV) - EIGVAL(JPIV)
      P = XDIF/(2.0*H(IPIV,JPIV))
C
C     *** PYTHAG berechnet SQRT(A**2 + B**2) - aus Eispack ***
C
      TANG = 1.0/(P+SIGN(1.0,P)*PYTHAG(P,1.0))
      COSINE = 1.0/PYTHAG(1.0,TANG)
      SINE = TANG*COSINE
      HII = EIGVAL(IPIV)
      EIGVAL(IPIV) = COSINE*COSINE*(HII + TANG*(2.0*H(IPIV,JPIV) +
     +                TANG*EIGVAL(JPIV)))
      EIGVAL(JPIV) = COSINE*COSINE*(EIGVAL(JPIV) -
     +                TANG*(2.0*H(IPIV,JPIV) - TANG*HII))
      H(IPIV,JPIV) = 0.0
C
C     *** Inspect the IQ's between I+1 and N-1 to determine   ***
C     *** wether a new maximum value should be computed since ***
C     *** the present maximum is in the I's or J's column     ***
C
C!$OMP PARALLEL SECTIONS
C!$OMP SECTION
      DO 350 I=2,N
        IF ((I.EQ.IPIV).OR.(I.EQ.JPIV)) GO TO 350
        IF ((IQ(I).EQ.IPIV).OR.(IQ(I).EQ.JPIV)) THEN
          K = IQ(I)
          HTEMP = H(K,I)
```

```
          H(K,I) = 0.0
          IMI1 = I - 1
          X(I) = 0.0
C
C     *** Search in depleted column new maximum ***
C
          DO 320 J=1,IMI1
            IF (X(I).GT.ABS(H(J,I))) THEN
              X(I) = ABS(H(J,I))
              IQ(I) = J
            END IF
  320     CONTINUE
          H(K,I) = HTEMP
        END IF
  350 CONTINUE
C!$OMP END PARALLEL SECTIONS
C
C     *** Change the order of elements of H ***
C
C     *** I < IPIV, I < JPIV ***
C
      X(IPIV) = 0.0
      X(JPIV) = 0.0
      DO 400 I=1,IPIV-1
        HTEMP = H(I,IPIV)
        H(I,IPIV) = COSINE*HTEMP + SINE*H(I,JPIV)
        IF (X(IPIV).LT.ABS(H(I,IPIV))) THEN
          X(IPIV) = ABS(H(I,IPIV))
          IQ(IPIV) = I
        END IF
        H(I,JPIV) = - SINE*HTEMP + COSINE*H(I,JPIV)
        IF (X(JPIV).LT.ABS(H(I,JPIV))) THEN
          X(JPIV) = ABS(H(I,JPIV))
          IQ(JPIV) = I
        END IF
  400 CONTINUE
C
C     *** I > IPIV, I < JPIV ***
C
      DO 410 I=IPIV+1,JPIV-1
        HTEMP = H(IPIV,I)
        H(IPIV,I) = COSINE*HTEMP + SINE*H(I,JPIV)
        IF (X(I).LT.ABS(H(IPIV,I))) THEN
          X(I) = ABS(H(IPIV,I))
          IQ(I) = IPIV
        END IF
        H(I,JPIV) = - SINE*HTEMP + COSINE*H(I,JPIV)
        IF (X(JPIV).LT.ABS(H(I,JPIV))) THEN
          X(JPIV) = ABS(H(I,JPIV))
```

```
          IQ(JPIV) = I
        END IF
  410 CONTINUE
C
C     *** I > IPIV, I > JPIV ***
C
      DO 420 I=JPIV+1,N
        HTEMP = H(IPIV,I)
        H(IPIV,I) = COSINE*HTEMP + SINE*H(JPIV,I)
        IF (X(I).LT.ABS(H(IPIV,I))) THEN
          X(I) = ABS(H(IPIV,I))
          IQ(I) = IPIV
        END IF
        H(JPIV,I) = - SINE*HTEMP + COSINE*H(JPIV,I)
        IF (X(I).LT.ABS(H(JPIV,I))) THEN
          X(I) = ABS(H(JPIV,I))
          IQ(I) = JPIV
        END IF
  420 CONTINUE
C
C     *** Compute new eigenvectors ***
C
      IF (IEGEN.EQ.0) THEN
        DO 550 I=1,N
          HTEMP = U(I,IPIV)
          U(I,IPIV) = COSINE*HTEMP + SINE*U(I,JPIV)
          U(I,JPIV) = - SINE*HTEMP + COSINE*U(I,JPIV)
  550   CONTINUE
      END IF
C
C     *** GO TO NEXT CYCLE ***
C
      GO TO 40
C
 1000 CONTINUE
C
C     *** Sort eigenvalues ***
C
      DO 920 I=1,N-1
        K = I
        P = EIGVAL(I)
        DO 910 J=I+1,N
          IF (EIGVAL(J).LT.P) THEN
            K = J
            P = EIGVAL(J)
          END IF
  910   CONTINUE
        IF (K.NE.I) THEN
          EIGVAL(K) = EIGVAL(I)
```

```
          EIGVAL(I) = P
          DO 900 J=1,N
            P = U(J,I)
            U(J,I) = U(J,K)
            U(J,K) = P
  900     CONTINUE
        END IF
  920 CONTINUE
C
 1100 CONTINUE
C
C     ------------------------------------------------------------------
C
C     *** End of SUBROUTINE JACOBI ***
C
      close (out_unit)
      END
```

Capítulo 4

Resultados

Las métricas de desempeño surgen cuando se desarrollan soluciones en paralelo, la primera interrogante es *"el cuán más rápido se resuelve el problema"*, pero antes de saber cuanto se acelera en paralelo es necesario enfrentar el algoritmo secuencial (en un solo procesador) para saber la aceleración y ver si compensa la inversión en el sistema paralelo.

Ahora bien, al saber que se tiene la mejor versión de la solución en secuencial podemos compararla contra la paralela. Es necesario aclarar que las mediciones no son exactas, si no un promedio, ya que existen más factores que influyen como la asignación de recursos del sistema operativo o la cantidad de memoria que se posea, entre otras cosas [13].

Para este proyecto se dedico un nodo de un Cluster, las características que cuenta el nodo son:

Nodo: n0	
Procesador	**AMD Opteron(tm) Processor 6376, 32 cores**
Memoria	**47 Gb**
Disco duro	**1 TB**
Sistema Operativo	**CentOS Linux release 7.5.1804**
Compilador	**gcc version 4.8.5 20150623 (Red Hat 4.8.5-28) (GCC)**
OpenMP	**openmpi 1.10.7**

Tabla 4.1: Características del nodo de pruebas.

4.0.1. Evaluación: Runtime

El tiempo de ejecución $T_p(n)$, se toma a partir del inicio del programa y hasta la finalización de todos los procesos. El programa con el se realizaron las pruebas es nano-cúmulo de Au de con tamaños: *8,18,32* los resultados se muestran en la tabla 4.2 y graficas 4.1.

	Número de procesadores	Tiempo (segundos)
Au8	1	843.0
	2	408.0
	4	327.0
	8	356.0
Au18	1	9729.0
	2	3250.0
	4	3245.0
	8	2884.0
	16	2960.0
Au32	1	64694.7
	2	17078.0
	4	16620.0
	8	16245.0
	16	13840.0

Tabla 4.2: Tiempo de ejecución de nano-cúmulos de Oro de tamaño 8,18 y 32.
Se puede apreciar que para nano-cúmulos de Oro de tamaño menores a 32, cuando se incrementa el número de procesadores el tiempo de procesamiento también aumenta, ésto es porque existe una relación óptima entre el tamaño de tarea y el número de procesadores que se utilizaran. Si se aumentan el número de procesadores el tiempo consumido en repartir las tareas crecerá y no será compensado por el paralelismo.

En estos resultados se puede apreciar la importancia de conocer la relación del tamaño de datos y el número de procesadores en el que se trabajará. En cálculos menores de 32 átomos con ocho procesadores el tiempo se ejecución es superior que al utilizar cuatro procesadores, esto es por que la cantidad de tiempo requerido para coordinar tareas paralelas influirá de forma negativa para trabajos en donde la la asignación de tareas de paralelas controlará una cantidad de datos pequeña en relación al número de procesadores.

4.0.2. Evaluación: Cost Factor

Representa la cantidad de trabajo realizado por el programa, ya que, aunque disminuye el tiempo al tener un sistema paralelo, los procesadores están trabajando al mismo tiempo, por ello muchas veces es necesario conocer la totalidad del costo que conlleva la utilización del algoritmo. Los resultados de las pruebas se muestran en la tabla 4.3 y figura 4.2. Como se puede apreciar el costo óptimo si ejecuta la misma cantidad total de operaciones que el programa secuencial más rápido que tiene tiempo de ejecución.

$$C_p(n) = p * T_p(n)$$

donde:

p *número de procesadores*

	Número de procesadores	Tiempo (segundos)	Cost Factor
Au8	1	843.0	843.0
	2	408.0	816.0
	4	327.0	1308.0
	8	356.0	2848.0
Au18	1	9729.0	9729.0
	2	3250.0	6500.0
	4	3245.0	12980.0
	8	288.0	23072.0
	16	2960.0	47360.0
Au32	1	64694.7	64694.7
	2	17078.0	34156.0
	4	16620.0	66480.0
	8	16245.0	129960.0
	16	13849.0	221584.0

Tabla 4.3: Cost factor de nano-cúmulos de Oro de tamaño 8,18 y 32.

4.0.3. Evaluación: Speedup

Es la medición relativa del rendimiento de un programa en paralelo, para calcularla se necesita medir el tiempo que tarda la versión secuencial entre la versión paralela. Se expresa de la siguiente forma:

$$S(p) = \frac{T^*(n)}{T_p(n)}$$

donde:
** Es la ejecución del tiempo serial*
p número de procesadores.

Hay que tener en cuenta que se debe calcular el tiempo con la misma cantidad de datos a procesar para que la medición sea justa. El SpeedUp observado se aprecia en la tabla 4.4 y figura 4.3, el factor de aceleración con mas procesadores a menudo conducen a más velocidad; sin embargo existen factores que ocasionarán que se reduzca la velocidad, como sucede en el cálculo de Au8 con 8 procesadores y Au18 con 16 procesadores.

	Número de procesadores	Tiempo (segundos)	Speedup
Au8	1	843.0	1
	2	408.0	2.0
	4	327.0	2.6
	8	356.0	2.4
Au18	1	9729.0	1
	2	3250.0	2.9
	4	3245.0	3.0
	8	2884.0	3.4
	16	2960.0	3.3
Au32	1	64694.6	1
	2	17078.0	3.8
	4	16620.0	3.9
	8	16245.0	4.0
	16	13849.0	4.7

Tabla 4.4: Speedup de nano-cúmulos de Oro de tamaño 8,18 y 32.
Se observa que el factor de aceleración se incrementa para casi todos casos, cuando se aumenta el número de procesadores.

4.0.4. Evaluación: Eficiencia

El desempeño de un programa paralelo es la eficiencia, esta indica la fracción en la cual un procesador es utilizado de manera útil por cálculos que también tienen que ser realizados

por un programa secuencial. Si se conoce cuánto tarda cada procesador en realizar su tarea, regularmente se expresa en porcentaje, por ejemplo, si se obtiene el 50 % significa que se están utilizando la mitad del tiempo en promedio que en la versión secuencial.

Los resultados respecto a eficiencia se muestra en la tabla 4.5 y gráfica 4.4.

$$E_p(n) = \frac{T^*(n)}{C_p(n)} = \frac{S_p(n)}{p} = \frac{T^*(n)}{p * T_p(n)}$$

	Número de procesadores	Tiempo (segundos)	Eficiencia
Au8	1	842.963	1
	2	408	1.0
	4	327	0.6
	8	356	0.3
Au18	1	9729.005	1
	2	3250	1.5
	4	3245	0.7
	8	2884	0.4
	16	2960	0.2
Au32	1	64694.757	1
	2	17078	1.9
	4	16620	1.0
	8	16245	0.5
	16	13849	0.3

Tabla 4.5: Eficiencia de nano-cúmulos de Oro de tamaño 8,18 y 32.
En la tabla se observa que cuando el número de procesadores aumenta el porcentaje de eficiencia mejora en los casos (8,18 y 32), para el caso de 2 procesadores la eficiencia decrece al aumentar el número de procesadores.

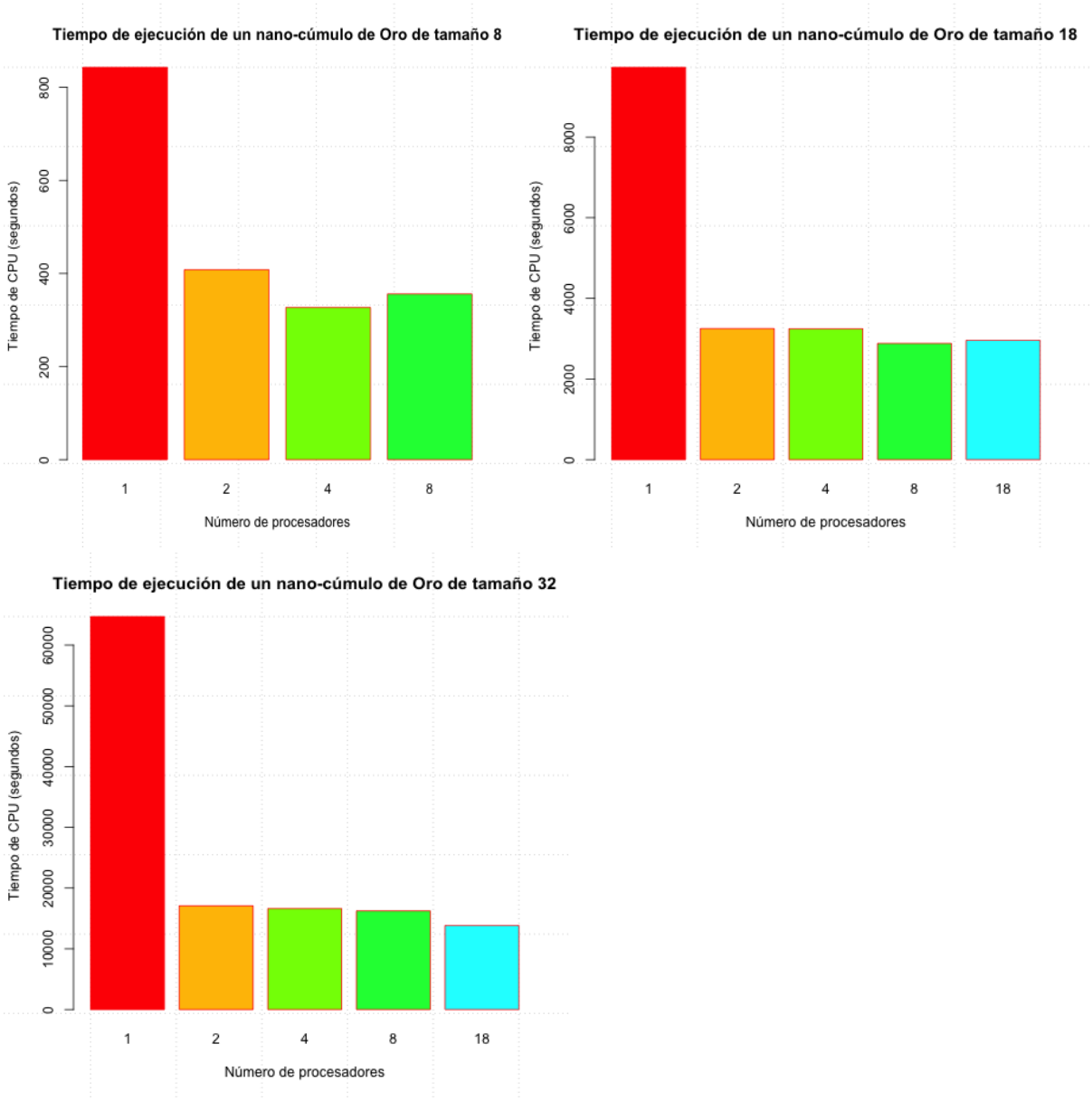

Figura 4.1: Tiempos de Ejecución del Au8, Au18 y Au32.
Para los casos de nano-cúmulos de Oro de tamaño menores a 32, cuando se incrementa el número de procesadores el tiempo de procesamiento también aumenta por la correlación entre el tamaño de tarea y el número de procesadores que se utilizaran.

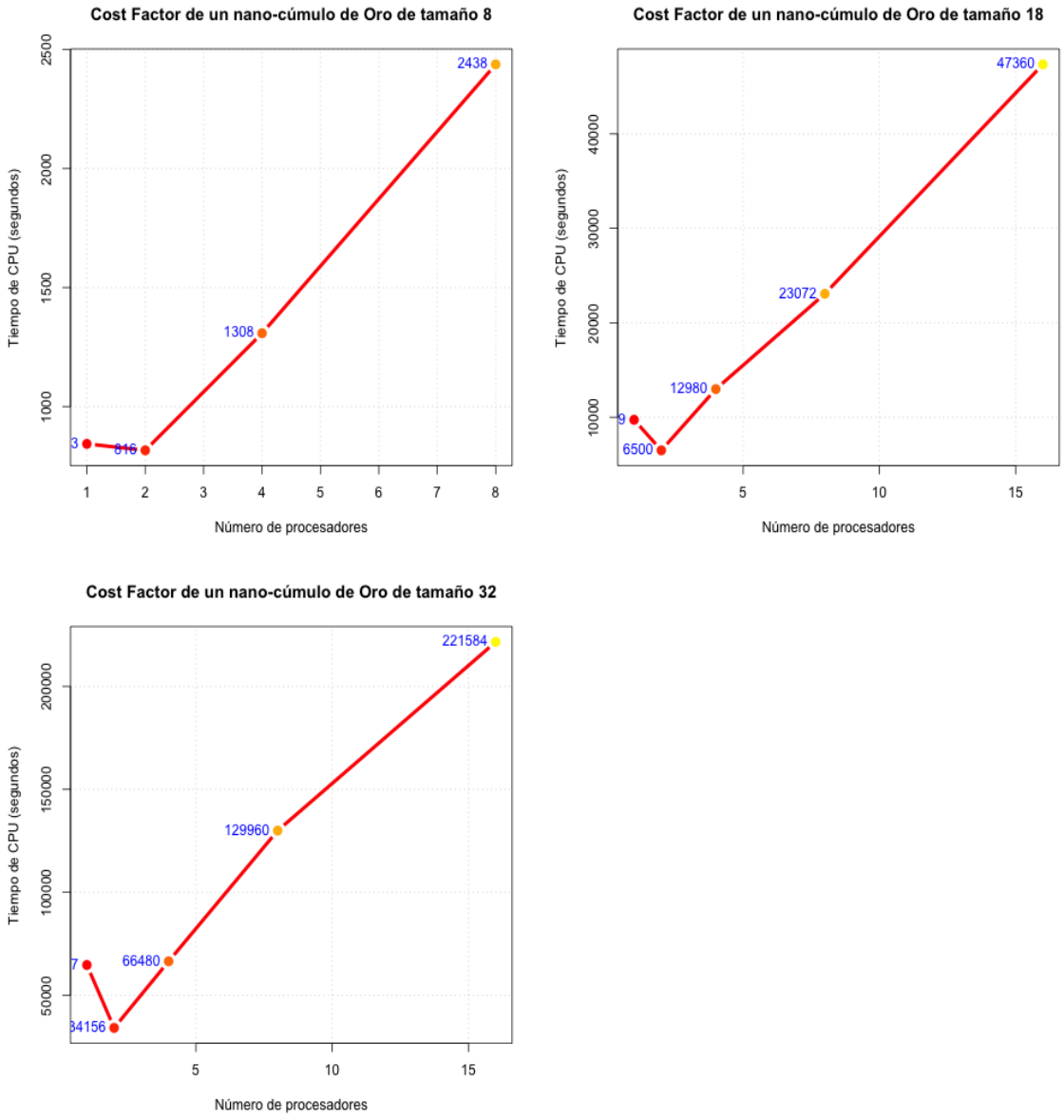

Figura 4.2: Se muestra el cost factor para nano-cúmulos de Oro de tamaño Au8, Au18 y Au32.

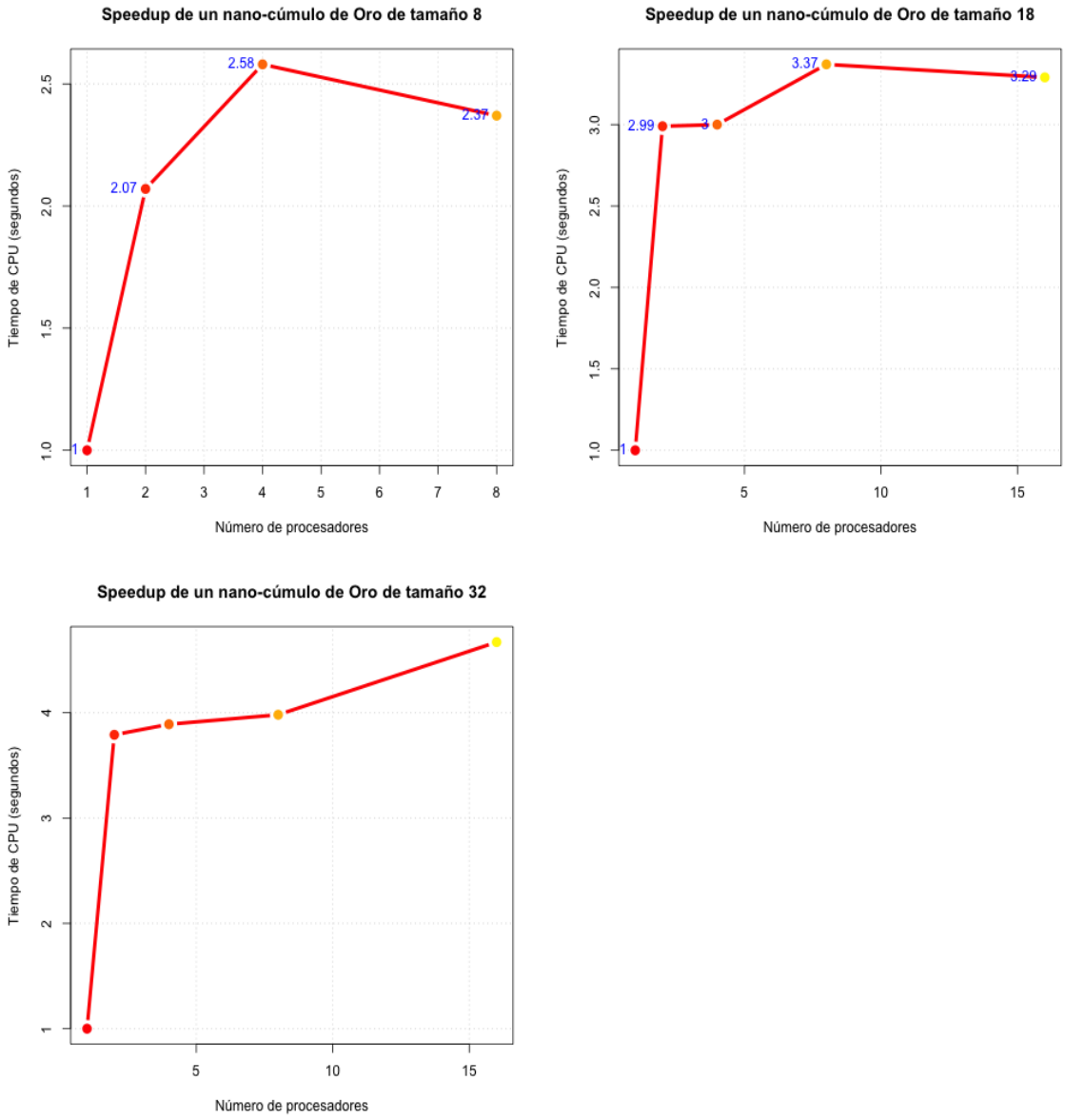

Figura 4.3: SpeedUp para nano-cúmulos de Oro de tamaño 8,18 y 32.
Se observa como actúa el factor de aceleración para casi todos los casos, también se aprecia la relación que existe entre el tamaño de datos, el algoritmo paralelo y el número de procesadores no siempre reduce el tiempo de cómputo.

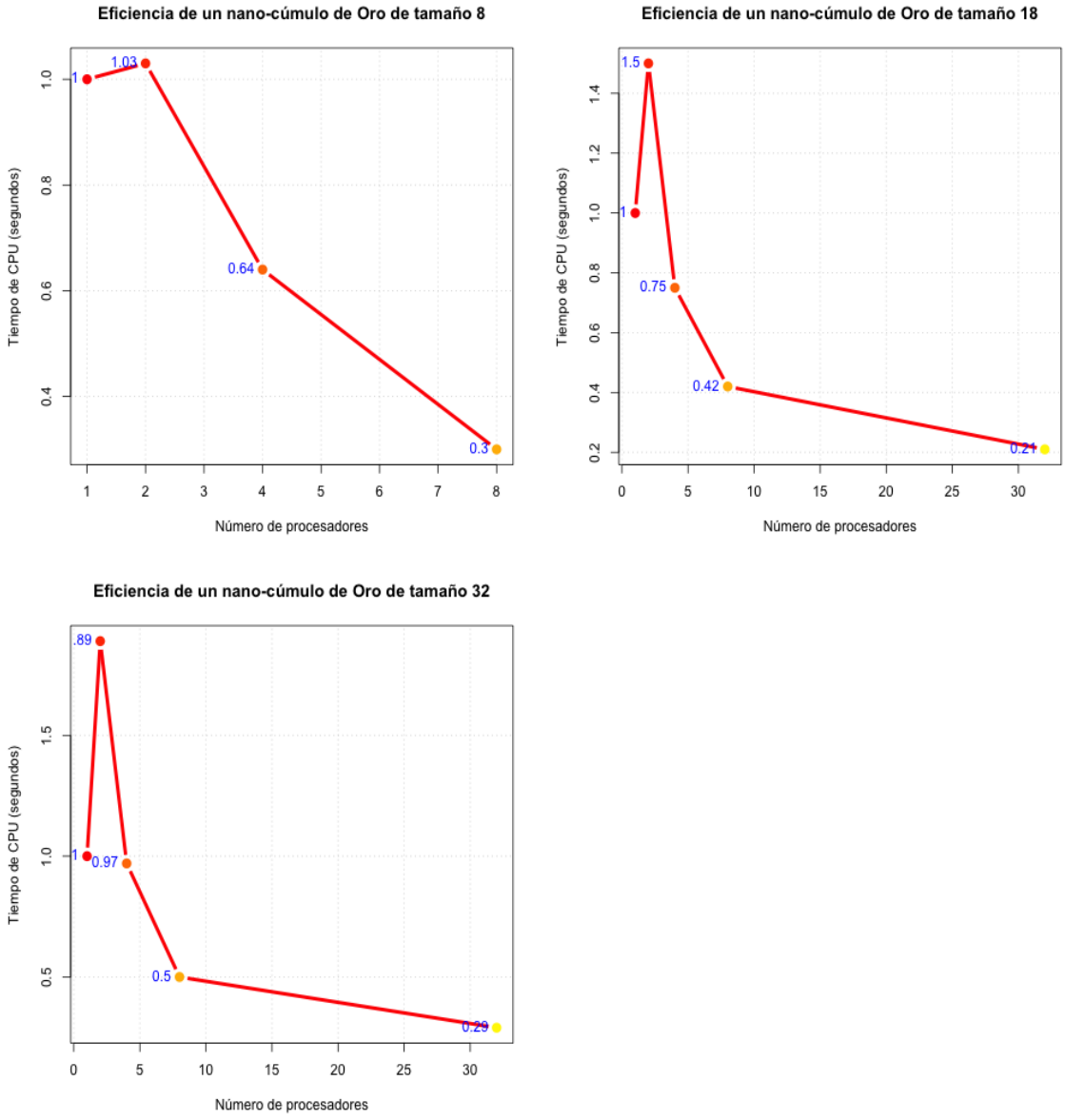

Figura 4.4: Evaluación de la eficiencia en nano-cúmulos de Oro de tamaño 8,18 y 32.

Capítulo 5

Discusión y Conclusiones

5.0.1. Discusión

Actualmente en cómputo intensivo hay muchas alternativas para paralelizar un código o una parte de éste, en este trabajo se exploró una de ellas, que fue la implementación de **OpenMP** en el programa *deMon2K*, para llevarla a cabo se realizó una evaluación detallada del funcionamiento de deMon2k y las directivas de OpenMP que mejor adecuarán con la operación del programa.

En este trabajo el objetivo principal es la *implementación de OpenMP en el programa deMon2k versión 6.0.2*, para alcanzarlo se trabajo en diferentes aspectos que a continuación se mencionan:

Primeramente se realizó un análisis del funcionamiento de programa **deMon2K**,en específico la comunicación entre los programas, ya que era primordial conocer el flujo de información y como está, interactúa con los programas que lo conforman; desafortunadamente no se cuenta con depuradores, el análisis se llevo cabo modificando programas y guardando los resultados en archivos. Derivado del análisis se encontró que el código el **deMon2K** utiliza bibliotecas *Basic Linear Algebra Subprogramas (BLAS)* de forma eficiente y recáe parte importante de operaciones matemáticas por lo que no era necesario paralelizar estas bibliotecas, por fortuna existe el proyecto OpenBlas en donde estas bibliotecas están paralelizadas con OpenMP y optimizadas, así que se adaptaron al código.

Por otra parte, de este análisis, también se determinó la conveniencia de implementar OpenMP en las rutinas de diagonalización, en particular en el programa `jacobi.f` ya que en este se reúne una gran cantidad de operaciones matemáticas. Aqui es conveniente

señalar que la inserción de las directivas adecuadas no es cualquier cosa, ya que deben corresponder al funcionamiento del código, una mala directiva puede entorpecer en lugar de eficientizar el programa.

Los resultados que obtuvimos con nuestra implementación de OpenMP fueron satisfactorios, ya que en ejecuciones con 2 procesadores se redujo a la mitad o mas de mitad del tiempo, sin embargo, la reducción de tiempo con mas procesadores no se alcanzo con el mismo comportamiento, es decir la reducción del tiempo cuando este era mayor a 4 procesadores fue aproximadamente del 60 %, ya que depende del tamaño del calculo, también se pudo comprobar que cuando el calculo era mayor y se usaban mas procesadores se lograba una mayor eficiencia.

5.0.2. Conclusiones

Durante el desarrollo de este trabajo se cumplió con los objetivos planteados de manera satisfactoria:

- Se determinó que las rutinas de diagonalización son la parte del código mas conveniente para paralelizar usando **OpenMP.**
- Se reemplazaron las bibliotecas BLAS por las bibliotecas **OpenBLAS** y se mantuvo la eficiencia y disminuyeron los tiempos de ejecución.
- Para un tamaño dado del cálculo a realizar, hay un número de procesadores para los cuales se tenía la mayor eficiencia, si se aumentaba el número de procesadores la eficiencia bajaba.
- El tamaño del cálculo influye en la eficiencias del código para un número dado de procesadores.
- En este trabajo la mejor eficiencia fue obtenida con 2 procesadores en todos los tamaños de las tareas (cúmulos de 8,18 y 32 átomos).
- La eficiencia con 4 procesadores es optima para el caso de 32 átomos.
- En los resultados se observa una tendencia que indica que la eficiencia aumenta si aumenta el tamaño de la tarea.

5.0.3. Trabajo a futuro

En este trabajo se ha estudiado y creado un modelos paralelos para el programa *deMon2K* utilizando la memoria compartida. Como una línea futura se piensa en migrar el código a las nuevas tecnologías de programación en paralelo, aquellas que utilizan las tarjetas gráficas.

Bibliografía

[1] *M.D. Hill and M.R. Marty.* **Amdahl's law in the multicore era. IEEE Computer Society, 33–38,** Consultado en febrero de 2019, `http://research.cs.wisc.edu/multifacet/ papers/ieeecomputer08 amdahl multicore.pdf`

[2] *Leslie G. Valiant.* **A bridging model for parallel computation** Communications of the ACM, Vol.33, Issue 8 Pp. 103-111, 1997.

[3] *G. W. George Hager.* **Introduction to High Performance Computing for Scientists and Engineers** T. & F Group, Año 2011

[4] *Peter Pachecho.* **An introduction to parallel programing.** Morgan Kaufmann, 2011.

[5] *W.P. Petersen and P. Arbenz.* **Introduction to Parallel Computing.** Oxford University Press, 2004.

[6] *A software package for density functional theory (DFT) calculation.* Consultado en febrero de 2019, `http://www.demon-software.com/public_html/index.html`

[7] *B. Chapman, G. Jost, and R. Pas.* **Using OpenMP: portable shared memory parallel programming. Scientific and engineering computation.** MIT Press, 2007.

[8] *R.Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald.* **Parallel Programming in OpenMP** Morgan Kaufmann, 2001.

[9] *Installation Guide* Consultado en enero de 2019,https://github.com/xianyi/OpenBLAS/wiki

[10] *Celebrating the 50th Anniversary of Moore's Law* Consultado en junio de 2019,https://technology.ihs.com/api/binary/532884

[11] *Past Project:* **GotoBLAS2** Consultado en diciembre de 2018, https://www.tacc.utexas.edu/research-development/tacc-software/gotoblas2

[12] *BLAS (Basic Linear Algebra Subprograms)* Consultado en febrero de 2019, http://www.netlib.org/blas/

[13] *M. A. Barry Wilkinson* **Parallel Programming: Techniques and Aplications Using Networked Workstations an Parallel Computers** Segunda Edicion ed., Pearson, 2005.

[14] *Eijkhout, Victor*, **An Introduction to Parallel and Vector Scientific Computing by Ronald W. Shonkwiler; Lew Lefton** SIAM Review, 2008 , 10.2307/20454083

[15] *GNU Press a division of the Free Software Foundation.* **The GNU OpenMP Implementation** Consultado en enero 2019 http://gcc.gnu.org/onlinedocs/libgomp.pdf

[16] *William B. Clodius* **User notes on Fortran programming** $UNFP$ Consultado en diciembre 2018 http://www.ibiblio.org/pub/languages/fortran/unfp.html

[17] *Paul Brook and et.* **The GNU Fortran Compiler.** Consultado en diciembre 2018 https://gcc.gnu.org/onlinedocs/gcc-4.8.5/gfortran/

[18] *C. Severance and K. Dowd.* **High Performance Computing.** Draft version:Rice University, Houston,Texas, Consultado en diciembre 2018 http://cnx.org/content/col11136/1.5/.

[19] *S. Nakamura.* **Métodos numéricos aplicados con software.** Prentice-Hall Hispanoamericana, 1992.

[20] *J Dongarra, I Foster, G Fox, W Gropp, K Kennedy* **Sourcebook of parallel computing** Morgan Kaufmann, 2003.