4to Congreso Argentino de Ingeniería Aeronáutica

OPTIMIZACION DEL CALCULO DE PARAMETROS DEL ALA TRIDIMENSIONAL PARA SER UTILIZADOS EN SIMULACION DE VUELO

D. S. Monserrat[a], **F. G. Tinetti**[b]

[a]*Dpto. Ing. Aernáutica., Facultad Regional Haedo, Universidad Tecnológica Nacional, Paris 532 (1706) Haedo, Buenos Aires, Argentina.* http://www.frh.utn.edu.ar

[b]*Investigador de la Comisión de Investigaciones Científicas Prov. de Bs. As., Facultad de Informática, Universidad Nacional de La Plata, La Plata, Buenos Aires, Argentina.* http://www.unlp.edu.ar

Palabras claves: Simulación, Prandtl, Glauert, Sustentación

Resumen

El modelo de la línea sustentadora de Prandtl, utilizado para predecir las características de alas tridimensionales, en flujo subsónico, fue desarrollado por Prandtl mediante su ecuación integro-diferencial. Hermann Glauert propuso un método de integración numérica de dicha ecuación.

El presente trabajo tiene por objeto analizar la implementación del método de Glauert de forma de lograr su optimización con vistas al uso del mismo en simulaciones en tiempo real con piloto en la cabina (pilot in the loop), ya que, lo que se busca en estos casos es el menor tiempo de cómputo para no afectar al resto de la simulación.

La metodología del presente trabajo ha sido primero plantear un ala rectangular con perfil NACA 4412, como caso de estudio. Luego se ha implementado el método de Glauert, resolviendo el sistema de ecuaciones involucrado, utilizando un algoritmo estándar de cómputo como es el de eliminación de Gauss con pivotes para preservar la estabilidad numérica del método. Se computan y analizan los tiempos de procesamiento desde la perspectiva de requerimientos de una simulación en tiempo real.

Luego se modificó el código para paralelizar partes del mismo y así buscar la reducción de tiempos de procesamiento. Por último se ha incorporado el uso de librerías como ATLAS (Automatically Tuned Linear Algebra Software), con algoritmos optimizados y se analiza la existencia o no de una mejora en los tiempos de procesamiento.

Los resultados obtenidos demuestran que se ha logrado una reducción en el tiempo de cómputo, lo cual abre la posibilidad de utilizar esta solución en un simulador de vuelo o en aplicaciones que requieran su resolución en tiempo real y constituye una base para la optimización de otros algoritmos.

e-mail: dmonserrat@frh.utn.edu.ar

1. INTRODUCCIÓN

El modelo de la línea sustentadora de Prandtl utilizado para predecir las características de alas tridimensionales, en flujo subsónico, fue desarrollado por Prandtl mediante su ecuación integro-diferencial:

$$\omega(y_1) = \frac{1}{4\pi}\int_{b/2}^{-b/2} \frac{\frac{\partial \Gamma}{\partial y}}{y_1 - y}\, dy \tag{1}$$

Siendo el ángulo de ataque efectivo igual a

$$\alpha_e(y) = \alpha(y) - \frac{\omega(y)}{V} \tag{2}$$

La integración de este modelo fue resuelta por Multhopp, Glauert, etc. los cuales resolvieron la integración para determinadas condiciones, como es el caso de Glauert que utilizó el desarrollo en series de Fourier aplicado para el caso de alas con distribuciones simétricas. La solución general de Glauert fue:

$$\Gamma_i = 2\, b\, V_i \sum_{n=1}^{m} An\, Sen(n\theta_i) \tag{3}$$

Donde "m" es el número de estaciones o particiones del ala que se consideren para el cálculo y $\theta = -\frac{2y}{b}$

Este método se ha utilizado para la predicción de las características de funcionamiento de spoilers en automovilismo deportivo, como también en alas y estabilizadores horizontales en aeronaves.

El presente trabajo tiene por objetivo analizar la implementación del método de Glauert para su optimización con vistas al uso del mismo en simulaciones en tiempo real con piloto en el circuito, ya que requiere el menor tiempo de cómputo posible de forma de no afectar al resto de la simulación. El método en si es la versión extendida desarrollada en [1].

El método consiste en la división (discretización) del ala bajo estudio en punto o estaciones donde se calcularán las características locales en cada uno, como se muestra en la Figura 1 a continuación.

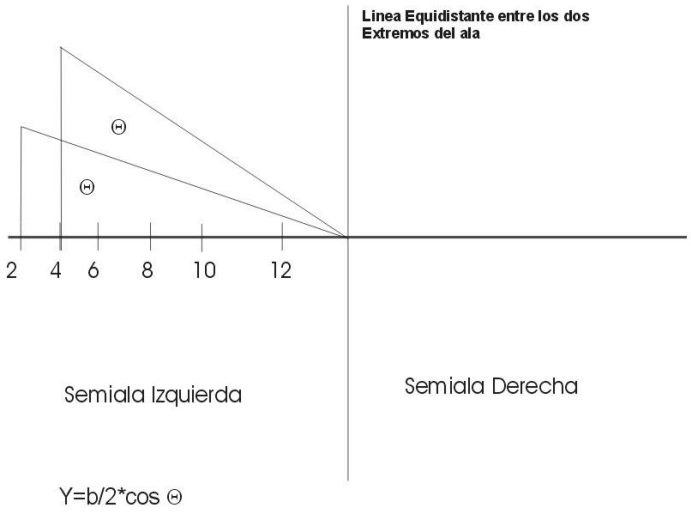

Figura 1: Distribución de Estaciones

Luego, en base a las características locales de cada punto se obtienen las características del ala completa. La Figura 2 es una vista de una simulación, que utiliza este método, donde se observa en color verde la distribución de coeficiente de sustentación resultante sobre el ala en cuestión (El gráfico de la aeronave es indicativo únicamente y no se corresponde necesariamente con el modelo cargado).

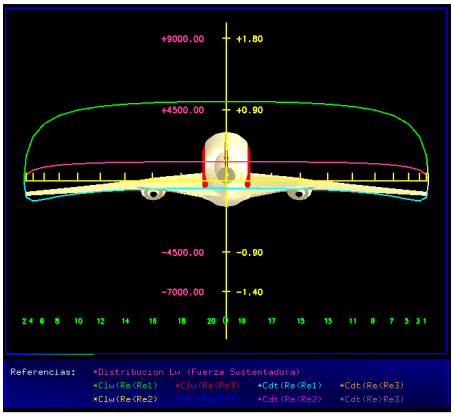

Figura 2: Ejemplo de Aplicación

Debemos hacer notar que la intención de este trabajo no es evaluar el método de Glauert en si mismo, sino, tomada una implementación del mismo, lograr su optimización para el uso en simulaciones de tiempo real.

Para el estudio de la optimización vía cómputo en paralelo utilizamos OpenMP (librería multiplataforma para procesamiento en paralelo con memoria compartida), la cual nos permite de forma sencilla implementar el procesamiento en múltiples hilos de ejecución.

En parte del estudio hemos utilizado ATLAS (Automatically Tuned Linear Algebra Software) el cual es un proyecto de investigación enfocado en aplicar técnicas empíricas para proveer rendimiento y portabilidad. Actualmente provee interfaces C y Fortran77 que implementan BLAS (Basic Linear Algebra) y LAPACK (Linear Algebra Package).

Los pasos que seguiremos en el desarrollo del trabajo son:

- Preparar programa de prueba.
- Medición de los tiempos con la implementación actual.
- Estudio de Optimización utilizando OpenMP.
- Estudio de Optimización utilizando ATLAS.
- Estudio de Optimización utilizando ATLAS con soporte paralelo.
- Estudio de Optimización utilizando OpenMP y ATLAS.

2. PROCEDIMIENTO Y DESARROLLO

Se ha tomado para el estudio una implementación del método de Glauert que se había implementado en el Grupo de Simulación Dinámica del Vuelo (UTN FRH) lo cual nos impone la restricción de mantener la forma de implementación estructural. La segunda restricción es el uso de Eclipse (www.eclipse.org) como entorno de desarrollo ya que constituye la herramienta actualmente establecida como estándar dentro del ámbito donde se utilizará el código. La tercera es que si bien el método es iterativo, como queremos mejorar la velocidad de cómputo del mismo, fijaremos un conjunto de datos los cuales sabemos se resuelven en una cantidad fija de iteraciones debido a que la cantidad de ciclos de iteración depende exclusivamente de los datos (situación) ingresados. Asumimos por lo tanto que logrando optimizar cada ciclo lograremos optimizar el cómputo de todas las situaciones a pesar de la cantidad de iteraciones que sean necesarias por el método en si mismo.

2.1. Implementación del programa de pruebas

Para la implementación de las pruebas se ha utilizado una computadora HP-530 con 2 GB de RAM, un procesador Intel CORE 2 DUO, sobre Ubuntu 10.04 como se puede observar en la Figura 3.

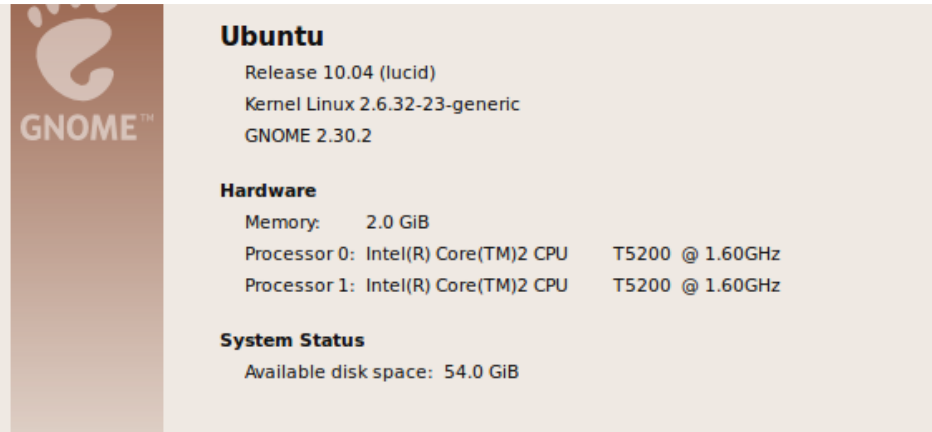

Figura 3: Plataforma Utilizada

El software de soporte se enumera en la tabla 1.

Nombre	Version
g++	4.4.3-4
CLAPACK	3.2.1
ATLAS	3.9.24 released 04/21/10
Eclipse	Galileo

Tabla 1: Plataforma de Software Utilizada

A continuación se presenta un diagrama de flujo del programa principal. Se observan tres bloques bien definidos:

1. Preparación de los datos: donde realizaremos los cálculos preliminares de los datos que sirven como entrada al cálculo. No interviene en la medición de los tiempos ya que se ejecuta una sola vez en el arranque de la simulación por lo cual no es de importancia.
2. Ejecución: se realiza el cálculo de los parámetros del ala y es dónde son medidos los tiempos de cómputo.
3. Validación y Presentación de Resultados: En esta etapa se realiza el cálculo del error numérico y se presentan los tiempos del cómputo.

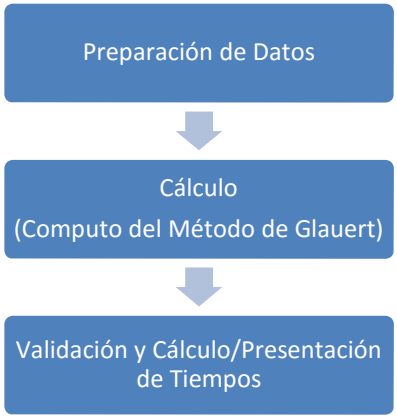

Figura 4: Flujo del Programa Principal

A continuación se presentan las partes relevantes del programa de prueba, en forma de pseudocódigo o diagramas con los comentarios pertinentes.

Los datos que se definen para realizar los cálculos se pueden observar en la Tabla 2, dónde básicamente son los datos correspondientes a un ala rectangular, sin alabeo, flecha ni alerones.

Parámetro	Valor
Tipo de Ala	Rectangular
Envergadura (b)	30.6 metros
Nro de Estaciones (n)	60 (30 por semiala)
Cuerda (C)	3.75 metros
Alargamiento (A)	10.309

Tabla 2: Parámetros Utilizados para el Cálculo

Dentro de los datos de entrada es necesario definir el perfil alar a utilizar. En nuestro caso utilizamos un perfil NACA 4412 cuya curva de coeficiente de sustentación en función del ángulo de ataque (Cl=f(α)) ha sido tabulada (Tabla 3). Para los valores que se encuentran entre dos puntos de la tabla se realiza una interpolación lineal.

Angulo de Ataque Bidimensional en Grados (α)	Coeficiente de Sustentación (Cl)
-90	0
-12	-1.21
-10	-1.06
0	0
10	1.06
12	1.21
14	1.34
15	1.35
17	1.392
20	1.14
21	1.065
90	0

Tabla 3: Tabla Representativa de las Curvas Cl=f(α)

Entonces la primera parte del programa de pruebas se encarga de generar los vectores que contienen los datos para cada una de las estaciones cuyos valores son los que hemos presentado hasta aquí. A continuación se presenta este primer bloque en forma de pseudocódigo:

```
.Fijar parámetros generales como alargamiento, envergadura, cantidad de estaciones.
.Para i:=1 hasta n estaciones
    .Asignar propiedades de la estación (cuerda local, perfil, etc.)
    .Calcular y asignar la posición y el ángulo de cada estación
.Fin Para
```

A continuación sigue lo que denominamos bloque de ejecución. En este bloque lo que realizamos es una llamada el método update de la clase Glauert, donde se ha implementado el cálculo de los parámetros del ala tridimensional por el método homónimo. En pseudocódigo es:

```
.tiempoInicio:= obtenerTiempoActual()
.glauert.update(datosDeEntrada)
.tiempoFinal:=obtenerTiempoActual()
```

Lo que se observa en estas líneas es que se almacena el tiempo antes y después de la llamada al método para luego computar el tiempo transcurrido. Update (función de actualización) encapsula el cálculo o implementación del método de Glauert y el resultado de su invocación es la resolución del método y cálculo de los parámetros del ala tridimensional acorde a los datos de entrada.

Una vez que se ha realizado el cálculo imprimimos los tiempos y datos característicos para saber si hubo anomalías durante los cambios realizados (checkpoint). En particular el cómputo del error se realiza comparando

si el sistema es A x = B, donde A son los coeficientes que hemos calculado resolviendo el sistema de ecuaciones, entonces calculamos A x por una lado y B por el otro y son comparados ambos de forma de determinar el error numérico introducido.

2.2. Caso 1: Implementación original

El código original, posee la implementación del método dentro de la función de actualización. La interface que implementa esta clase obedece a la arquitectura que ha definido el Grupo de Simulación Dinámica del Vuelo (GSDV).

El algoritmo que se utiliza para resolver el sistema de ecuaciones es el método de pivotes [2] para preservación de la estabilidad numérica en el cálculo de la solución. Inicialmente no se realizó ningún tipo de optimización de las operaciones involucradas o en las inicializaciones para que se realicen fuera de la actualización. Sin embargo, a pesar de todo esto, veremos la optimización por procesamiento en paralelo del método como está, sin entrar en primera instancia en las optimizaciones mencionadas anteriormente.

Habiendo expuesto las consideraciones del caso, vamos a ver de forma simplificada, en pseudocódigo la implementación de método update para ver los fragmentos de mayor interés en la optimización.

Se inicializa la matriz Z que es la matriz de coeficientes del sistema de ecuaciones. Es decir si el sistema es A x = B, sería la matriz A. Se puede observar que se han dejado específicamente los bucles de forma que se pueda observar los puntos susceptibles de optimizar. El código siguiente implementa el método de pivotes [2] para la resolución del sistema de ecuaciones.

```
.para I:=1 hasta n hacer
  .para J:=1 hasta n hacer
    .inicializar elemento I,J de la matriz de coeficientes
  .fin para
.fin para

.para M:=N hasta M>=2 hacer
  .para I:=1 hasta I<=M-1 hacer
    .para J:=1 hasta J<=N hacer
      .calculo parcial elementos por pivote
    .fin para
  .fin para
.fin para

.para I=2 hasta I<=N hacer
  .calculo del valor del determinante
.fin para

.para I=1 hasta n hacer
  .calculo del vector de términos independientes
.fin para

.para F:=1 hasta n hacer
  .para I:=1 hasta n hacer
    .para J:=1 hasta n hacer
      .Asignar nuevamente determinante de coeficientes
    .fin para
  .fin para

  .para G=1 hasta n hacer
    .reemplazo de la columna en cuestión por el término independiente
  .fin para

  .para M=N hasta M>=2 hacer
```

```
  .para I=1 hasta I<=M hacer
    .para J=1 hasta J<=N hacer
      .calculo parcial elementos determinante expandido
    .fin para
  .fin para
.fin para

.para I=2 hasta I<=N hacer
  .calculo parcial determinante expandido
.fin para
.fin para
```

Una vez que poseemos los valores de las incógnitas del sistema, contenidas aquí en el vector A, podemos calcular las primeras características y ver si se puede finalizar la iteración. En nuestro caso ese código se ha dejado para incluirlo en el cálculo de tiempos pero no es relevante para la optimización por lo cual se omite aquí.

Una vez finalizados los ciclos de iteración, se procede al cálculo de los demás parámetros característicos de nuestra ala tridimensional. El pseudocódigo muestra que son principalmente dos bucles de cálculo.

```
.para I=1 hasta n hacer
  .para Y=1 hasta n hacer
    .calcular coeficientes de momento de rolido
  .fin para
.fin para
.calcular parámetros tridimensionales
```

Con esta implementación del método (implementación original) se realizan sucesivas ejecuciones y se toman los tiempos de ejecución. Estas sucesivas ejecuciones nos muestran un tiempo total de **116 milisegundos** (0.116 segundos). Esto nos da como resultado una velocidad de 8 fps (frames per second) o ciclos de cálculo del método por segundo. Al mismo tiempo se observa que la resolución del sistema de ecuaciones es de 114 milisegundos, sobre los 116 totales y que los resultados presentan un error promedio del **0.56%.**

Lo primero que se debe notar es que las velocidades de cómputo utilizadas normalmente para los algoritmos de simulaciones de vuelo son 60, 30, 15 fps dependiendo del algoritmo que se trate y sus requerimientos de cómputo, con lo cual estamos muy por debajo de los requerimientos.

2.3. Caso 2: Optimización por computo paralelo (OPENMP)

Como primer paso de optimización incorporamos la librería OpenMP para poder realizar parte de los cálculos en paralelo. Principalmente realizando un análisis del código se pudo identificar el punto a paralelizar dónde se producirían los mayores efectos.

En el siguiente código se muestra el fragmento del código anterior que ha sido modificado. La modificación obedece a que se ha identificado que el bucle externo corresponde a cada una de las incógnitas y el cómputo interno puede hacerse en paralelo.

```
#pragma omp for private(AC)
.para F:=1 hasta n hacer
  .para I:=1 hasta n hacer
    .para J:=1 hasta n hacer
      .Asignar nuevamente determinante de coeficientes
    .fin para
  .fin para

  .para G=1 hasta n hacer
    .reemplazo de la columna en cuestión por el término independiente
  .fin para
```

```
.para M=N hasta M>=2 hacer
  .para I=1 hasta I<=M hacer
    .para J=1 hasta J<=N hacer
      .calculo parcial elementos determinante expandido
    .fin para
  .fin para
.fin para

.para I=2 hasta I<=N hacer
  .calculo parcial determinante expandido
.fin para
.fin para
```

De esta modificación hemos realizado varias pruebas cambiando la cantidad de threads mediante la variable de entorno OMP_THREADS_NUM de OpenMP observando que no se obtenía ninguna mejora fuera del valor de 2 threads.

Este caso arrojó un valor de **115 milisegundos**, con **8 fps** y un error del 0.56%.

También se han intentado realizar cómputo en paralelo sobre otras secciones del código (afectando los otros ciclos for existentes), pero se ha observado en todas ellas que el tiempo no pudo ser mejorado sino que por el contrario se mantenía o incluso empeoraba, por lo menos para el caso de 2 procesadores que era el hardware disponible.

De todas formas hasta aquí nos encontramos con tiempos que están alejados de los requeridos con lo cual pasamos a intentar optimizar utilizando LAPACK de forma de buscar optimizar el cálculo en si mismo.

2.4. Caso 3: Optimización por ATLAS implementación secuencial

El hecho de incorporar ATLAS significó cambios tanto en el programa de pruebas como en nuestra clase de cálculo desde el punto de vista de como se declaraban las matrices con los datos, además de la necesidad de modificar el proyecto para incorporar las librerías.

La utilización de esta librería se centró básicamente en el reemplazo del código que realizaba la resolución del sistema de ecuaciones por la función clapack_dgesv. A continuación se muestra en forma simplificada cómo quedó el código final:

```
.para I:=1 hasta n hacer
  .para J:=1 hasta n hacer
    .inicializar elemento I,J de la matriz de coeficientes
  .fin para
.fin para
.invocamos a clapack_dgesv
.para I=1 hasta n hacer
  .para Y=1 hasta n hacer
    .calcular coeficientes de momento de rolido
  .fin para
.fin para
.calcular parámetros tridimensionales
```

Se ha ejecutado el nuevo código y en este caso el tiempo de cómputo total es de **6 milisegundos** lo que nos da **167 fps** y un error estimado en el **0%** (Ambos términos de la ecuación por separados resultan iguales).

2.5. Caso 4: Optimización por ATLAS implementación paralela

En este caso lo que se introduce es la versión para múltiples hilos de ejecución de la librería ATLAS de forma que la función clapack_dgesv sea procesada en paralelo. No se realizan cambios en el código sino que lo que cambia es la versión de la librería utilizada.

En este caso se observa que los tiempos no se han modificado significativamente siendo de 5 milisegundos el tiempo total y de 0.8 milisegundos para el tiempo de ejecución de clapack_dgesv. Por esto cual la incorporación del cómputo en paralelo de ATLAS no ha introducido ninguna optimización.

2.6. Caso 5: Optimización por ATLAS y OMP

Considerando que la utilización de ATLAS en su versión con soporte de múltiples hilos, no ha producido mejoras, vamos a utilizar la versión secuencial y probaremos ahora la incorporación de OpenMP. Para esto se incorpora código para paralelizar cada uno de los ciclos "for" que sean posibles.

El resultado de la ejecución en este caso es de 6 milisegundos totales y 1 milisegundo asociados a dgesv.

Debido a que los tiempos no han sido mejorados, tomaremos como mejor caso el número tres.

3. RESULTADOS

Hasta aquí hemos analizado la optimización mediante el uso de librerías de cálculo algebraico y/o el uso de cómputo en paralelo. En Tabla 4 se resumen los resultados obtenidos de las distintas combinaciones e intentos de optimización.

Caso	Tiempo (miliseg)	Diferencia o mejora s/original
01-Original	116	--
02-Optimización con OpenMP	115	1 %
03-Optimización con ATLAS	6	95 %
04-Optimización con ATLAS PT	5	96 %
05-Optimización con ATLAS y OpenMP	6	95 %

Tabla 4: Resultados Obtenidos

Aquí podemos observar que el uso de OpenMP para ejecutar parte del algoritmo en forma paralela no ha generado una disminución significativa del tiempo de proceso, pero igual se encontró limitado a la implementación en si, que en nuestro caso utilizaba el método de pivotes para resolver el determinante del sistema de ecuaciones, agregando a esto que el código no poseía ni siquiera optimizaciones a nivel del lenguaje.

Como resultado final se pudo observar que el uso de librerías como ATLAS y las optimizaciones que estas poseen genera una reducción en el tiempo de ejecución del 95%.

4. DISCUSION

De los resultados obtenidos se desprende que el uso de ATLAS nos permite disminuir los tiempos de procesamiento. Lo que se debe determinar ahora, es que habiendo seleccionado ese caso, realizar un análisis de tiempos más fino, para ver en qué lugar del algoritmo se consume la mayor cantidad del tiempo de procesamiento y si es posible optimizar aún más esos tiempos.

Para esto lo que se identificaron tres partes principales en el método que resolvía el método de Glauert.

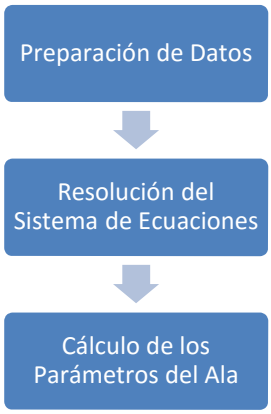

Figura 4: Flujo de Implementación del Método de Glauert

Realizamos una modificación del programa para poder computar los tiempos que insume el procesamiento de cada uno de estos bloques lógicos. Los tiempos obtenidos son:

- Tiempo de Ejecución: 6 milisegundos
- Tiempo Resolución del Sistema: 1 milisegundo (918)
- Tiempo Preparación: 3 milisegundos(2701)
- Tiempo Iteración: 1 milisegundo(1318)
- Tiempo Finalización: 1 milisegundo(952)

Se observa que el mayor tiempo de procesamiento se está consumiendo en el bloque asociado a la preparación de los datos. Para la optimización del mismo lo que utilizamos es la paralelización de dicho bloque como se puede observar en el siguiente código.

```
#pragma omp parallel default(shared) private(aux)
.inicio parallel

#pragma omp for
.para I:=1 hasta n hacer
  .para J:=1 hasta n hacer
    .calculo de coeficientes
  .fin para
.fin para

#pragma omp for
.para I:=1 hasta n hacer
    .preparacion vector A
.fin para

.fin parallel
```

Por último realizamos la ejecución de ésta optimización. Los tiempos obtenidos son:

- Tiempo de Ejecución: 4 milisegundos (250 fps)
- Tiempo Resolución del Sistema: 1 milisegundo (932)
- Tiempo Preparación: 1 milisegundos(1492)
- Tiempo Iteración: 1 milisegundo(1331)
- Tiempo Finalización: 1 milisegundo(955)

Vemos que la utilización de OpenMP para realizar el cómputo en paralelo de los ciclos correspondientes a la preparación de los datos ha resultado en una reducción del tiempo de cómputo de dichos bucles.

Una alternativa más que se ha intentado es el uso del parámetro “nowait” en las sentencias pragma de OpenMP como se observa en el código siguiente.

```
#pragma omp parallel default(shared) private(aux)
.inicio parallel

#pragma omp for nowait
.para I:=1 hasta n hacer
  .para J:=1 hasta n hacer
    .calculo de coeficientes
  .fin para
.fin para

#pragma omp for nowait
.para I:=1 hasta n hacer
    .preparación vector A
.fin para

.fin parallel
```

Los resultados obtenidos no presentan mejoras en los tiempos.

5. CONCLUSIONES

Las pruebas realizadas aquí han sido orientadas a la optimización del algoritmo a nivel del uso de múltiples procesadores que comparten memoria. Esto presenta ventajas para el caso de Programación Orientada a Objetos (OOP) ya que el uso de múltiples hilos de ejecución elimina los problemas del multi-proceso, como en el caso específico de utilizar patrones de diseño como Singleton.

Si bien el método de Glauert no implica un algoritmo complejo, ya que se limita a la resolución de un sistema de n ecuaciones más el uso de los coeficientes obtenidos en sendas sumatorias para obtener los parámetros característicos objeto del método, el requerimiento de utilizarlo en tiempo real nos obliga a reducir el tiempo de cómputo al mínimo.

Se ha podido observar que el uso de cálculo paralelo no ha incorporado una gran diferencia en los tiempos, esto puede ser explicado por la utilización de sesenta estaciones (matrices de 60 x 60) que no justifican el costo en cuanto a tiempos de cómputo para creación de los hilos de ejecución. Al mismo tiempo se ha observado que la utilización de librerías como ATLAS que se encuentran optimizadas para el uso de los recursos que brinda el microprocesador como son los conjuntos de instrucciones SSE. Justamente, el uso de estas librerías nos ha permitido una optimización del tiempo de cómputo mayor al 90% respecto a la implementación original que podríamos denominar clásica, por lo tanto si consideramos un bucle de 8 Hz como era el original, contra uno de 250 Hz resultado de la optimización, pasamos a tener un escenario donde el método de cálculo se hace práctico para ser aplicado en simulación en tiempo real. La base para tal afirmación es que la implementación de simuladores de vuelo como los referenciados en [7], poseen ciclos de cálculo en frecuencias de 60, 30, 15 Hz, una velocidad de 8 Hz para una porción del código a emplear sería de gran importancia, mientras que 250 Hz implica una menor porción del ciclo del simulador (menor al 25%).

También es importante notar que gracias a la incorporación de OpenMP en el bloque de preparación de datos se ha podido llegar a esa velocidad de cómputo con lo cual se hace evidente que la mezcla de varias formas de optimización y el análisis del lugar correcto del código donde aplicarlos determinarán los resultados finales en cuanto a tiempo de cómputo. El uso de hilos de ejecución nos permite la mejor explotación de los procesadores instalados en la máquina en que se realizan las pruebas.

Hasta aquí se ha estudiado la optimización del método de Glauert a nivel de nodo para ejecutar en múltiples procesadores, mediante el uso de hilos de ejecución (threads). El trabajo futuro a plantear sería preguntarnos si lograríamos mejoras mediante el cómputo paralelo del método dentro de un cluster distribuyendo tareas en varias máquinas. La respuesta puede ser objeto de un estudio futuro, pero en realidad casi intuitivamente, por el esfuerzo de cómputo involucrado parecería que no se van a lograr mejoras sustanciales, quizás por el contrario, la

transmisión de los datos y la carga adicional para el procesamiento distribuido generarán un aumento de los tiempos de cómputo.

El análisis de optimización para ejecución en clusters podría proponerse desde otro punto de vista: La simulación de vuelo exige la ejecución de varias instancias del método de Glauert, que como mínimo implicarían la ejecución del método para calcular los parámetros del ala y una segunda para los parámetros del estabilizador horizontal. Obviamente cada una ya optimizada a nivel de nodo. En este caso se podría realizar un estudio lanzando un proceso para cada una de las instancias, con los datos de configuración de entrada locales a cada máquina en cuestión (una para ala y una para estabilizador) de forma que no necesiten ser transmitidos. Con esto los datos a ser transmitidos serían los de interés y no las matrices y vectores en si. Este tipo de distribución de procesamiento involucra al utilización de servicios como los que brinda OpenMPI (www.open-mpi.org).

Para finalizar, parte del trabajo futuro en este campo que se desprende de lo visto en este trabajo, es que arquitecturas de simuladores históricamente utilizadas como DARTS (Distributed Architecture for RealTime Simulation) de Boeing y AVSM (Air Vehicle Simulation Model) de NASA [7] podrían ser adaptadas para ser ejecutadas en Clusters mediante el uso de OpenMP y OpenMPI.

REFERENCIAS

[1] V. Caballini y otros, *"Predicción de la Distribución de Sustentación y Resistencia Aerodinámica en Alas con Asimetría Geométrica, Aerodinámica y/o Funcional"*, Universidad Tecnológica Nacional, Facultad Regional Haedo, 1994.

[2] R. E. Scraton, *"Métodos Numéricos Básicos: Introducción a las Matemáticas Numéricas con Base en la Microcomputadora"*, ISBN 968-451-953-2, 1984.

[3] *Creando un Programa Paralelo en OpenMP* [online]. Available: http://pintucoperu.wordpress.com/2008/09/27/creando-un-programa-paralelo-en-openmp/

[4] *Guide into OpenMP: Easy multithreading programming for C++* [online]. Available: http://bisqwit.iki.fi/story/howto/openmp/

[5] *OpenMP Sections* [online]. Available: https://computing.llnl.gov/tutorials/openMP/#SECTIONS

[7] Karl-Erik Sjôquist, *"Design of a Flight Simulator Sofware Architecture"*, Master Thesis, School of Mathematics and Systems Engineering, Vâxjô University, Suecia, 2009.