Sistemas complejos en maquinas paralelas

TRABAJO PRÁCTICO DE FIN DE CURSADA

UNIVERSIDAD DE BUENOS AIRES

ESCRITO POR

ALEJANDRO MARTÍN VENTURA

25 DE JULIO DE 2017

ÍNDICE

1. Introducción

Los flujos son governados por ecuaciones diferenciales parciales, que representan las leyes de conservación de masa, momenyo y energía. La dinamica de fluidos computacional se encarga de resolver esas ecuaciones diferenciales utilizando tecnicas de analisis numerico. Las computadoras son utilizadas para realizar los calculos requeridos para simular la interacción entre liquidos, gases y superficies definidas por las condiciones de borde. Disponer de mas poder computacional es util para disminuir el tiempo requerido para realizar las simulaciónes, o aumentar la calidad de los resultados.

Para poder aumentar el poder computacional disponible, se utilizan a menudo, tecnicas de computacion en paralelo. La computacion en paralelo consiste en la realización de varios calculos, o la ejecución de varios procesos de forma simultanea. Para poder aprovechar esta forma de caclulo, los problemas grandes deben ser divididos en problemas mas pequeños que puedan resolverse independientemente, o con le intercambio de solo pequeñas cantidades de informacion entre los agentes que resuelven cada parte.

Para facilitar el trabajo de la programación de una solución paralela, se utilizara en este trabajo MPI (message passing interface). MPI es un sistema de pasaje de parametros estandarizado y portable, que puede ser utilizado en una gran variedad de arquitecturas paralelas, y tiene un uso muy difundido en el campo de la computación de alto rendimiento.

En este trabajo se realizara una simulación de las ecuaciones de Navier Stokes bidimensionales. Concretamente se resolverá el problema del flujo interno en un contenedor rectangular, en cuyo centro se encuentra situada una helice que perturba el fluido creando así un campo de velocidades. Esta simulación es de particular interes para aquellos que trabajen con dispositivos similares, sean estos reactores químicos, enfriadores, o torres de mezclado.

2. Modelo

Las ecuaciones de Navier Stokes son ecuaciones diferenciales en derivadas parciales, no lineales e inhomogéneas. Ademas son parabólicas, ya que tienen un termino difusivo distinto de cero.

En nuestro caso trabajaremos con las ecuaciones para fluidos incompresibles.

$$\nabla \cdot v_e = 0$$
$$\frac{\partial v_e}{\partial t} + (v_e \cdot \nabla)v_e = -\frac{1}{\rho}\nabla p + \nu\nabla^2 v_e$$

Donde p es la presión, v la velocidad, ρ la densidad, y ν la viscosidad. La primera ecuación define la conservación de la masa para densidad constante ρ. Los valores que se utilizaran para cada una de estas constantes son 0.01 para la viscocidad, 1 para la densidad, ya que son consistentes con los valores para el agua en unidades del sistema internacional. Para acoplar la velocidad y la presión debemos realizar dos pasos. Primero aplicamos divergencia a ambos miembros de la segunda ecuación aplicando luego la primera ecuación sobre el resultado. Se obtiene el siguiente sistema de ecuaciones, donde la primera ecuación representa la velocidad en la dirección de u, y la segunda la velocidad en la dirección de v. Es decir, $v = (u, v)$. Ademas agregamos a estas ecuaciónes, los terminos F_u y F_v, por ahora genericos, que corresponderan a la fuerza ejercida por la helice en las componentes u y v.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + F_u$$
$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + F_v$$
$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -\rho(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y})$$

Las ecuaciones están ahora parcialmente acopladas, pasaremos ahora a discretizarlas.

3. Discretización

Comenzaremos con algunas definiciones. al modelar con diferencias finitas, se utilizan ciertos reemplazos de los operadores diferenciales conocidos como discretizaciónes. Como su nombre indica, estos son versiones discretas de los operadores, y se los usa bajo el supuesto de que en el limite se comportan de forma similar. Pasaremos ahora a definir algunas discretizaciónes que serán utilizadas en al hacer el pasaje.

Centradas de primer orden:

$$\frac{dU}{dx} = \frac{U^n_{i+1,j} - U^n_{i-1,j}}{2dx}$$

$$\frac{dU}{dy} = \frac{U^n_{i,j+1} - U^n_{i,j-1}}{2dy}$$

$$\frac{dU}{dt} = \frac{U^{n+1}_{i,j} - U^{n-1}_{i,j}}{2dt}$$

Centradas de segundo orden:

$$\frac{d^2U}{dx^2} = \frac{U^n_{i+1,j} - 2*U^n_{i,j} + U^n_{i-1,j}}{dx^2}$$

$$\frac{d^2U}{dy^2} = \frac{U^n_{i,j+1} - 2*U^n_{i,j} + U^n_{i,j-1}}{dx^2}$$

$$\frac{d^2U}{dt^2} = \frac{U^{n+1}_{i,j} - 2*U^n_{i,j} + U^{n-1}_{i,j}}{dt^2}$$

Adelantadas de primer orden:

$$\frac{dU}{dx} = \frac{U^n_{i+1,j} - U^n_{i,j}}{dx}$$

$$\frac{dU}{dy} = \frac{U^n_{i,j+1} - U^n_{i,j}}{dy}$$

$$\frac{dU}{dt} = \frac{U^{n+1}_{i,j} - U^n_{i,j}}{dx}$$

Atrasadas de primer orden:

$$\frac{dU}{dx} = \frac{U^n_{i,j} - U^n_{i-1,j}}{dx}$$

$$\frac{dU}{dy} = \frac{U^n_{i,j} - U^n_{i,j-1}}{dy}$$

$$\frac{dU}{dt} = \frac{U^n_{i,j} - U^{n-1}_{i,j}}{dx}$$

Reemplazando estas discretizaciónes en las ecuaciones semi acopladas de Navier Stokes y obtenemos:

$$\frac{u^{n+1}_{i,j} - u^n_{i,j}}{\Delta t} + u^n_{i,j}\frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x} + v^n_{i,j}\frac{u^n_{i,j} - u^n_{i,j-1}}{\Delta y} = -\frac{1}{\rho}\frac{p^n_{i+1,j} - p^n_{i-1,j}}{2\Delta x} + \nu\left(\frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{\Delta x^2} + \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{\Delta y^2}\right) + Fu$$

$$\frac{v^{n+1}_{i,j} - v^n_{i,j}}{\Delta t} + u^n_{i,j}\frac{v^n_{i,j} - v^n_{i-1,j}}{\Delta x} + v^n_{i,j}\frac{v^n_{i,j} - v^n_{i,j-1}}{\Delta y} = -\frac{1}{\rho}\frac{p^n_{i,j+1} - p^n_{i,j-1}}{2\Delta y} + \nu\left(\frac{v^n_{i+1,j} - 2v^n_{i,j} + v^n_{i-1,j}}{\Delta x^2} + \frac{v^n_{i,j+1} - 2v^n_{i,j} + v^n_{i,j-1}}{\Delta y^2}\right) + Fv$$

$$\frac{p^n_{i+1,j} - 2p^n_{i,j} + p^n_{i-1,j}}{\Delta x^2} + \frac{p^n_{i,j+1} - 2*p^n_{i,j} + p^n_{i,j-1}}{\Delta y^2} = \rho\left[\frac{1}{\Delta t}\left(\frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\Delta y}\right)\right.$$

Aquí en la ultima ecuación podemos ver que no se reemplazó directamente cada operador mediante las ecuaciones de discretización, sino que se agregó un termino temporal, sin que hubiera en principio información sobre el tiempo en la ecuación de la presión. Este cambio se hace con el objetivo de terminar de acoplar la ecuación de la presión con las ecuaciones de velocidad. La derivación de esta solución no se presentará en este trabajo.

Cabe aclarar que al discretizar, se puede modelar el sistema mediante un método implícito o explicito. Un método implícito, o parcialmente implícito, incluiría una ponderación entre los valores de las variables en la iteración n, y la iteración n+1. En este trabajo utilizaremos un método explicito, ya que el sistema de ecuaciones determinado por un método explicito es lineal, y resulta en relaciones donde un elemento en la iteración n+1 depende de otros en la iteración n, pudiendo entonces realizarse los reemplazos en las matrices que representan el sistema de forma directa, y resultando así en una implementación mas sencilla. Un método implícito da como resultado un sistema no lineal, en el cual hay que hacer uso de algún método de resolución de sistemas no lineales, como punto fijo, lo cual aumenta la complejidad de la implementación.

4. Condiciones de contorno

El sistema fisico esta compuesto por un recipiente rectangular que contiene un liquido dentro. Ademas una helice en el centro, cuyas aspas son rectas y de exactamente la misma longitud a distintas alturas, perturba el liquido produciendo en el cambios de velocidad. Para la simulación se aprovechara la simetria del problema para modelarlo en dos dimensiones. Se tomara entonces un corte horizontal del reactor. El sistema, como sera modelado, consiste entonces de un plano horizontal, cuyos vértices y aristas representan las paredes del recipiente, un espacio dentro que representa el fluido, y segmentos en el centro que representan la posición de las aspas de la helice. Definir la cantidad de aspas en la helice es sencillo, basta con cambiar las condiciónes en un condicional que logra que se trate distinto a los elementos de dluido donde deberia estar un aspa. En el presente trabajo se realizó la experimentación con una unica aspa ya que esto aporta mayor claridad a la hora de evaluar el comportamiento del fluido.

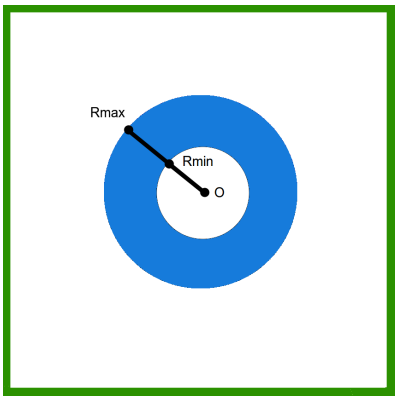

FIGURA 1. Podemos ver en azúl el área barrida por la helice, desde R_{min} hasta R_{max} desde el centro O. En verde las paredes del contenedor, y en blanco el resto del fluido presente en el mismo.

Las condiciones de contorno para Las ecuaciones de velocidad no presentan mayores complejidades, se establece la velocidad del fluido en los bordes del contenedor como cero en ambas direcciones, u y v. Estas velocidades no se cambian en las subsecuentes iteraciones del programa ya que representan las paredes del contenedor que se mantienen quietas en todo momento en el sistema físico real. En cuanto a la ecuación de la presión, utilizar una presión de cero en el contorno no seria realista dado que la presencia de una presión muy baja produce la atracción del fluido circundante. Una forma de deducir el valor correcto de la presión en los bordes del contenedor es utilizar las ecuaciones de Newton. Si un elemento de fluido realiza una fuerza en la pared del contenedor, dado que esta no se mueve, devuelve una fuerza igual al elemento de fluido. Realizando este calculo, se llega a una condición de igualdad entre el elemento de borde y el elemento de fluido que se encuentre junto a este. Básicamente lo que ese resultado nos dice es que

el valor de presión para un elemento de borde debe ser el mismo que el del elemento de fluido que esta junto a el, pudiendo entonces solucionar este aspecto de la simulación simplemente copiando el valor del elemento de fluido mas cercano a cada elemento de borde en cada iteración.
Finalmente para las condiciones iniciales, se estableció la velocidad en u y en v en cero para todo el sistema.

5. Implementacion

La implementación fue realizada casi completamente en C++, excepto por el graficador que se escribió en Matlab.

Corriendo de forma no paralela, el programa define las matrices U2, U2, V1, V2, P1, P2, que representan el estado del sistema en una iteración para la velocidad en u, en v, y la presión, y luego estas mismas en la iteración siguiente.

Se definen las condiciones iniciales del problema, y luego se utiliza el metodo explicito explicado anteriormente para calcular los nuevos valores del sistema. Estos son guardados en U2, V2, y P2. Seguido de esto el programa reemplaza los valores de U1, V1, y P1, por aquellos de U2, V2 y P2, quedado así preparado para la siguiente iteración.

Una salvedad es que al realizar el calculo de los nuevos valores, si un elemento forma un angulo respecto del centro del sistema que es congruente con aquel que debería tener el aspa para el tiempo de esa iteración, entonces a ese elemento se le aplica una fuerza de la forma $F = A\delta v^2$, donde A es el área compartida por los elementos, y δv es la diferencia de velocidad, para ese punto, entre el aspa y el fluido. Esta diferencia es calculada restando componente a componente, los elementos correspondientes de las matrices U1 y V1, y la velocidad tangencial del aspa en ese punto, que resulta de multiplicar la velocidad angular por el radio elemento menos el punto central del sistema.

Se implementó también una clase mat2, que representa una matriz, y que contiene un puntero a un arreglo de números de punto flotante de doble precisión y dos enteros que representan el tamaño en filas y columnas de la matriz. Ademas la clase cuenta con funciones que realizan la abstracción de indexar en el arreglo calculando la posición del elemento buscado como la columna pedida, mas la fila pedida multiplicada por la cantidad de columnas. Esta clase también cuenta con una función de impresión que escribe los elementos de la matriz en formato separado por espacios.

En cuanto a la paralelización, como se comentó anteriormente se utilizó MPI. El procedimiento básico es el siguiente.

- Se inicializan el comunicador, cantidad de procesos y demás variables pertinentes a MPI.
- Se calcula una división del espacio de trabajo por filas, para saber de que sección sera responsable cada proceso.
- El proceso cero define las matrices del sistema y sus condiciones iniciales y luego envía la sección horizontal correspondiente de la matriz a cada proceso. Luego calcula la sección que le corresponde a el mismo.
- Los otros procesos reciben las secciones que les corresponden.
- Todos los procesos ejecutan una función encargada de procesar la simulación.
- Dentro de esta función, se encuentra un ciclo en el cual realizan los cálculos pertinentes a la simulación, y al terminar una iteración, cada proceso envía los nuevos valores de los elementos que pertenecen a su sección, pero que son necesarios para que procesos vecinos calculen la siguiente iteración de sus elementos.
- Cada proceso recibe los valores enviados por los procesos vecinos.
- Cada proceso chequea el numero de iteración, y en caso de que sea multiplo de cierto valor prefijado, envía su sección entera al proceso cero para que este imprima. Luego el proceso cero reúne todas las secciones armando las matrices completas.
- Si es el caso, el proceso cero imprime las matrices U y V por la salida estándar.
- Luego de esto el ciclo descripto en el paso seis termina, y se vuelve a ejecutar. En caso de haber llegado al tiempo máximo de la simulación, todos los procesos retornan de la función, y luego de la función principal.

Aquí cabe aclarar algunas cosas. Primero, la impresión realizada por el proceso cero, imprime U y V de un tiempo, y U yV del tiempo siguiente de forma contigua. Es la rutina de Matlab la que se encarga de separar y conferir significado a esos datos.
Por otro lado, como se dijo el trabajo se divide entre los procesos otorgando a cada uno una franja horizontal del sistema, es decir, se le otorga un numero de fila inicial, y un numero de fila final que determinan los rangos en los que el proceso trabajará. Esta división no es optima, ya que otras formas de dividir el trabajo resultan en un menor intercambio de información entre los procesos durante la simulación, que es un factor

importante a la hora de optimizar la velocidad de ejecución del programa. La decisión de utilizar una división horizontal es simplemente por simplicidad del código.

Otro detalle es que si bien se dice que cada proceso toma los valores que le faltan para la iteración siguiente, de los procesos vecinos, en cada iteración. En el caso del proceso cero, o del ultimo, solo interactúan con un proceso vecino, ya que las condiciones de contorno no se modifican, o son calculadas por ellos mismos ya que no dependen de elementos extra.

Ademas, durante la simulación no se crean ni se destruyen matrices, sino que estas son reutilizadas cambiando los valores que contienen para no perder tiempo manejando memoria.

Por ultimo se hablará sobre los valores de la malla, es decir, la división del sistema físico representada por las matrices. Durante el testeo del programa se encontró que la solución a veces puede ser inestable. En particular cuando los valores de dx y dy son muy chicos, o el valor de dt es grande, la simulación se vuelve inestable, acumulando error numérico y desbordando la precisión máxima de los tipos numéricos de C++.

Llegado a este punto el programa emite un mensaje de error, avisando que recibió durante la simulación NaN como resultado de un calculo. Sucedido esto el programa termina, y es responsabilidad del usuario del mismo, aumentando el valor del tamaño de los elementos, haciendo menos fina la malla espacialmente, o disminuyendo el tamaño del salto temporal entre iteraciones, haciendo mas fina la simulación en el tiempo. Cualquiera de estas dos medidas, elegido el valor correcto, resuelve el problema de la estabilidad.

Se aclara también que a veces una simulación se vuelve inestable, pero termina antes de que los valores calculados desborden de la precisión numérica de las variables, en ese caso la simulación sera incorrecta, incluso no habiendo mensaje de error alguno.

Sin embargo, algo bueno de las simulaciones de dinámica de fluidos por diferencias finitas, es que muy a menudo, cuando se vuelven inestables, lo hacen de una forma muy clara, que no se comporta para nada como un fluido, se recomienda entonces hecha un vistazo, siempre que se pueda, a algún gráfico de los resultados de la simulación para comprobar que esta se mantuvo estable.

6. Experimentación

No se comparó la simulación contra un fluido real bajo las mismas condiciones. Sin embargo la simulación parece reflejar correctamente el comportamiento de un fluido, como puede apreciarse en el mapa de velocidad total codificado por color, o en el campo vectorial de las figuras 2 y 3

Como puede verse se aprecia claramente la posición del aspa rotante que empuja el fluido, asi como las ondas producidas por este movimiento. Tambien estas ondas reflejan correctamente en la pared del contenedor, y la dirección de rotación obtenida es la esperada por el empuje de un aspa moviendose en esa dirección.

Fueron realizados estudios de rendimiento, tratando de medir el impacto de la ejecución en paralelo. Como marco teórico de esta sección, se utilizó por un lado la Ley de Amdahl, midiendo así el speedup $S(n,p) = \frac{T_{ser}(n)}{T_{par}(n,p)}$ a trabajo fijo aumentando la cantidad de procesadores, y por otro la Ley de Gustavson para la cual medimos el trabajo realizado por cantidades cada vez mas grandes de recursos de procesamiento a tiempo constante, y luego calculamos la eficiencia $E(n,p) = \frac{S(n,p)}{p}$. Donde $T_{ser}(n)$ es el tiempo que tarda el programa en su versión serial para una entrada de tamaño n, y $T_{par}(n,p)$ es el tiempo que tarda el programa paralelo para una entrada de tamaño n y cantidad de procesos p.

6.1. Ley de Amdahl.

La Ley de Amdahl nos da el speedup teórico en la ejecución de una tarea que consta de una cantidad de trabajo fijo al incrementar los recursos del sistema.

Llevada al limite, sirve para calcular la mejora máxima posible para una tarea que consta de una parte paralelizable, y una parte serial que no puede ser efectivamente paralelizada.
La formulación matemática de la Ley de Amdahl es la siguiente:

$$S(s) \leqslant \frac{1}{(1-p)+\frac{p}{s}}$$

Donde S es el speedup total, s el speedup de la parte del programa que se favorece por el paralelismo, y p es la proporción de tiempo que era ocupada por la parte del programa que tiene speedup alguno. Se asume que todo el tiempo que no corresponde a

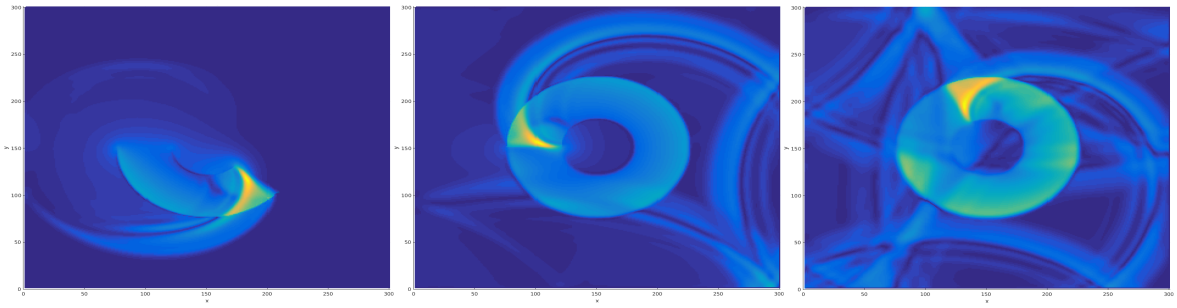

FIGURA 2. Estos mapas de calor muestran la evolución del fluido en el tiempo.

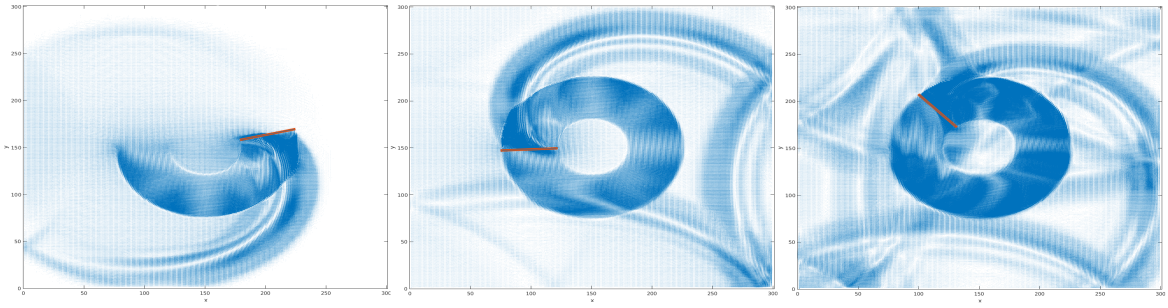

FIGURA 3. El aspa (rojo) empuja al fluido al rotar.

p, o sea 1-p, se mantiene igual. Un resultado directo de esta ley es que incluso utilizando infinitos recursos, no puede aumentarse el speedup mas que:

$$S(s) \leqslant \frac{1}{1-p}$$

Para este experimento se quiere dejar fija la cantidad de trabajo total realizado, y medir como cambia la velocidad al agregar unidades de procesamiento.

El programa fue ejecutado en una red ethernet con 21 maquinas, cada una disponiendo de 4 núcleos. Al momento de realizar la experimentación, estas maquinas estaban siendo utilizadas, a razón de dos nucleos por maquina, con lo cual se disponía de 42 núcleos para procesar la prueba. Los speedups resultado son los siguientes:

Speedup

#CPU	S(n,p)
30	19.8541013255
25	19.2648777129
20	17.4229950177
15	15.4322650675
12	12.3897738475
10	10.5198631716
9	9.1780910351
8	8.434695922
6	6.0758387804
5	5.0493848725
3	2.5275473744
1	1

Los datos de esta tabla pueden ser apreciados en la figura 4

6.2. Ley de Gustavson.

La Ley de Gustafson calcula el speedup teórico para una tarea de tiempo de ejecución fijo al incrementar los recursos de un sistema. Al aplicar la Ley de Gustavson, lo que varia no es el tiempo de ejecución, sino la cantidad de trabajo realizado. La formulación matemática de la Ley de Gustavson es la siguiente:
$S(s) = p - \alpha(p-1)$
Donde S es el speedup, p es el número de procesadores, y α la parte no paralelizable del proceso. Notar que si se aumenta notablemente la el trabajo total en la sección paralelizable, haciendo que la

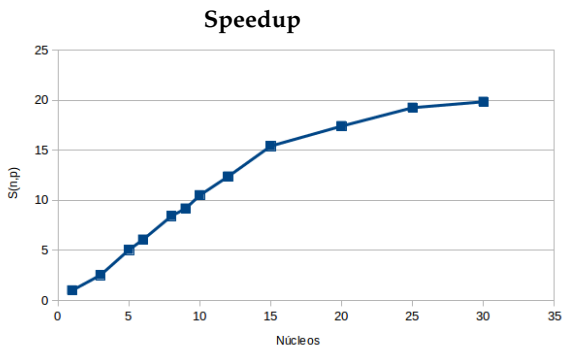

FIGURA 4. El speedup tiende a disminuir al aumentar mucho la cantidad de nucleos, perdiendo approximadamente un tercio del rendimiento al alcanzar los 30.

proporción de trabajo serial α sea muy pequeña, esto resulta en algo muy similar a $S(s) = p$
Segun Guftavson, en su trabajo de 1988, tiene mucho mas sentido hablar de tiempo fijo que de tamaño fijo. Esto es así porque en la practica los timepos utilizados para realizar simulaciónes no varian tanto como los tamaños o cantidad de recursos utilizados para las mismas. Siguiendo este espiritu, en este experimento se quiere dejar el tiempo limite en 120s, y medir cuanta utilidad, o trabajo neto, pudo extraerse para distintas cantidades de unidades de procesamiento.
Para lograr esto, programa fue nuevamente ejecutado en una red ethernet con 21 maquinas, cada una disponiendo de 3 núcleos. Al momento de realizar la experimentación, estas maquinas estaban siendo utilizadas, a razón de un núcleo por maquina, con lo cual se disponía de 63 núcleos para procesar la prueba.

Como medida de trabajo efectivo, se contó la cantidad de elementos del sistema calculados, ya que es lo que da utilidad. Se aclara que cuando en la tabla de resultados figura que se utilizó un solo núcleo, esa medición fue realizada sobre la versión serial del programa, no perdiendo así tiempo inicializaciónes o calculos necesarios para paralelizar. Los resultados son los siguientes, que tambien pueden apreciarse en la figura 5. Tambien se calcula el trabajo normalizado por nucleo, obteniendose asi los datos que se ven en la figura 6

Trabajo

#CPU	elementos calculados
1	111.113 Millones
5	413.814 Millones
20	1976.24 Millones
35	3198.04 Millones
50	3661.31 Millones
60	4294.03 Millones

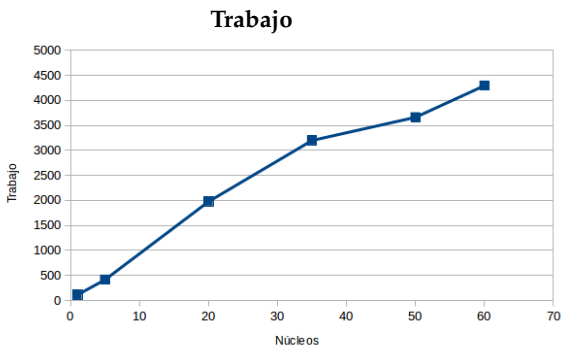

FIGURA 5. el trabajo neto calculado se comporta de forma aproximadamente lineal.

Los datos parecen indicar un comportamiento que escala correctamente con la cantidad de núcleos. En la siguiente tabla sin embargo, se muestra el trabajo por nucleo, y se ve que decrece con el aumento de los recursos.

Trabajo por núcleo

#CPU	trabajo/núcleos
1	111.113
5	82.7628
20	98.8120
35	91.3725
50	73.2262
60	71.5671

En particular, si normalizamos los datos respecto del trabajo realizado por la version serial, notamos una reducción del rendimiento de la simulación que llega a bajar hasta un 64 % respecto del rendimiento extraido por un solo núcleo.

Trabajo normalizado

#CPU	trabajo/(núcleos*111.113)
1	1
5	0.7448
20	0.8892
35	0.8223
50	0.6590
60	0.6440

A primera vista, parece que se pierde mucho rendimiento. En realidad, si tenemos en cuenta que la corrida de un solo nucleo fue realizada con una versio distinta que no incluye codigo de MPI, se entiende que ese valor aparezca como un outlier. Dicho esto, lo que esperariamos ver si el programa escalara, seria una recta horizontal. Si bien no es el resultado obtenido, no se dista mucho. Lo importante es que encontramos que se puede aumentar la cantidad de trabajo realizable significativamente agregando unidades de procesamiento, lo cual es un resultado mas optimista que el insinuado por la Ley de Amdahl.

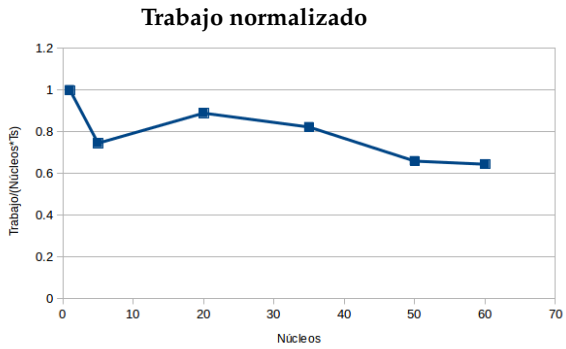

FIGURA 6. Al normalizar se nota mas facilmente la desviación de la linealidad.

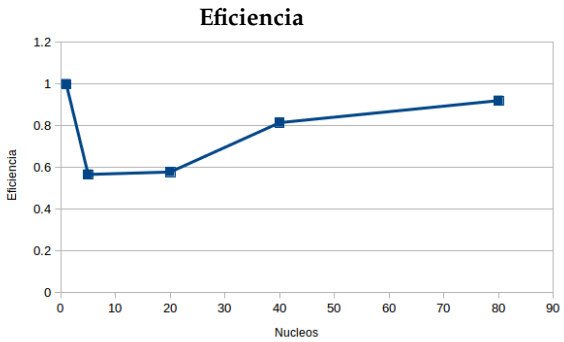

FIGURA 7. La eficiencia esta normalizada en función de la cantidad de núcleos, por lo que es muy sensible a las desviaciones de la linealidad.

Otra medida importante relacionada con la ley de Gustafson, que ya fue mencionada anteriormente es la eficiencia. Con el objetivo de medir la eficiencia se realizó otra experimentación en la cual variara la cantidad de nucleos y el tamaño del problema, de forma tal de poder calcular el tiempo de ejecución y asi, la eficiencia. Recordemos que la definición de eficiencia es $E(n,p) = \frac{S(n,p)}{p}$.
Los tiempos conseguidos en esta etapa de la experimentación son los siguientes.

Tiempos

#CPU	raíz(2,n)	Tp	Ts
80	100	105.000	7731.128
40	75	74.730	2435.544
20	50	61.764s	713.772
5	50	252.129	713.772
1	1	0.366	0.366

Con lo cual la eficiencia en funcion de la cantidad de núcleos sería:

Eficiencia

#CPU	eficiencia
80	0.920372381
40	0.8147812124
20	0.5778220323
5	0.5661958759
1	1

Si vemos la figura 7, encontramos devuelta que si bien la eficiencia no es optima, el resultado es mucho mejor que el esperado al seguir el paradigma planteado por la ley de Amdahl. Tambien puede verse nuevamente la diferencia entre la version serial y la version paralela para tareas de poco trabajo en el primer punto del grafico. Como nota aparte, la eficiencia es una medida mucho mas sensible de la escalabilidad que el speedup, como podemos apreciar si comparamos este ultimo grafico, con el grafico de speedup (figura 8) de los mismos datos.

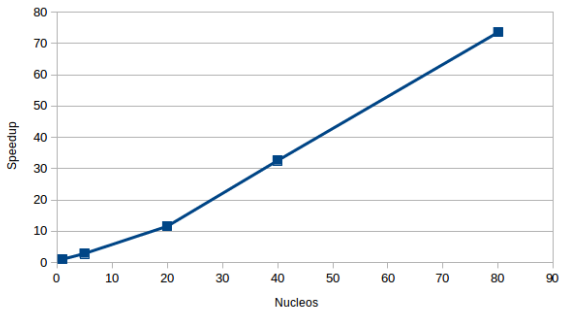

FIGURA 8. Al graficar el speedup, se nota mucho menos el comportamiento no lineal.

7. CONCLUSIÓN

Los resultados obtenidos sugieren que, como es sugerido por la Ley de Amdahl, intentar agregar unidades de procesamiento con la esperanza de reducir el tiempo de procesamiento que toma un trabajo dado es eficiente solo hasta cierto punto. La sección serial del programa pasará a dominar el tiempo de ejecución y no se podrá disminuir lo que toma en terminar.

Por otro lado, notamos que es beneficioso aumentar la carga de trabajo paralelo, ya que así el porcentaje de tiempo de procesador que se gasta en procesar datos aumenta cada vez mas, mientras que en comparación el tiempo utilizado en procesar la porción serial permanece pequeño. Esto nos indica que los tipos de procesamiento que se verán beneficiados del paralelismo son aquellos donde podamos incrementar fuertemente los cálculos realizados por la sección paralela, siendo las simulaciones realizadas mediante diferencias finitas un buen ejemplo de esto último.