DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Final

Modelo Navier Stokes 2D

10 de febrero de 2018 — Organización del Computador II

Integrante	LU	Correo electrónico
Ventura, Martín Alejandro	249/11	venturamartin90@gmail.com
Muiño, María Laura	399/11	mmuino@dc.uba.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

1. INTRODUCCIÓN

Los flujos son gobernados por ecuaciones diferenciales parciales, que representan las leyes de conservación de masa, momento y energía. La dinámica de fluidos computacional se encarga de resolver esas ecuaciones diferenciales utilizando técnicas de análisis numérico. Las computadoras son utilizadas para realizar los cálculos requeridos para simular la interacción entre líquidos, gases y superficies definidas por las condiciones de borde. Disponer de mas poder computacional es útil para disminuir el tiempo requerido para realizar las simulaciones, o aumentar la calidad de los resultados.

Para poder aumentar el poder computacional disponible, se utilizan a menudo, técnicas de computación en paralelo, o de computación vectorial. En este trabajo nos centraremos en la tecnología de computación vectorial SIMD (Single instruction multiple data) de Intel.

Concretamente se desarrollará codigo en assembler que utilizando las instrucciónes de vectorización de los procesadores Intel logre un aumento de rendimiento. Luego se comparará ese aumento de rendimiento con tecnicas automaticas de vectorización o paralelización, tales como OpenMP, una api para el procesamiento multinucleo con memoria compartida, y las optimizaciónes disponibles en los compiladores ICC (Intel C compiler) y g++, que a su vez utilizan instrucciónes SIMD.

Hay diversos problemas de flujo conocidos que son utilizados frecuentemente para testear aplicaciónes de este estilo. En este trabajo utilizaremos cavity flow y channel flow. describirlos

Se realizó una solución iterativa de punto fijo, creo. Por que? No hay iteracion de punto fijo en el codigo.

2. DESARROLO

2.1. Discretización. Comenzaremos con algunas definiciones. al modelar con diferencias finitas, se utilizan ciertos reemplazos de los operadores diferenciales conocidos como discretizaciones. 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 discretizaciones 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 discretizaciones en las ecuaciones semi-acopladas de Navier Stokes y obtenemos:

$$\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta t}+u_{i,j}^{n}\frac{u_{i,j}^{n}-u_{i-1,j}^{n}}{\Delta x}+v_{i,j}^{n}\frac{u_{i,j}^{n}-u_{i,j-1}^{n}}{\Delta y}=$$
$$-\frac{1}{\rho}\frac{p_{i+1,j}^{n}-p_{i-1,j}^{n}}{2\Delta x}+\nu\left(\frac{u_{i+1,j}^{n}-2u_{i,j}^{n}+u_{i-1,j}^{n}}{\Delta x^{2}}+\frac{u_{i,j+1}^{n}-2u_{i,j}^{n}+u_{i,j-1}^{n}}{\Delta y^{2}}\right)+Fu$$

$$\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta t}+u_{i,j}^{n}\frac{v_{i,j}^{n}-v_{i-1,j}^{n}}{\Delta x}+v_{i,j}^{n}\frac{v_{i,j}^{n}-v_{i,j-1}^{n}}{\Delta y}=$$
$$-\frac{1}{\rho}\frac{p_{i,j+1}^{n}-p_{i,j-1}^{n}}{2\Delta y}+\nu\left(\frac{v_{i+1,j}^{n}-2v_{i,j}^{n}+v_{i-1,j}^{n}}{\Delta x^{2}}+\frac{v_{i,j+1}^{n}-2v_{i,j}^{n}+v_{i,j-1}^{n}}{\Delta y^{2}}\right)+Fv$$

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

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.

2.2. Implementación. La implementación fue realizada casi completamente en C++, excepto por la sección donde es critico el rendimiento, la cual fue programada en C+ y Assembler. Esta sección es la correspondiente a la función calcVelocities, que como su nombre indica, calcula las velocidades en cada punto.

Corriendo de forma no vectorizada, 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 un método explicito 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.

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 vectorización, como se comentó anteriormente se utilizó la tecnología SIMD de Intel, que fue utilizado de la siguiente forma:

- Mediante una directiva DEFINE, se elije si se desea compilar con soporte para SIMD, soporte para OpenMP, ambos, o ninguno.
- El programa inicializa las matrices necesarias con los valores iniciales segun lo estipulado por el metodo de discretización utilizado.
- La sección del programa que realiza el calculo consta de tres ciclos for consecutivos. El primero cicla en la variable t, que representa el tiempo. el segundo en la variable i, que representa la altura, y el tercero en la variable j que representa el ancho.
- Mediante la utilización de las directivas de compilador, el codigo compilado constara de una implementación en C++ plano, Una implementación SIMD, donde al llegar a un valor menos al ancho de los registros XMM dividido por el tamaño de el tipo de datos flotante de precision simple se cambia el procesamiento mediante SIMD por el de C++, y ademas mediante estas mismas directivas puede definirse o no la presencia de OpenMP, logrando asi la utilización de multiples nucleos.
- La paralelización mediante OpenMP se realiza en la variable i.
- La vctorización mediante SIMD, se realiza mediante la variable j. Es decir, en un solo llamado a la versión de assembler de la funcion de calculo se calculan 4 elementos.
- 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.

2.3. Experimentacion(version borrador). Usando la herramienta objdump sobre los archivos objeto (.o) del código de c++ (sin flags de optimización), obtuvimos y analizamos el código ensamblado por el compilador. Cosas que notamos

- Dentro de la función calcVelocities, hay calls a distintas secciones de la función misma que no son necesarios, ya que saltan a una línea consecutiva.
- Hay consultas a memorias innecesarias (pide un valor a memoria, que no se pisa y luego de varias operaciones, vuelve a pedirlo).
- Se manejan las variables locales almacenandolas en la pila, mientras que sólo se usan los registros de manera auxiliar para realizar operaciones

Probando compilar el código de c++ con el flag de optimización -O1 se obtuvieron los siguientes resultados respecto al código sin flags de optimización:

- El código en asembler obtenido mediante la herramienta objdump era reducido en cuanto a cantidad de líneas (por ejemplo, la función calcVelocities tenia 558 líneas en assembler sin flags de optimización y luego con el flag 341 líneas).
- El tiempo de compilación aumento con el uso del flag. Los tiempos fueron (real 0m0.693s, user 0m0.644s, sys 0m0.044s) sin flags y (real 0m1.113s, user 0m0.948s, sys 0m0.056s) con el flag.
- El tiempo de ejecución disminuyó (real 0m3.718s, user 0m3.672s, sys 0m0.000s) con el uso del flag (real 0m1.447s, user 0m1.444s, sys 0m0.000s).

3. Conclusión