Trabajo Práctico 2 - Navier Stokes

Organización del Computador II

Segundo cuatrimestre 2017

1. Introducción

En este trabajo práctico se busca una primera aproximación al procesamiento con instrucciones que operan con mutiples datos (SIMD). El objetivo es conocer y comprender los principios de la programación con dichas instrucciones. La familia de procesadores x86-64 de Intel posee una serie de extensiones para operaciones SIMD. En un comienzo, éstas se denominaron MMX (MultiMedia eXtension), luego SSE (Streaming SIMD Extensions) y por último AVX (Advanced Vector Extensions).

El trabajo práctico consiste en implementar un subconjunto de funciones en ASM, para luego evaluar su rendimiento. La evaluación debe ser un análisis exhaustivo de las propiedades del código ejecutado y de las circunstancias que justifican por qué una implementación funciona mejor (o peor) que otra. Trabajaremos con una técnica de simulación de flujo de fluidos basada en las ecuaciones de Navier-Stokes con el objetivo de conseguir un resultado estable y verosímil aunque no sea numéricamente preciso.
El modelado de flujo de fluidos se estudió durante los siglos XVII y XVIII desde un enfoque análitico llegando a desarrollar las llamadas ecuaciones de Navier-Stokes. A partir de 1950 se comenzaron a desarrollar algoritmos numéricos para aproximarse computacionalmente a estos modelos.

2. El modelo

Las ecuaciones de Navier Stokes capturan la dinámica del flujo de fluidos de la siguiente forma:

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \nu\nabla^2 u + f \tag{1}$$

$$\frac{\partial \rho}{\partial t} = -(u \cdot \nabla)\rho + \kappa\nabla^2 \rho + S \tag{2}$$

La velocidad está modelada a través de la equación 1 y la densidad del fluido a través de la ecuación 2. Podemos pensar que la velocidad y la densidad se representan como una progresión de los valores sobre un campo vectorial que tiene en cuenta el estado en el instante actual y las fuerzas aplicadas sobre su dominio para computar el estado inmediato siguiente. Para representar la velocidad utilizaremos cuatro matrices:

- `u`, `u_prev` que representan la velocidad y la velocidad en el paso inmediato anterior en el eje `x` dado un isntante en el tiempo

- `v`, `v_prev` que representan la velocidad y la velocidad en el paso inmediato anterior en el eje `y` dado un isntante en el tiempo
- `dens`, `dens_prev` que representan la densidad y la densidad en el paso inmediato anterior dado un isntante en el tiempo

Las matrices de velocidad son la representación discreta del campo vectorial asociado simplifcado al tomar un vector representante de la grilla obtenida al dividir el espacio en celdas de tamaño fijo. En las matrices con las que vamos a operar se agrega una columna y una fila a cada lado de nuestra grilla para simplificar el trabajo sobre los bordes. A la hora de acceder a un elemento dentro de nuestra matriz (que será representada sobre un espacio lineal de memoria) realizamos el siguiente computo:

```
#define IX(i,j) ((i)+(N+2)*(j))
```

La forma de resolver el estado de nuestro modelo es iterando sobre una cadena de funciones que se aplican sobre las matrices de velocidad y densidad. Cada iteración se corresponde con el paso de un lapso de tiempo capturado por el valor `dt`. Para actualizar la densidad se comienza con un estado inicial (valores iniciales de `dens` y `dens_prev`) y luego se aplican fuerzas (`solver_add_source`), se calcula la difusión discreta de la densidad (`solver_diffuse`) y se actualizan los valores de las matrices de velocidad de forma de conservar la cantidad de masa en el sistema (`solver_advect`).

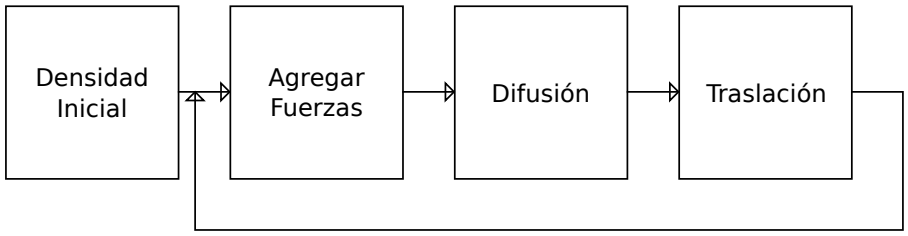

Figura 1: Ciclo de resolución de la densidad

En la figura 1 se exponen los pasos involucrados en la actualización de la matriz de densidad. Los pasos generales para actualizar las matrices de velocidad y densidad se muestran en la figura 2, a continuación detallamos sus componentes.

En el código de la figura 4 vemos la implementación de la función que resuelve el cálculo de difusión de la densidad (figura 3).

En cada paso los valores de cada celda se ven ponderados por los de sus celdas contiguas.

La figura 5 muestra el código que resuelve de forma general el cómputo de los valores borde en las matrices.

La figura 6 muestra el código que resuelve la proyección de los valores en la matriz de velocidad.

3. Funciones a implementar

Se pide que se implementen en lenguaje ensamblador las siguientes funciones:

- `void solver_lin_solve ( fluid_solver* solver, uint32_t b, float * x, float * x0, float a, float c )`
  Que calcula la difusión del fluido a modelar, **se deben utilizar operaciones SIMD**.

```
1
2  void solver_dens_step ( fluid_solver* solver, float * x, float * x0){
3        solver_add_source ( solver, x, x0);
4        SWAP ( x0, x ); solver_diffuse ( solver, 0, x, x0);
5        SWAP ( x0, x ); solver_advect ( solver, 0, x, x0, solver->u, solver->v);
6  }
7
8  void solver_vel_step ( fluid_solver* solver, float * u0, float * v0){
9        solver_add_source ( solver, solver->u, u0); solver_add_source ( solver, solver->v, v0);
10       SWAP ( u0, solver->u ); solver_diffuse ( solver, 1, solver->u, u0);
11       SWAP ( v0, solver->v ); solver_diffuse ( solver, 2, solver->v, v0);
12       solver_project ( solver, u0, v0 );
13       SWAP ( u0, solver->u ); SWAP ( v0, solver->v );
14       solver_advect ( solver, 1, solver->u, u0, u0, v0); solver_advect ( solver, 2, solver->v, v0, u0, v0);
15       solver_project ( solver, u0, v0 );
16 }
```

Figura 2: Código de solver_dens_step y solver_vel_step

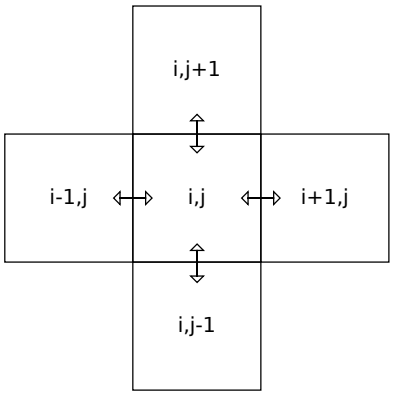

Figura 3: Esquema de intercambio de densidades

- `void solver_set_bnd ( fluid_solver* solver, uint32_t b, float * x )`
  Que calcula los valores para los casos borde de las matrices.
- `void solver_project ( fluid_solver* solver, float * p, float * div )`
  Que proyecta los nuevos valores en la matriz de velocidad, **se deben utilizar operaciones SIMD**.

3.1. Consideraciones

Tener en cuenta las siguientes características generales,

- Interprete el codigo en C y diseñe la forma de realizar los cómputos sobre múltiples datos donde corresponda.
- No se debe perder precisión en los cálculos.
- Las funciones implementadas en ASM deberán operar con la mayor cantidad posible de bytes.
- El trabajo práctico se debe poder ejecutar en las máquinas de los laboratorios.

```
1  void solver_diffuse ( fluid_solver* solver, uint32_t b, float * x, float * x0){
2        float a=solver->dt*solver->diff*solver->N*solver->N;
3        uint32_t i, j, k;
4        for ( k=0 ; k<20 ; k++ ) {
5              for ( i=1 ; i<=solver->N ; i++ ) {
6                    for ( j=1 ; j<=solver->N ; j++ ) {
7                          x[IX(i,j)] = (x0[IX(i,j)] + a*(x[IX(i-1,j)]+x[IX(i+1,j)]
8                                +x[IX(i,j-1)]+x[IX(i,j+1)]))/c;
9                    }
10             }
11             solver_set_bnd ( solver, b, x );
12       }
13 }
```

Figura 4: Código de solver_diffuse

```
1  void solver_set_bnd ( fluid_solver* solver, uint32_t b, float * x ){
2        uint32_t i;
3        uint32_t N = solver->N;
4        for ( i=1 ; i<=N ; i++ ) {
5              x[IX(0 ,i)] = b==1 ? -x[IX(1,i)] : x[IX(1,i)];
6              x[IX(N+1,i)] = b==1 ? -x[IX(N,i)] : x[IX(N,i)];
7              x[IX(i,0 )] = b==2 ? -x[IX(i,1)] : x[IX(i,1)];
8              x[IX(i,N+1)] = b==2 ? -x[IX(i,N)] : x[IX(i,N)];
9        }
10       x[IX(0 ,0 )] = 0.5f*(x[IX(1,0 )]+x[IX(0 ,1)]);
11       x[IX(0 ,N+1)] = 0.5f*(x[IX(1,N+1)]+x[IX(0 ,N)]);
12       x[IX(N+1,0 )] = 0.5f*(x[IX(N,0 )]+x[IX(N+1,1)]);
13       x[IX(N+1,N+1)] = 0.5f*(x[IX(N,N+1)]+x[IX(N+1,N)]);
14 }
```

Figura 5: Código de solver_set_bnd

3.2. Desarrollo

Para facilitar el desarrollo se cuenta con todo lo necesario para poder compilar y probar las funciones a medida que las implementan. Los archivos entregados están organizados de la siguiente forma:

- `src`: Contiene los fuentes del programa principal
- `bin`: Contiene los ejecutables
- `tmp`: Contiene las imágenes de referencia de la cátedra y guarda tambien allí las imágenes generadas por ustedes (tanto de las matrices como de las diferencias)
- `bmp`: Contiene código para manipular archivos BMP, se utiliza para guardar las imágenes de las matrices durante las pruebas

3.2.1. Compilar

Ejecutar `make` desde la carpeta `src`.

3.2.2. Uso

Ejecutando `./bin/demo --help` obtenemos:

```
1
2  void solver_project ( fluid_solver* solver, float * p, float * div ){
3        uint32_t i, j;
4        FOR_EACH_CELL
5              div[IX(i,j)] = -0.5f*(solver->u[IX(i+1,j)]-solver->u[IX(i-1,j)]
6                    +solver->v[IX(i,j+1)]-solver->v[IX(i,j-1)])/solver->N;
7              p[IX(i,j)] = 0;
8        END_FOR
9        solver_set_bnd ( solver, 0, div ); solver_set_bnd ( solver, 0, p );
10       solver_lin_solve ( solver, 0, p, div, 1, 4 );
11       FOR_EACH_CELL
12             solver->u[IX(i,j)] -= 0.5f*solver->N*(p[IX(i+1,j)]-p[IX(i-1,j)]);
13             solver->v[IX(i,j)] -= 0.5f*solver->N*(p[IX(i,j+1)]-p[IX(i,j-1)]);
14       END_FOR
15       solver_set_bnd ( solver, 1, solver->u ); solver_set_bnd ( solver, 2, solver->v );
16 }
```

Figura 6: Código de solver_project

```
Uso de la aplicacion:
demo m [ID PORT] donde m es el modo de ejecucion y
ID PORT (opcional) el Id:puerto de cliente
0 para correr una Demo automatica
1 para correr una Demo interactiva
(click izquierdo modifica el campo vectorial de fuerzas,
 click derecho agrega densidad, v muestra el campo de fuerzas
 y q finaliza la ejeucion)
2 para correr los tests
3 ID PORT para correr en modo cliente-servidor
 ID es el numero de maquina
 PORT es el puerto por el que recibe los mensajes
```

Esto nos dice que podemos correr el programa en cuatro modos, el primero es una demo automática que genera valores predefinidos de velocidad y densidad. El segundo modo permite interactuar con la aplicación, modificando las velociades a través del botón izquierdo del mouse y agregando densidad a través del botón derecho. La tercera opción permite correr una prueba contra el resultado de la cátedra. La cuarta opción está reservada para la fecha de entrega.

La opción de prueba va a inicializar matrices de diversos tamaños, va a ejecutar la simulación por una cantidad fija de pasos y va a guardar el resultado de la misma en formato BMP (uno por cada matriz: velocidad en el eje x, velocidad en el eje y, densidad). También se guardará un BMP (diff) que calcula la diferencia entre su simulación y la implementación de la cátedra, si la implementación es conforme a la de la cátedra el resultado debería ser una imagen negra.

3.2.3. Mediciones de rendimiento

La forma de medir el rendimiento de nuestras implementaciones se realizará por medio de la toma de tiempos de ejecución. Como los tiempos de ejecución son muy pequeños, se utilizará uno de los contadores de performance que posee el procesador.

La instrucción de assembler `rdtsc` permite obtener el valor del Time Stamp Counter (TSC) del procesador. Este registro se incrementa en uno con cada ciclo del procesador. Obteniendo la diferencia entre los contadores antes y despues de la llamada a la función, podemos obtener la

cantidad de ciclos de esa ejecución. Esta cantidad de ciclos no es siempre igual entre invocaciones de la función, ya que este registro es global del procesador y se ve afectado por una serie de factores.

Existen principalmente dos problemáticas a solucionar:

1. La ejecución puede ser interrumpida por el *scheduler* para realizar un cambio de contexto, esto implicará contar muchos más ciclos (*outliers*) que si nuestra función se ejecutara sin interrupciones.

2. Los procesadores modernos varian su frecuencia de reloj, por lo que la forma de medir ciclos cambiará dependiendo del estado del procesador.

Para medir tiempos deberán idear e implementar una metodología que les permita evitar estos dos problemas, o al menos reducir su impacto. En el archivo `rdtsc.h` encontrarán las funciones necesarias para implementarla.

3.3. Informe

El informe debe incluir las siguientes secciones:

1. Carátula: La carátula del informe con el **número/nombre del grupo**, los **nombres y apellidos** de cada uno de los integrantes junto con **número de libreta** y **email**.

2. Introducción: Describe lo realizado en el trabajo práctico. (y si quedó algo sin realizar)

3. Desarrollo: Describe **en profundidad** cada una de las funciones que implementaron. Para la decripción de cada función deberan decir cómo opera una iteración del ciclo de la función. Es decir, cómo mueven los datos de la matriz a los registros, cómo los reordenan para procesarlos, las operaciones que se aplican a los datos, etc. Para esto pueden utilizar pseudocódigo, diagramas (mostrando gráficamente el contenido de los registros **XMM**) o cualquier otro recurso que les resulte útil para describir la adaptación del algoritmo al procesamiento vectorial. No se deberá incluir el código assembler de las funciones (aunque se pueden incluir extractos en donde haga falta).

4. Resultados: **Deberán analizar** y **comparar** las implementaciones de las funciones en su versión de la cátedra en **C** y la suya en **assembler** y mostrar los resultados obtenidos a través de tablas y gráficos. Para esto deberán plantear experimentos que les permitan medir el rendimiento y comparar entre las implementaciones. Deberán además explicar detalladamente los resultados obtenidos y analizarlos. En el caso de encontrar anomalías o comportamientos no esperados deberán construir nuevos experimentos para entender qué es lo que sucede.

5. Conclusión: Reflexión final sobre los alcances del trabajo práctico, la programación vectorial a bajo nivel, problemáticas encontradas, y todo lo que consideren pertinente.

El informe no puede exceder las **20** páginas, sin contar la carátula.

Importante: El informe se evalúa de manera independiente del código. Puede reprobarse el informe y en tal caso deberá ser reentregado para aprobar el trabajo práctico.

4. Entrega

Se deberá entregar el solamente contenido de la carpeta `src` junto con el informe.

La fecha de entrega de este trabajo es **5/10**. Deberá ser entregado a través de un repositorio GIT almacenado en `https://git.exactas.uba.ar` respetando el protocolo enviado por mail y publicado en la página web de la materia.
Ante cualquier problema con la entrega, comunicarse por mail a la lista de docentes `orga2-doc@dc.uba.ar`.

5. Recomendaciones finales

Recuerden planificar el trabajo, tanto implementación, testing, experimentos e informe. Diseñen en términos esquemáticos la forma de procesar los datos, qué registro usarán y de que modo, cómo recorrerán las matrices, etc, lo mismo que se les pide entregar en el informe pero recomendamos hacerlo efectivamente **antes** de hacer el informe y no luego de haberse metido en un atolladero. Utilicen la planificación y los esquemas para **gestionar responsabilidades y discutir entre quienes conformen el grupo**, eviten volcarse rapdidamente a discusiones acaloradas sobre el orden de las instrucciones o la cantidad de espacios en una tabulación.
La curiosidad es buena pero no hace falta comprender a fondo los mecanismos subyacentes al algoritmo, deben poder implementar una versión eficiente del código sin necesidad de manejar los detalles finos de la teoría. Para quien quiera consultar el material sobre el cual se basa este trabajo pueden acceder a:

`http://www.intpowertechcorp.com/GDC03.pdf`