Organizacion del Computador II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico N2

Grupo Lofrani

Integrante	LU	Correo electrónico
Ruiz Ramirez, Emanuel	343/15	ema.capo2009@hotmail.com
Serio, Franco	215/15	francoagustinserio@gmail.com
Soberon, Nicolás	641/10	nico.soberon@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción

Este Trabajo Práctico se basa en utilizar el modelo de procesamiento SIMD (Single Instruction Multiple Data) por medio del uso de instrucciones SSE (Streaming SIMD Extensions), para poder desarrollar distintos metodos para 'Navier Stokes' y asi poder evaluar el rendimiento de las mismas. Algunas de las ventajas de usar las instrucciones SSE: Ejecutar de manera paralela (simultaneamente) la misma instrucción sobre distintos datos. Utilizar los registros XMM, los cuales nos sirven para operar con datos empaquetados y de punto flotante. Reducir los accesos a memoria, ya que podemos guardar mas datos en registros y con una sola instrucción mover 128 bits a memoria.

Se implementaron los siguientes metodos en Assembler:

- **solver_lin_solve** : Se encarga de calcular la difusion del fluido a modelar.
- **solver_set_bnd**: Calcula los valores para los casos borde de las matrices.
- **solver_project**: Proyecta los nuevos valores en la matriz de velocidad

2. Desarrollo

Vamos a trabajar con matrices. Estas matrices de velocidad son la representación discreta del campo vectorial asociado simplificado al tomar un vector representante de la grilla obtenida al dividir el espacio en celdas de tamaõ fijo. En estas matrices agregamos una columna y una fila a cada lado de nuestra grilla para simplificar el trabajo sobre los bordes. Al recorrer la matriz, vamos a poder acceder a 4 celdas a la vez. De esta manera podemos aprovechar el modelo SIMD, para poder operar con estos datos juntos, y asi realizar menos iteraciones.

2.1. solver_lin_solve

2.2. solver_set_bnd

2.3. solver_project

En cuanto al desarrollo de solver_project, tuvimos q aplicar una formula a cada celda de la matriz, luego llamar solver_set_bnd y solver_lin_solve, y luego volver a procesar las celdas de la matriz. El metodo funciona de la siguiente manera:

- Primero, se van a hacer un total de (ancho * alto / 4) de iteraciones, dividimos a N por 4 porque vamos a agarrar de a 4 posiciones de la matriz.
- Vamos a diferenciar 3 casos, uno cuando estamos a principio de una fila, otro en el medio de la fila y otro al final de una fila.
- Estos casos nos van a servir a la hora de ver las posiciones $j+1$ y $j-1$.
- Comenzamos por traer 4 celdas de source al comienzo del ciclo.
- Si estamos al comienzo de la fila con el caso de inicio de fila, nos vamos a traer las 4 posiciones que estoy mirando, y las 4 siguientes.
- La primera posicion no nos va a interesar, porque luego la vamos a pisar, cuando llamemos al metodo que setea los bordes de la matriz.
- Utilizando SHIFTS vamos a acomodar los dos registros XMM de manera tal que podamos realizar la resta correspondiente, y obtener el valor de $j+1$ - $j-1$ para las cuatro posiciones que estamos mirando.
- Luego si estamos a mediados de la fila, vamos a tener las 4 celdas que estoy mirando, las 4 siguientes y las 4 anteriores.
- (Para poder aprovechar que las anteriores ya las vamos a tener de la iteracion anterior, las vamos almacenando en un XMM, donde guardamos las 4 celdas anteriores, asi no las tenemos que pedir a memoria nuevamente)
- Con estos 3 sets de 4 celdas, vamos a realizar SHIFTS nuevamente, para poder realizar la operacion entre las posiciones $j+1$ y $j-1$, para las cuatro celdas a la vez.
- Si entramos al caso que estamos al final de la fila, es analogo al caso de principio de fila, con la diferencia que aca vamos a traernos las 4 celdas que estamos mirando y las 4 anteriores. Operamos de la misma forma que al comienzo de la fila.
- Luego en cualquiera de los 3 casos, a la hora de operar con las posiciones $i+1$ y $i-1$, vamos a traernos en cada iteracion las 4 celdas actuales, las 4 celdas que se encuentran en la fila anterior, arriba de las actuales y las 4 celdas de la fila siguiente, que se encuentran debajo de las actuales.
- teniendo *anteriores* y *siguientes*, podemos realizar una resta entre esos dos registros, para poder computar para las 4 celdas actuales, la operacion p[IX(i+1,j)]-p[IX(i-1,j) que queriamos realizar.
- Por ultimo, dependiendo del caso, vamos a utilizar una mascara que contiene valores 0,5 y el valor N para poder completar las operaciones con los valores que ya obtuvimos para las 4 celdas que estamos mirando.

- Una vez que terminamos con el primer ciclo, vamos a llamar a *solver_set_bnd* y a *solver_lin_solve*.
- Luego vamos a construir el segundo ciclo, de la misma manera que construimos el primero, con la unica diferencia que vamos a realizar las operaciones correspondientes con las celdas que estamos mirando.
- Por ultimo llamamos al metodo *solver_set_bnd*.

De esta manera podemos aplicar el modelo SIMD a nuestro metodo. Aprovechando poder trabajar con multiples datos a la vez, nos vamos a ahorrar operaciones, lo que implica un menor tiempo de computo, y una performance mas alta de nuestro algoritmo.

3. Resultados

Nuestra Hipotesis a la hora de experimentar, es que, utilizar el modelo SIMD con operaciones SSE, nos da una mayor performance a la hora de desarrollar nuestros algoritmos. El hecho de poder procesar varios datos a la misma vez, nos permite ahorrarnos tiempo de ejecucion y cantidad de iteraciones en nuestros algoritmos. Realizamos distintas corridas de nuestro codigo, tanto en ASM, como en C con las correspondientes optimizaciones que nos permite el compilador. De esta manera pudimos respaldar nuestra hipotesis con resultados concretos.

3.1. solver_lin_solve

3.2. solver_set_bnd

3.3. solver_project

4. Conclusión

4.1. C y sus optimizaciones vs ASM