Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI0118 – LENGUAJE ENSAMBLADOR
Profesor Luis Quesada

Tarea Programada 2

Estudiante:
Emmanuel D. Solís [redacted]

6 de julio de 2021

Índice

CI-0118 Lenguaje Ensamblador, g01

Tarea programada #2

I/2021

"There should be no such thing as boring mathematics"
— Edsger W. Dijkstra

Motivación

Múltiples cálculos matemáticos son usados a diario en aplicaciones computacionales. Estas aplicaciones incluyen desde despliegue de gráficos (en juegos, por ejemplo), hasta métodos de clasificación y reconocimiento que le dan soporte a los algoritmos de *machine learning*. Muchas veces las operaciones matemáticas involucradas se realizan múltiples veces. En aplicaciones que utilicen grandes cantidades de datos, podrían hacerse millones de cálculos por segundo.

Por otro lado, hemos hablado en clase de lo importante que puede ser el rendimiento. Algunas aplicaciones requieren dar respuestas en un par de jiffies[1].

En esta tarea se desea implementar un cálculo matemático ampliamente utilizado de distintas formas, esto con el objetivo de comparar las distintas implementaciones y determinar cuál de ellas parece tener un mejor rendimiento.

Note que con su resultado no se puede concluir que una implementación es mejor que otra en absolutamente todos los contextos. Existen factores de ruido (elementos que pueden causar variabilidad en la salida del sistema) que podrían afectar: sistema operativo, arquitectura de la máquina, memoria disponible, otros programas que se estén ejecutando al mismo tiempo, entre otros.

Paso 1

Supongamos el cálculo del producto cruz entre dos vectores. Este cálculo es muy usado en aplicaciones gráficas. Si no recuerda cómo hacer el cálculo del producto cruz, aquí *#julioprofe* se lo explica: https://www.youtube.com/watch?v=fmAhi1N-uL8.

En alto nivel (C/C++), genere una cantidad de datos "grande" que le sirva como insumo para realizar mucho producto cruz entre dos vectores. El que tan grande es esa cantidad de datos va a depender de su máquina. Diez mil vectores, cien mil vectores, un millón de vectores. No importa la cantidad pero sí que intente llevarla al máximo de la capacidad de su computadora. Recuerde que C/C++ también tiene un límite que debe considerar. Puede generar los datos aleatoriamente una única vez, guardarlos en un archivo.

[1] Más sobre jiffies en Linux aquí
http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch10lev1sec3.html

Paso 2

Sabiendo ya cómo hacer el cálculo del producto cruz y teniendo datos suficientes, implemente una función por cada uno de los siguientes casos, que reciba dos vectores y devuelva de alguna forma el resultado del producto cruz. Los casos son los siguientes:

1. Cálculo de producto cruz en C++ (como lo haría de forma tradicional).
2. Cálculo de producto cruz en C++/ASM usando instrucciones AVX escalares.
3. Cálculo de producto cruz en C++/ASM usando instrucciones AVX vectorizadas.
4. Cálculo de producto cruz en C++/ASM usando instrucciones SSE escalares.
5. Cálculo de producto cruz en C++/ASM usando instrucciones SSE vectorizadas.

Paso 3

Usando la biblioteca chrono[2], cronometre el tiempo obtenido en la misma máquina ejecutando los casos anteriores. Para que el tiempo le de valores significativos puede calcular en nanosegundos, además, puede hacer cálculos de cientos o miles de productos cruz.

Además, debe evitar incluir dentro de tiempo a cronometrar situaciones que pueden "inflar" los números, como por ejemplo imprimir en consola o leer datos de la misma.

Debe ejecutar varias veces cada caso y anotar los tiempos obtenidos. Recuerde usar siempre la misma unidad de tiempo, poner los cronómetros en el mismo lugar, etc. ¿Cuántas corridas son suficientes? No menos de 20. Aunque 20 puede ser un número arbitrario, y probablemente no es suficiente para tener resultados estadísticos definitivos, lo definiremos como base dadas las limitaciones de tiempo.

Paso 4

Tome los resultados obtenidos producto de ejecutar los 5 casos, al menos 20 veces cada caso, tabúlelos en una hoja de cálculo y analícelos. Puede recurrir a sus conocimientos de cursos anteriores y calcular la moda, la varianza, el promedio, o cualquier dato que le permita dar una conclusión sobre cuál caso pareciera haberse comportado mejor y cuál peor, dadas las restricciones y el entorno en el que se ejecutó la prueba.

Prepare un documento donde incluya:

- Descripción del sistema donde ejecutó sus pruebas.
- Descripción de la ejecución (datos de prueba -cantidad, tipo de datos usado, etc.-, cómo se midió el tiempo, y cualquier otro dato que considere relevante).
- Una tabla resumen con los datos obtenidos y al menos un gráfico.
- Un párrafo donde interprete la tabla resumen y el gráfico presentados.

Suba antes de la 11.59pm del viernes 2 de julio un pdf con su nombre que incluya todo lo solicitado en este paso. Suba además, como un archivo aparte, un comprimido con todo el código producto de esta tarea.

[2] https://www.geeksforgeeks.org/chrono-in-c/

1. Descripción del sistema

La máquina principal donde las pruebas fueron ejecutadas es en una computadora de marca *Apple* con las siguientes características:

- **Procesador:** 2.2GHz 6-core Intel Core i7, Turbo Boost up to 4.1GHz, with 9MB shared L3 cache.
- **Memoria RAM:** 16GB of 2400MHz DDR4 onboard memory.
- **Disco duro:** 256GB SSD.
- **Tarjeta Gráfica:**
  - Radeon Pro 555X 4 GB.
  - Intel UHD Graphics 630 1536 MB
- **Sistema Operativo:** MacOS Big Sur 11.4

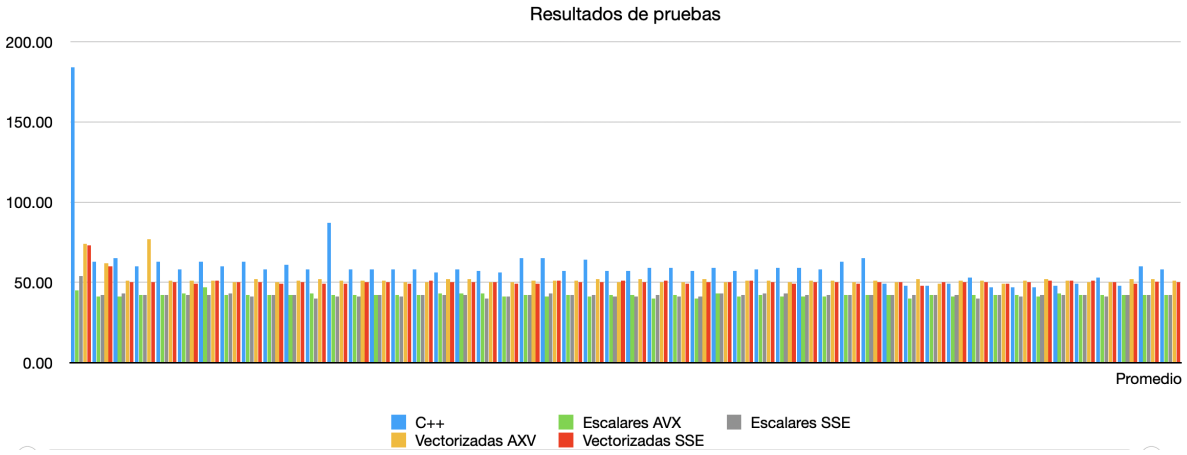
A la vez se provee un enlace a la página oficial del fabricante de donde esta información ha sido tomada: `https://support.apple.com/kb/SP776?locale=en_US`.

2. Descripción de la ejecucuión

Los datos de prueba usados fueron para los números $m = \{-3, -2, 5\}$, y $n = \{6, -10, -1\}$. Estos fueron siempre usados en todas las funciones como *flotantes*, es decir con 32 bits. El tiempo fue medido usando nanosegundas, los resultados de estas pruebas se encuentran en una tabla que ha sido adjunta, pues los resultados de las pruebas eran mucho se veía desordenados ser incluidos en este documentos.

3. Datos obtenidos

Se presenta a continuación los resultados en la siguiente tabla, mostrando que la versión en C++ tiende ser la de mayor duración para casi todas las pruebas.



4. Conclusiones

A partir de los datos presentados se puede concluir que un cóodigo creado en C++ tiende a tener una mayor duración, pues por debajo deben ser efectuadas muchas conversiones y demás operaciones. Ademas de que las instrucciones escalares en AVX tienden a ser las de menor duración, aún tomando en cuenta que estos registros están mas lejos del procesador.