Reporte de resultados Proyecto 2

Arturo Chinchilla Sánchez, Jeremy Rodríguez Solórzano, Santiago Gamboa Ramírez
mchinchilla11@gmail.com jrodriguezs0292@gmail.com santigr17@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—El propósito de este proyecto es la implementación en lenguaje C++ de un método para la optimización en la ejecución de funciones y estructuras algorítmicas básicas, como lo es el uso de instrucciones SIMD. Además, se propone como ejemplo para esta implementación la ejecución de la descomposición LU usada previamente.**

***Palabras clave*—SIMD, optimización, intrinsics.**

I. Introducción

Cuando se desea implementar la solución a un problema de software, uno de los aspectos de importancia a tomar en cuenta es la optimización del mismo, principalmente, si los parámetros a analizar abarcan gran cantidad de memoria y llevan consigo tiempos de ejecución a niveles de minutos o hasta horas. Es por eso que, la importancia de conocer conceptos que ayudarán a resolver esta clase de aspectos, va más allá de un simple conocimiento extra.

En este sentido, las funciones llamadas intrinsics entran al juego para asistir al desarrollador con este tema. Intrinsics se refiere, según Miguel-Alonso et al, a funciones en C++ que contienen una traducción directa (o casi directa) a instrucción máquina [1]. Esta funciones dan acceso a las instrucciones SIMD (Single Instruction, Multiple Data), las cuales se basan en usar operaciones en paralelo con registros adicionales del procesador para almacenar datos que luego van a ser procesados en una sola instrucción en un ciclo de reloj [2].

Lo anterior abarca una gran área de conceptos que son necesarios conocer, pero más importante aún, engloba un gran ámbito de aplicaciones en la actualidad -como por ejemplo, el procesamiento de multimedia o, como en este caso, la manipulación de estructuras de datos (como matrices) de gran tamaño- lo que hace que la investigación profunda en este tema valga la pena.

Para el caso de este proyecto, se profundiza en características de importancia hacia este tema, como el manejo de memoria y una aplicación común en ingeniería como lo es la solución a sistemas de ecuaciones lineales.

Se pretende profundizar en la materia para alentar la experiencia en el mismo, y así, esbozar soluciones a problemas que requieren un alto desempeño de programación, con mayor facilidad.

II. Estado del Arte

Las operaciones SIMD (single instruction multiple data) por sus siglas en inglés, son utilizadas para lograr una mejora en el tiempo requerido para procesar datos que utilizan la misma función.
Es necesario que en la arquitectura del procesador sea capaz de soportar este tipo de instrucciones. Es por eso que empresas fabricantes de procesadores, ponen a disposición de los usuarios sets de instrucciones. Entre estos sets de instrucciones existe AVX. Según [4] Intel AVX por sus siglas en inglés, o extensiones vectoriales avanzadas, son un conjunto de instrucciones de 256 bits desarrollado por Intel Corporation extensión al conjunto de instrucciones x86 utilizado en procesadores de Intel y AMD.

III. Propuesta

Las instrucciones SIMD se ejecutan de manera diferente en relación con la arquitectura y procesador de la máquina, es decir, no poseen portabilidad. Por ello, las funciones intrinsics usadas en este proyecto se eligen de acuerdo con el set de instrucciones que soporte los computadores de todos los integrantes de grupo. Para ello, se usó la guía *intrinsics guide* de [3], que selecciona el set de entre distintos tipos, para este caso se usó AVX.

Se desea implementar una solución optimizada para dar con el resultado de ecuaciones lineales, para poder encontrar los valores de corrientes en una matriz *nxm* de resistencias, donde cada posición de la matriz es un nodo, como se muestra en la figura 1.

Con esta matriz se modela la trayectoria que puede seguir un robot, dicha trayectoria se obtiene a partir de una imagen que representa el mapa con los obstáculos. De este modo cada nodo de la matriz representa un pixel del mapa, y el valor de la resistencia entre un nodo y otro representa si el camino es libre (valor bajo de 1 ohm) o con obstáculos (valor alto de 1 Mohm).

En base al código utilizado para la tarea 4 (descomposición LU), se agregó archivos necesarios:

- MapToLinear.hpp: Una función que permita el mapeo de índices matriciales a lineales, con la cual se puede, a partir de los índices de dos nodos y la cantidad de columnas de la matriz, obtener el índice de la corriente (resistencia) ubicada entre esos nodos.
- MapToNode.hpp: optiene, a partir del número de columna de la matriz usada y el índice mapeado de manera lineal de la resistencia, el índice de los nodos de los extremos de dicha resistencia. Estos índices se almacenan y retornan en un vector de 4 posiciones, donde, de forma respectiva desde la posición 0 hasta la 3, se guarda fila del nodo 1, columna del nodo 1, fila del nodo 2 y columna del nodo 2.

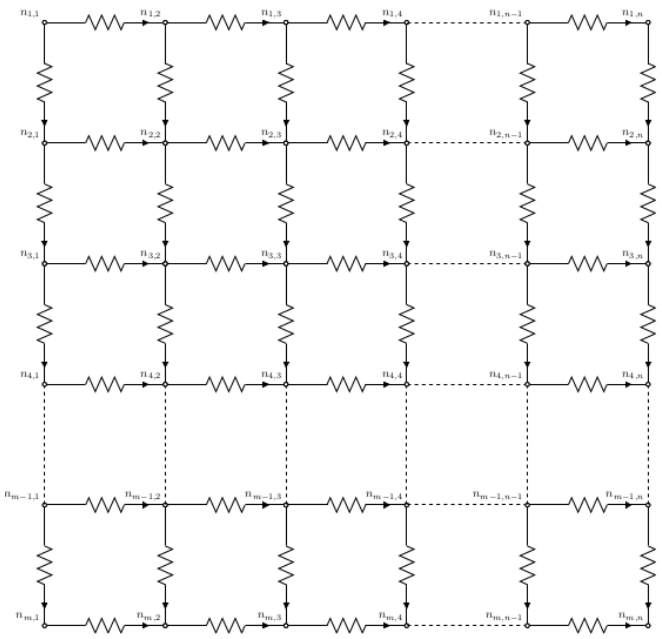

Figura 1. Malla resistiva para modelar búsqueda de trayectoria.

- imgManager.hpp: Se encarga de llenar una matriz con 1's y 0's a partir de una imagen cargada con la biblioteca *OpenCV*. En esta matriz, los 1's representan un espacio en negro de la imagen, es decir, un obstáculo, y los 0's representan un espacio en blanco, es decir, un camino libre. Esto se muestra en la figura 2
- LUDoolitleSIMD.hpp: optiene el resultado de la optimización LU por medio del método de Doolitle optimizado con instrucciones SIMD.

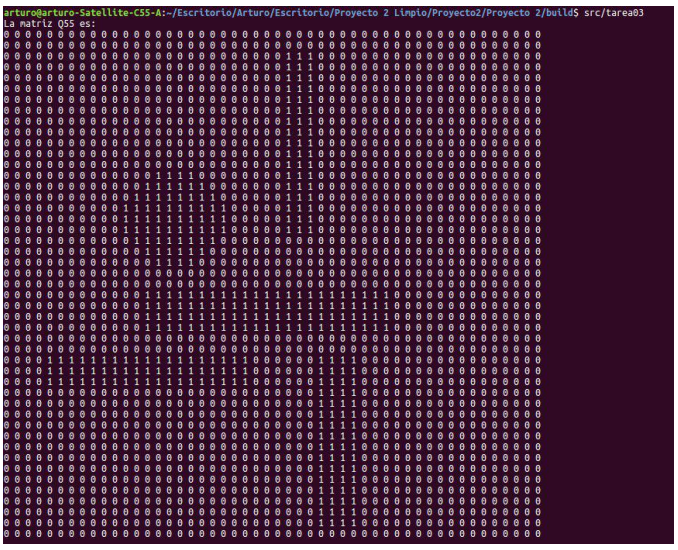

Figura 2. Ejemplo de matriz 50x50 para una imagen cargada mediante la biblioteca *OpenCV*.

Además, se agregó la función *fill*, que ejecuta el llenado de una matriz con un valor escalar de parámetro. Dicha función ya estaba implementada de forma genérica dentro de la clase *Matrix*, pero se implementó de forma obtimizada mediante instrucciones SIMD.

IV. Resultados

Por medio de la clase Plot2d del archivo PlotPy.hpp del código original, se obtuvo el resultado del benchmark para la función *fill* que realiza el llenado de una matriz con un número escalar, tanto de manera genérica como obtimizada. Este resultado se presenta en la figura 3

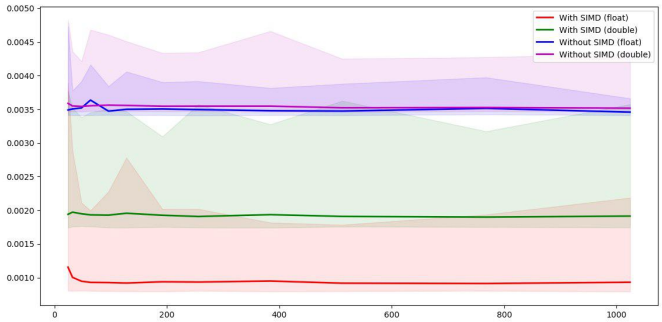

Figura 3. Resultado gráfico del Benchmark aplicado a la función fill.

Para el caso de la descomposición por el método LU, en este caso ejecutado con el algoritmo Doolitle (ya que según resultados y análisis de la tarea anterior, fue el método más eficiente en comparación con el algoritmo de Crout), se obtuvo la prueba Benchmark ilustrada en el gráfico de la figura 4. para este caso se puede notar una clara diferencia en cuanto al uso del algoritmo con funciomes genéricas y su uso con funciones SIMD.

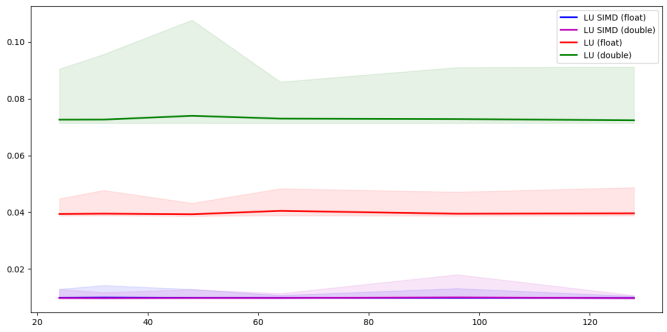

Figura 4. Resultado gráfico del Benchmark aplicado a la función LU.

V. Conclusiones

Dado el resultado obtenido (figura 3 y 4), se puede decir que la optimización de funciones por medio de la biblioteca *intrinsecs*, resulta en un método confiable, para la manipulación de estructuras de datos de gran tamaño, como en este caso lo es una matriz.

Con lo anterior, se puede entender que el uso de SIMD para operaciones básicas en matrices, ayuda a disminuir el tiempo de ejecución, esto se nota sin importar la medida del tamaño de datos.

Al evaluar el desempeño, se puede concluir que el uso de SIMD resulta imprescindible en la manipulación de estructuras de datos que efectúan operaciones a grandes cantidades de valores.

El uso de *intrinsics* y SIMD, lleva consigo conocimientos avanzados de programación, y el entendimiento de instrucciones usadas a nivel de procesador, además del manejo de memoria por parte de las mismas, por ello, se debe ahondar más en el tema, para procurar la culminación de todas las partes que este proyecto evalúa.

REFERENCIAS

[1] Miguel, J, Pérez, C, "Programación SIMD para x86, AltiVec y Cell", *sc.ehu.es*, s. f. [online]. Available: http://campusv.uaem.mx/cicos/imagenes/memorias/6tocicos2008/Articulos/ponencia %206a.pdf. [Accessed April 27, 2018].

[2] Rodríguez, A, "Programación de Alto Desempeño", *campusv.uaem.mx*, 2008. [online]. Available: http://www.sc.ehu.es/ccwbayes/members/cperezmig/pubs/programacion_simd.pdf. [Accessed April 27, 2018].

[3] INTEL, "Intrinsics Guide", *software.intel.com*, s. f. [online]. Available: https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX. [Accessed April 12, 2018].

[4] Software.intel.com. (2018). ISA Extensions — Intel® Software. [online] Available at: https://software.intel.com/en-us/isa-extensions [Accessed 28 Apr. 2018].