Optimización SIMD

Juan Pablo Brenes Coto, Pablo Bustamante Mora, Emily Sancho Murillo
juanp1995@gmail.com pablex.bumo@gmail.com emilysancho@gmail.com
Área académica de Ingeniería en Computadores
Tecnológico de Costa Rica

***Resumen*—En el presente documento se detallará la implementación de funciones de tipo *intrinsics* para el manejo de instrucciones SIMD (*Single Instruction Multiple Data*, por sus siglas en inglés) en una arquitectura AVX, las cuales permiten la optimización de un programa mediante el paralelismo.**

***Palabras clave*—Intrinsics, SIMD, paralelismo, AVX.**

I. Introducción

I-A. Contexto

A la hora de tener una gran cantidad de datos como entrada de un algoritmo es de esperar que el tiempo necesario para ejecutar el mismo aumente. Ahora, esto no representa una razón válida para atrasar un procedimiento entero, por lo que se tuvo que buscar una manera de disminuir el tiempo de ejecución de los procedimientos. Para esto, la paralelización resulta una herramienta muy útil, dado que ya no es necesario trabajar de manera serial los datos, sino que al mismo tiempo en el que se ejecuta normalmente una operación se pueden ejecutar 2, 3 o más.

I-B. Problema

¿Cómo se puede optimizar un procedimiento de manera que requiera menos tiempos de ejecución?

I-C. Objetivos y esbozo de la solución

Para implementar la paralelización se utilizó *Intrinsics* [2], un conjunto de funciones que permiten ejecutar instrucciones SIMD [4] - (*Single Instruction Multiple Data*). Las instrucciones SIMD responden a un tipo de arquitectura de procesador, por lo que no es portable. Para esto, existe una gran variedad de instrucciones, cada una para un tipo de procesador específico. En el presente proyecto se utilizó el set de instrucciones para un procesador con arquitectura AVX [3].
Se buscó realizar lo siguiente:

- Optimizar las operaciones de suma y resta de matrices.
- Optimizar el algoritmo de descomposición LU, específicamente el método Doolittle, para descomponer una matriz. Así como el algoritmo que utiliza dicha descomposición para resolver un sistema de ecuaciones del tipo $A\underline{x} = \underline{b}$. Siendo *A* la matriz de coeficientes, *x* el vector a calcular y *b* el vector con los resultados de cada ecuación.
- Aplicar las optimizaciones anteriores para la resolución de un problema práctico y poder comparar visualmente la diferencia de rendimiento con y sin optimización SIMD.

II. Propuesta

Para poder realizar la comparación del código sin optimización y con optimización se utilizan directivas de preprocesador [1] como banderas, que indican si se desea utilizar optimización. Estas se escriben en los archivos *headers* de cada clase y son modificadas en tiempo de compilación.La herramienta CMake, además de automatizar la compilación del código, permite definir dichas directivas por medio de sus archivos de configuración.

Para poder determinar si la optimización se realizó o no de manera correcta, se utilizan las pruebas de *benchmark*, las cuales miden el tiempo requerido por el algoritmo al ejecutar la misma cantidad de datos. Los resultados se grafican para tener una referencia más sencilla de la optimización.

La optimización SIMD de operaciones como la resta, resulta trivial en compiladores modernos; ya que estos optimizan automáticamente dichas operaciones básicas durante tiempo de compilación. Esto se podrá ver con mayor claridad en la Sección III, donde se muestran los tiempos requeridos para completar la resta de matrices de diferentes tamaños, utilizando el algoritmo sin optimizar y su versión SIMD.

III. Resultados

	Tiempo (s)	
Tamaño n	Sin optimizar	SIMD
128	$7{,}972 \times 10^{-6}$	$2{,}254 \times 10^{-5}$
512	$2{,}861 \times 10^{-4}$	$9{,}262 \times 10^{-4}$
1024	$1{,}849 \times 10^{-3}$	$3{,}813 \times 10^{-3}$
2048	$6{,}979 \times 10^{-3}$	$1{,}326 \times 10^{-2}$
4096	$2{,}604 \times 10^{-2}$	$5{,}250 \times 10^{-2}$

Cuadro I
Comparación de tiempos requeridos para completar la resta de dos matrices de tamaño $n \times n$ con precisión doble.

	Tiempo (s)	
Tamaño n	Sin optimizar	SIMD
128	$6{,}451 \times 10^{-6}$	$8{,}206 \times 10^{-6}$
512	$2{,}047 \times 10^{-4}$	$2{,}086 \times 10^{-4}$
1024	$1{,}630 \times 10^{-3}$	$1{,}686 \times 10^{-3}$
2048	$6{,}438 \times 10^{-3}$	$7{,}200 \times 10^{-3}$
4096	$2{,}662 \times 10^{-2}$	$2{,}627 \times 10^{-2}$

Cuadro II
Comparación de tiempos requeridos para completar la resta de dos matrices de tamaño $n \times n$ con precisión simple.

IV. Conclusiones

- Para realizar la implementación de las funciones de intrinsics es necesario tener un control muy riguroso sobre el tipo de dato manejado. Son funciones fuertemente tipadas.
- La aplicación de la optimización al código LUDoolittle y el LUSolver puede llevar a comportamientos anómalos, si no se maneja de la manera adecuada.
- Intrinsics tiene una amplia gama de funciones que permiten optimizar el código. Es necesario estudiarlas antes de codificar, puesto que de lo contrario se podría estar realizando un procedimiento de manera manual que ralentiza el código en comparación con la función de Intrinsics respectiva.
- En compiladores modernos la optimización por medio de instrucciones SIMD, de operaciones básicas como la suma y resta, es algo trivial; ya que el compilador optimiza automáticamente dichas operaciones durante tiempo de compilación.

Referencias

[1] **Cpluplus. (2017). Preprocessor directives: Obtenido de http://www.cplusplus.com/doc/tutorial/preprocessor/**

[2] **Intel. (s.f). Intrinsics Guide: Obtenido de https://software.intel.com/sites/landingpage/IntrinsicsGuide/techs=AVX**

[3] **Lomont, C. (2011, Junio 21). Introduction to Intel Advanced Vector Extensions. Obtenido de https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions**

[4] **Inter. (2018, Setiembre 10). Tecnología de extensiones SIMD de transmisión de Intel. Obtenido de intel.la/content/www/xl/es/support/articles/000005779/processors.html**