Aproximación de funciones mediante series de Taylor

Evaluación eficiente de polinomios

Francisco Alvarado F. Andrés Arias V. Pablo Rodríguez Q.
Área académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica

***Abstract*—En esta investigación se analizan las series de Taylor para dos diferentes funciones y diferentes técnicas para optimizar dicha aproximación y obtener resultados con menos error en el menor tiempo de ejecución.**

Palabras clave*—*Series de Taylor; Análisis numérico; Optimización; Paralelismo; SIMD; Metaprogramación.

I. Introducción

El teorema de Taylor es fundamental en el análisis numérico y en la ingeniería. Mediante las series de Taylor es posible aproximar funciones mediante polinomios fácilmente programables. Las series de Taylor implican la evaluación constante de polinomios, operación que es posible de optimizar mediante reducción de operaciones o paralelismo.

Se explorarán maneras de optimizar el análisis numérico mediante el teorema de Taylor con diferentes ténicas que incluyen reducción de operaciones, paralelismo mediante SIMD (*Single Instrucion, Multiple Data*) y metaprogramación. Se analizarán los tiempos de ejecución y los errores generados en la aproximación.

II. Series de Potencias

Las series de potencias son una sumatoria polinomial que presentan la siguiente forma:

$$\sum_{n=0}^{\infty} a_n(x-c)^n = a_0 + a_1(x-c)^1 + a_2(x-c)^2 + \dots \quad (1)$$

III. Teorema de Taylor

El teorema de Taylor describe la posibilidad de aproximar funciones alrededor de un punto en el cual la función es derivable. Dicha aproximación se realiza mediante un tipo especial de series de potencias conocidas como **series de Taylor**, las cuales presentan la siguiente forma:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \dots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n \quad (2)$$

La ecuación (2) puede ser reescrita como una sumatoria de la siguiente manera:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n \quad (3)$$

La ecuación 3 describe lo que se conoce como la serie infinita de Taylor. Usualmente solo se toma un número arbitrario de coeficientes, entre más se tomen, más fiel es la aproximación al valor real. Los valores no tomados en cuenta se verán reflejados como error de truncamiento.

IV. Evaluación de polinomios

Las series de Taylor generan polinomios de grado N, donde "N" es la cantidad de coeficientes de la serie calculados. Las constantes de los polinomios vienen siendo dados por la evaluación del centro en las derivadas de la función.

La idea principal del teorema de Taylor es obtener el valor aproximado de la evaluación de valores en la función original, lo que implica la evaluación de dicho valor en un polinomio.

La evaluación tradicional de polinomios es sumamente ineficiente, ya que para un polinomio de grado "n" se requieren "n" sumas y n(n+1)/2 multiplicaciones, operación que tiende a ser lenta.

Existen maneras más eficientes de agrupar y evaluar polinomios de manera eficiente y rápida, de las cuales se analizaron dos en la presente investigación.

A. *Esquema de Horner*

El esquema de Horner es considerado como la forma más rápida y eficiente para evaluar un polinomio, sin embargo presenta la traba de ser meramente secuencial, por lo que su implementación no aprovecharía el paralelismo presente en procesadores modernos. La forma general del esquema de Horner se describe en la ecuación 4.

$$f_n(x) = a_0 + x(a_1 + x(\dots(a_{n-1} + a_n x)\dots)) \quad (4)$$

El esquema de Horner permite evaluar el polinomio utilizando únicamente n multiplicaciones y n sumas, lo cual disminuye el impacto de errores de redondeo.

B. *Esquema de Estrin*

El esquema de Estrin organiza el polinomio como un árbol de multiplicaciones y divisiones, en el cual cada nivel es capaz de ejecutarse en paralelo. El esquema de Estrin divide el polinomio en expresiones del tipo Ax+B [7].

A diferencia del esquema de Horner, el esquema de Estrin no disminuye la cantidad de operaciones requeridas para evaluar el polinomio, sin embargo logra compensar esa

desventaja de rendimiento con alta capacidad de paralelismo, característica explorada en la presente investigación.

V. Paralelismo mediante SIMD

SIMD (*Single Instruction, Multiple Data*) es una tecnología de Intel encontrada en procesadores de arquitectura x86 que permite aplicarle el mismo conjunto de instrucciones a diferentes conjuntos de datos, lo que permite alcanzar paralelismo y obtener tiempos de ejecución significativamente menores. [8]

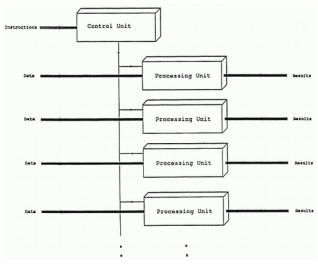

Fig. 1. Diagrama de funcionamiento de SIMD. [8]

Los procesadores Intel que soportan dicha tecnología presentan 8 registros SIMD, en el caso de procesadores de 32 bits, y 16 registros para procesadores de 64 bits. Cada registro SIMD es de 128 bits y su finalidad es contener más de un dato y aplicar instrucciones de manera paralela a cada dato en el registro.

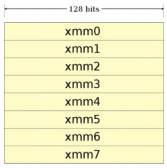

Fig. 2. Registros SIMD para un procesador Intel x86 de 32 bits. [1]

A. *Intel Intrinsics para SSE*

Intel Intrinsics es una biblioteca para C y C++ que permite realizar llamadas a funciones desde C/C++ las cuales están mapeadas directamente a alguna instrucción de ensamblador x86, lo que permite realizar operaciones de bajo nivel sin salir del entorno de programación usual, dando más control al programador.

Dentro de los Intrinsics se encuentra una categoría de instrucciones llamadas SSE (*Streaming SIMD Extensions*) que permite implementar SIMD dentro de C y C++. SSE incluye nuevos tipos de datos que representan registros SIMD, dichos tipos de datos son *__m128*, *__m128d* y *__m128i*, los cuales tienen 128 bits de extensión y se pueden distribuir entre enteros o flotantes de diferente largo. La finalidad de dichos tipos de datos es ser utilizados en operaciones SIMD y **no deben ser usados nunca** en operaciones regulares de C/C++ [4].

Debido a que SIMD navega la memoria en bloques de 16 bytes, es necesario que la memoria esté alineada en 16 bytes, de lo contrario se paga una severa multa de desempeño.

Las instrucciones SSE presentan una convención para sus nombres. Todas están precedidas por el prefijo *_mm_* seguido por la operación que realiza (según la instrucción x86 a la que está mapeada) seguido de un sufijo que indica los tipos de datos con los que opera, tal como se ilustra en las tablas 1 y 2.

TABLE I. Primeras letras del sufijo de instrucciones SSE. [4]

Primeras letras	Significado
p	Packed
ep	Extended packed
s	Escalar

TABLE II. Resto de las letras del sufijo de instrucciones SSE. [4]

Resto	Significado
s	Flotante de precisión simple.
d	Flotante de precisión doble.
i[n]	Entero con signo de [n] bits.
u[n]	Entero sin signo de [n] bits.

VI. Implementación del problema

Se implementó una aproximación para las funciones coseno y logaritmo natural y sus enésimas derivadas. La aproximación se realizó mediante series de Taylor truncadas cuya evaluación de polinomios se implementó tanto con el esquema de Horner como con el de Estrin, sin embargo el último se reimplementó con SSE para mejorar los tiempos de ejecución. Los coeficientes de la serie de obtienen en tiempo de compilación mediante técnicas de metaprogramación. Se midieron los tiempos de ejecución y se compararon con las implementaciones secuenciales. Posteriormente se graficó el error de la aproximación con una implementación en C++ de la biblioteca matplotlib de Python.

Referencias

[1] Abecassis, F. (2011, 30 de Septiembre). *Getting started with SSE* Recuperado de https://felix.abecassis.me/2011/09/cpp-getting-started-with-sse/

[2] Chapra, S. & Canale, R. (2007). *Métodos numéricos para ingenieros* (5 ed.) México DF: McGraw-Hill.

[3] Evers, B. (2014). *Matplotlib-cpp* [Software]. Berlin: Independiente.

[4] Intel Corporation (2007). *Intel C++ Intrinsic Reference* Manuscrito no publicado.

[5] Khan, S. (2010). *Parallel Evaluation Of Fixed-Point Polynomials* Linköping: Linköpings universitet.

[6] Rentzsch, J. (2005, 08 de Febrero). *Data alignment: Straighten up and fly right* Manuscrito no publicado.

[7] Reynolds, G. (2017). *Investigation of different methods of fast polynomial evaluation* Edimburgo: University of Edinburgh.

[8] Tonteri, T. (2009, 22 de Septiembre). *A practical guide to SSE SIMD with C++* Recuperado de http://sci.tuomastonteri.fi/programming/sse

[9] Xu, S. (2013). *Efficient Polynomial Evaluation Algorithm and Implementation on FPGA* Singapur: Nanyang Technological University.